Reclaiming Privacy for Smartphone Applications (Revised Version) *

Emiliano De Cristofaro[1], Anthony Durussel[2], Imad Aad[2]

[1] University of California, Irvine

[2] Nokia Research Center, Lausanne, Switzerland

Abstract

The scope of mobile phones has skyrocketed in recent years to such an extent that smartphone sales are expected to surpass those of PCs by the end of 2011. Equipped with relatively powerful processors and fairly large memory and storage capabilities, smartphones can accommodate increasingly complex interactive applications. As a result, the growing amount of sensitive information shared by smartphone users raises serious privacy concerns and motivates the need for appropriate privacy-preserving mechanisms.

In this paper, we present a novel architecture geared for privacy-sensitive applications where personal information is shared among users and decisions are made based on given optimization criteria. Specifically, we focus on two application scenarios: (i) privacy-preserving interest sharing, i.e., discovering shared interests without leaking users' private information, and (ii) private scheduling, i.e., determining common availabilities and location preferences that minimize associate costs, without exposing any sensitive information. We propose efficient yet provably-private solutions, and conduct an extensive experimental analysis that attests to the practicality of the attained privacy features.

1 Introduction

The increasing dependence on anywhere-anytime availability of information has made smartphones commensurably more influential in people's computing ecosystems. Many tasks once performed exclusively on PCs have now branched out to phones and have successfully accommodated device constraints, such as display, connectivity, and power [32]. Smartphone applications are progressively enabling sophisticated interactions among people. Nowadays, users share information, for instance, for real-time social networking [39], to discover nearby friends (e.g., using Google Latitude, Foursquare, Gowalla, etc.), to exchange recommendations [29], to build P2P networks [8], or even for work-related purposes [33].

Beneath the proliferation of information exchange lays an extremely rich source of data: mobile phones, typically accompanying users 24/7, learn significant amounts of information about their owners and their environment. Business models behind many of today's free web applications, such as social network websites or search engines, heavily rely on data *in the clear*, collected from the users and later used to offer context-based services and recommendations. Yet, as the amount (and sensitiveness) of shared information increases, so do related privacy concerns.

In this paper, we focus on the design of collaborative applications involving participants—with limited reciprocal trust—willing to share *sensitive information from their smartphones* and use it to (cooperatively) perform operations without endangering their privacy. Previous work has attempted to address this problem by using distributed cryptographic primitives, such as generic Secure Multi-Party Computation [25] or Private Set Intersection (PSI) [21, 35, 15, 27, 31, 14, 3]. While some of these solutions are quite efficient for the two-party

*A preliminary version of this paper appears in [13]. This version extends description and analysis of proposed techniques. Furthermore, in Section 4.3.3, it discusses in more details how the PrivSched-v2 algorithm trades off some privacy guarantees (w.r.t. the server) for increased efficiency, in comparison to the PrivSched-v1 counterpart.

case, the multi-party variants are not practical enough for large-scale deployment. For instance, the most efficient multi-party PSI, [35], incurs computational complexity—specifically, a number of long modular exponentiations—*quadratic* in the size of participants' inputs. Alternatively, statistical methods for protecting privacy have also been used [30, 12], yet they do not achieve provable privacy guarantees and generally require a fully-trusted party to produce privacy-preserving statistical digests. Such a trust assumption is often unrealistic, since all sensitive information is concentrated at this party. Beyond its obvious privacy implications, this modus operandi imposes liability issues with regard to participants' private data and concerns about possible data loss, subpoena, etc.

We explore provably-secure (efficient) cryptographic techniques assisted by a *semi-trusted* server, which acts as a bulletin board service. Observe that this entity is never trusted with participants' data itself – i.e., it learns no information about users' inputs or outputs. The server only obtains (and processes) *encrypted* data and it is trusted only not to collude with other parties. We envision that this entity may be implemented by public services, (e.g., Nokia Ovi [2] or Amazon EC2 [1]), without disclosing any data in the clear. (Users can select or set up their own preferred servers). Note that the use of the server remarkably minimizes the overhead incurred by mechanisms for privacy protection in smartphone applications, but does not sacrifice solid privacy guarantees.

1.1 Roadmap

Motivated by the arguments above, we introduce an appropriate architecture that involves several smartphone users and a semi-trusted server (Section 2). We then present the design, implementation, and evaluation of several privacy-enhanced applications. We guarantee privacy by modeling efficient cryptographic tools, which function as privacy "shields" that protect users from disclosing more than the required minimum of their respective sensitive information.

First, we address the generic problem of letting smartphone users share their *interests*, while maintaining their privacy (Section 3). We overview several application examples, such as discovering matching locations, routes, availabilities, etc., without exposing any other information beyond the matching interests, or calling "polls" to find most popular preferences, without revealing anything about other preferences or single user's choices. We present a generic efficient cryptographic solution that addresses all these scenarios.

Next, we consider assigning *costs* to the various interests, and we focus on minimizing the overall aggregated cost in a privacy-preserving way. We consider, as an example, the problem of Private Scheduling, where a group of smartphone users want to schedule a meeting while preserving the privacy of their calendars (Section 4). We are stimulated by ongoing projects aiming at automatically filling mobile phones' calendars with users' activities (e.g., based on emails, text messages, routes). Also, widespread calendar applications, such as Apple iCal, Microsoft Outlook, and Google Calendar, currently provide software counterparts for smartphones. We propose three solutions that target different scenarios, corresponding to different efficiency, privacy, and system requirements.

Finally, we analyze the performance of proposed constructs (Section 5), discuss related work (Section 6), and conclude with some future work items (Section 7). Appendix A reviews the Paillier Cryptosystem [40], used in two of our protocols, while Appendix B overviews the Private Set Intersection construction from [31], also used by one of our techniques.

2 Preliminaries

In this section, we model our system, by describing players, privacy properties, and trust assumptions.

2.1 Players

Our system consists of:

- N smartphone users, $P_1, \ldots, P_N$, called *participants*. Participants $P_1, \ldots, P_N$ hold private information, denoted, resp., with $x_1, \ldots, x_N$. In the rest of the paper, we assume that $N > 2$.
- A public *server*, S, that generates no input.

Participant P_1 is called *Initiator*, to denote that she is starting the interaction. Depending on the application, P_1 might be performing additional operations compared to other participants.

2.2 Privacy-preserving Computation

We assume that participants, $P_1, \ldots, P_N$, wish to jointly compute some function $f(x_1, \ldots, x_N)$, without revealing any information about their private input, $(x_1, \ldots, x_N)$, beyond what can be deduced by obtaining the output of the function. Note that one or all parties may receive the output of the computation, depending on the application. Server S assists the participants in the function computation, but learns no information about either $(x_1, \ldots, x_N)$ or $f(x_1, \ldots, x_N)$. In other words, S receives and processes only encryptions of participants' inputs. Specifically, we define the following privacy requirements:

- ***Participant's Privacy w.r.t. Server:*** Privacy of each participant P_i, running on input x_i, is guaranteed w.r.t. server S if no information about x_i is leaked to S. Privacy is considered as the probabilistic advantage that S gains from obtaining encrypted inputs toward learning participants' actual inputs. Formally, we say that the algorithm A is *Private w.r.t. Server* if no polynomially bounded adversary $\mathcal{A}$ can win the following game with probability non-negligibly over 1/2. The game is between $\mathcal{A}$ (playing the role of the server) and a challenger Ch:

  1. Ch executes setup operations and computes public parameters (if any).
  2. $\mathcal{A}$, on input the public parameters, selects two inputs (w_0, w_1).
  3. Ch picks a random bit $b \leftarrow \{0, 1\}$ and interacts with $\mathcal{A}$ by following the computation on behalf of a participant, on input the public parameters and private input w_b.
  4. $\mathcal{A}$ outputs b' and wins if $b' = b$.

  We anticipate that, since S only receives encrypted inputs, privacy will be reduced to the security of the encryption algorithm.

- ***Participant's Privacy w.r.t. Other Participants:*** Privacy of each participant P_i, running on input x_i, is guaranteed w.r.t. other participants $P_j (\forall j \neq i)$, if no information about x_i is leaked to P_j, beyond what can be inferred from $f(x_1, \ldots, x_N)$. Privacy is considered as the probabilistic advantage that P_j gains from the protocol execution in learning P_i's input. Specifically, we define the advantage of P_j of identifying the input of any P_i, i.e., w_i (for $P_i \neq P_j$), in the algorithm A as

$$Adv_{P_j}(A; P_i) = \left| Pr_{P_j}[w_i' = w_i] - \frac{1}{2} \right|$$

  where w_i' is P_j's guess of P_i's input w_i.
  We say that the algorithm A is *Private w.r.t. Other Participants* if no polynomially bounded adversary $\mathcal{A}$, playing the role of P_j in A, has a non-negligible advantage $Adv_{P_j}(A; P_i)$ for any $P_i \neq P_j$.

2.3 Trust Assumptions

We assume participants to be *semi-honest*, i.e., they faithfully follow protocol specifications and do not misrepresent any information on their inputs, as per Goldreich's definitions [23]. However, during or after protocol execution, they try to infer additional information about other parties' inputs. Also, we assume that participants

have a minimum degree of knowledge among each other, e.g., they belong to the same organization or community. Thus, they do not have any incentive to deviate from the protocol.

The server is assumed to be semi-trusted, i.e., it follows protocol specifications and does not collude with any participants. It is not trusted with any private data, thus, we require minimal trust in it.

3 Sharing Interests with Privacy

We start by considering the following application examples:

1. A smartphone application provides users with the possibility of calling *privacy-preserving polls*: a user can request to a community of users to indicate their preferences on a topic of interest, in order to jointly determine the most popular choice. Nothing should be revealed about a single user's preferences or other choices beyond the most popular one.
2. Members of a virtual community (e.g., AC Milan fans) want to share their location only to discover if there is at least a given number of members around the same location (e.g., 10 fans in the same pub to watch a given match).
3. A group of UC Irvine commuters want to find out common routes to join a carpooling program, i.e., they agree on sharing rides from their homes (or from a meeting location) to a common destination, e.g., as a work center. Nonetheless, information about their routes should be revealed only for matching commutes.
4. A group of Nokia employees are willing to share their availabilities, e.g., to discover times and locations at which at least 75% of them are available (if any), but do not want to reveal whether or not they are free in any other timeslots.

The four examples above exhibit some similar features: users are not willing to reveal their information wholesale, rather, they want to find only matching interests.

Motivated by these examples, in this section, we introduce the problem of ***Privacy-preserving Interest Sharing*** (PIS). Note that we intentionally describe PIS as a broad notion and we present a generic efficient and provably-private cryptographic construct that targets a diversified set of scenarios.

PIS employs a semi-trusted server—introduced in Section 2—to assist the computation. However, the server is never trusted with any user data (as opposed to a non privacy-preserving approach requiring all users to submit their interests and let the server find the matches). An additional trivial approach would require users to share a symmetric key, use a (deterministic) symmetric-key encryption scheme, and let the server match ciphertexts. However, establishing and managing secret keys is not viable in several settings, like in scenarios 1-3 above. For instance, in (1) the user calling the poll does not even know who is participating. Moreover, if the key is compromised, all interests of all participants would be revealed.

Clearly, deploying actual solutions for the aforementioned problems requires application developers to address specific system technicalities, and additional (potential) security issues, such as authentication, anonymity, DoS prevention, etc., using standard state-of-the-art techniques. Indeed, our efforts are currently focused on the deployment of actual privacy-enhanced applications relying on PIS.

Informal Definition of PIS. We assume N participants, $P_1, \ldots, P_N$, where P_1 acts as the *Initiator* of the protocol. Each participant P_i (for $i \in [1, N]$) holds a list of interests of size m_i. Without loss of generality, we assume that $\forall i, m_i = m$ (i.e., all lists are equal size) to ease presentation. The *interests* of each participant P_i are denoted as $\{b_{i:1}, \ldots, b_{i:m}\}$ – i.e., participants' inputs in the PIS protocol.

The goal of PIS is to find the interests that are shared among at least a ***threshold of τ participants***. Participants' privacy needs to be guaranteed by ensuring that, after protocol execution, the participants do not learn anything

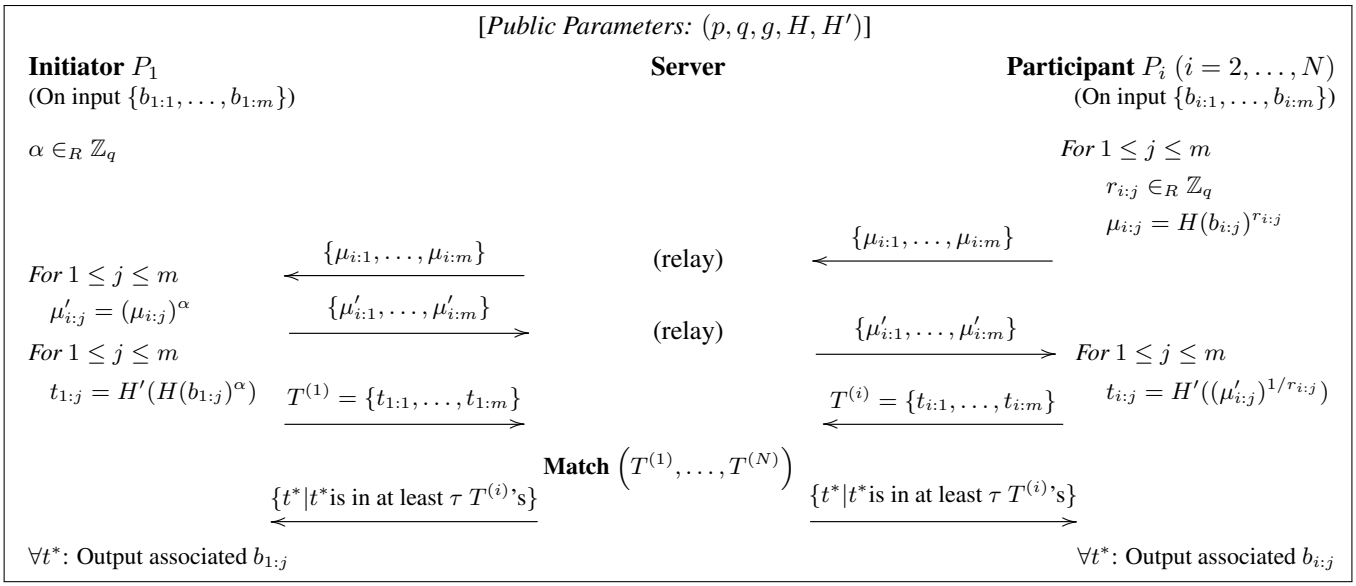


Figure 1: Our Private Interest Sharing (PIS) Protocol.

beyond matching interests. Also, server S learns no information about any participant's input. (Recall privacy definitions from Section 2.2).

3.1 PIS Protocol Specification

Our PIS construction is illustrated in Figure 1 and works as follows:

[1. Setup] The server, S, publishes public parameters (p, q, g, H, H'), where: p, q are prime numbers s.t. $q|p-1$, g is a generator of the subgroup of size q, $H : \{0,1\}^* \to \mathbb{Z}_p^*$, and $H' : \{0,1\}^* \to \{0,1\}^{\gamma}$ (given a security parameter γ) are cryptographic (i.e., collision-resistant) hash functions. The Initiator, P_1, privately picks a random $\alpha \in_R \mathbb{Z}_q$. (Note that all computations below are performed $\bmod p$.)

[2. Interaction] Each participant P_i (for $i \in [2, N]$), on input $\{b_{i:1}, \ldots, b_{i:m}\}$, for each $j \in [1, m]$:

- Picks $r_{i:j} \in_R \mathbb{Z}_q$
- Computes and sends S $\mu_{i:j} = H(b_{i:j})^{r_{i:j}}$

Next, S forwards all received $\mu_{i:j}$'s to P_1, who, in turn, responds to participants (via S) with $\mu'_{i:j} = (\mu_{i:j})^{\alpha}$. Finally, each participant P_i (for $i \in [2, N]$), upon receiving $\mu'_{i:j}$ ($j \in [1, m]$), computes and sends S:

$$T^{(i)} = \left\{ t_{i:j} \mid t_{i:j} = H' \left[(\mu'_{i:j})^{1/r_{i:j}} \right] \right\}_{j \in [1,m]}$$

Observe that $t_{i:j} = H'[H(b_{i:j})^{\alpha}]$. Also, note that, as opposed to $\mu_{i:j}$'s, S does *not* forward $t_{i:j}$'s to any participant.

[3. Matching] First, the Initiator P_1, for $j \in [1, m]$ sends S:

$$T^{(1)} = \{ t_{1:j} \mid t_{1:j} = H' [H(b_{1:j})^{\alpha}] \}_{j \in [1,m]}$$

Next, S identifies all the items t^* which appear in *at least* τ different $T^{(i)}$ sets, and outputs them to the original participants that contributed them.

Finally, these participants learn (threshold) interest matching by associating t^* to values $b_{i:j}$ producing it.

Remark. PIS can (straightforwardly) be applied to the application examples discussed above. For instance, a user may call for a poll on the best bars in the city, e.g., using a social networking application on her smartphones. Every participant in the poll would engage in a PIS computation as described above. At the end of the poll, the server, e.g., the social network provider, outputs the value t^* that appears most times to the participants, who can then reconstruct the most "popular" bar.

Complexity of PIS. The computational complexity of the protocol amounts to $(N \cdot m)$ exponentiations for the Initiator, whereas, all other participants perform $(2m)$ modular exponentiations. We pick p to be 1024-bit long, and q of size 160-bit (with no loss of security). Thus, using short exponents (160-bit), modular exponentiations in the protocol are very efficient. Communication overhead for the server and the Initiator amounts to $(N \cdot m)$ group elements (i.e., 1024-bit) and hash values (i.e., 160-bit using SHA-1 [18]), whereas, for the other participants, the overhead amounts to m group elements and hash values.

3.2 Privacy of PIS

Our PIS construction provides provable-privacy guarantees. Specifically, *Privacy w.r.t. Server* is guaranteed as the server S only receives outputs of the one-way functions H, H', whose inputs cannot be "forged" unless S knows either α (secret to P_1) or some $r_{i:j}$ (secret to P_i's). Thus, if an adversary $\mathcal{A}$ violates Privacy w.r.t. Server, then we can construct another adversary that violates the collision resistance of the hash functions H, H'.

Next, *Privacy w.r.t. Other Participants* immediately stems from security arguments of the Private Set Intersection technique in [31], proven secure under the One-More-DH assumption [5], on which our PIS protocol is based. In other words, if any participant P_j has a non-negligible advantage $Adv_{P_j}(A)$ (defined in Section 2.2), then we can construct an attack to the Private Set Intersection protocol in [31]. We review the [31] protocol in Appendix B.

Recall, however, that [31] only provides a two-party protocol, while our variant extends to multiple parties. Remark that we minimize overall overhead using the semi-trusted public server: indeed, available multi-party PSI techniques [35] require several rounds of computation and computational complexity at least quadratic in the size of participants' inputs.

4 Private Scheduling

In this section, we explore the concept of *Private Scheduling* for smartphones. Recall example (4) from Section 3: a group of employees want to schedule a meeting and select a timeslot such that at least a given number of users are free. We now go a step further: instead of assigning a binary value to time periods or to proposed locations (i.e., available/busy, suitable/unsuitable), we consider *non-binary* costs. For instance, the smartphone can calculate the carbon footprint or the gas cost required to reach a given destination, or how much the user is *tied* to a busy timeslot. Such a flexibility is particularly appealing in the mobile environment, where users carry their device anytime and anywhere. Thus, smartphones can infer their preferences, habits, routes, and assist them in determining availabilities and preferred locations. Therefore, we assume that a cost, between 0 and c_{max}, is assigned to each timeslot and/or location. (In the rest of the paper we refer to "timeslots" only, while referring to timeslots and/or locations.)

Users' calendars potentially contain a high volume of sensitive information. Exposed availabilities could be misused to infer affiliation, religion, culture, or correlated to other users. Hence, our goal is to allow users to find the most suitable timeslot — i.e., the one with the minimum sum of costs — while learning *nothing* about single users' availabilities. Our solutions employ the semi-trusted server introduced in Section 2 to aggregate users' encrypted inputs, but prevent the server from learning any information about them. One user, denoted as the *Initiator*, initiates the protocol. She accepts a slightly increased computational overhead — a reasonable assumption, considering she is the one willing to schedule the meeting. In return, only the Initiator obtains the outcome of the protocol.

4.1 The Private Scheduling Problem

Private Scheduling involves N different participants, $P_1, \ldots, P_N$. P_1 is the *Initiator* of the protocol. Each P_i (for $i \in [1, N]$) maintains a private calendar, divided into m timeslots. Typical timeslot granularities are 30 or 60 minutes (but one can tune it according to users' preferences). Each P_i assigns a cost $c_{i:j}$ ($0 \leq c_{i:j} \leq c_{max}$), for each timeslot $j \in [1, m]$ (e.g., ranging from 0 to 10).

Definition 1 *(Aggregated Cost.) For a given timeslot j, the aggregated cost $ac_j = \sum_{i=1}^{N} c_{i:j}$ denotes the sum of all participants' cost.*

Definition 2 *(Threshold.) A threshold value, τ, depending on c_{max} and N, denotes the maximum acceptable aggregated cost to consider a timeslot to be* suitable. *We consider $\tau = f(c_{max}, N)$. A typical value could be $\tau = \frac{c_{max}}{2} \cdot N$.*

The goal of Private Scheduling is to output to the Initiator all timeslots with aggregated costs smaller than τ (if any).

4.2 PrivSched-v1

We now present our first solution for Private Scheduling, *PrivSched-v1*. Before, we provide some technical background.

Preamble. PrivSched-v1 relies on the Paillier Cryptosystem [40], a public-key probabilistic encryption scheme that provides additive homomorphism – i.e., the product of two ciphertexts decrypts to the sum of the corresponding plaintexts. We refer to Appendix A for a detailed description.
Following the intuition of [17], additively homomorphic cryptosystems, such as Paillier, can be used to compute *homomorphic minimization* (or maximization), i.e., one can find the minimum of some integers while operating on ciphertexts only, thus, without learning any information on those integers. We extend this technique to obtain the *homomorphic argmin*, i.e., to additionally find which integer corresponds to the minimum. We use a *tagging* system based on powers of 2. Observe that this extension is, to the best of our knowledge, the first attempt in this direction, thus, it can be of independent interest.[1]
We encode integers in a unary system: to represent an integer X, we repeat X times encryptions of 0's. Throughout the rest of the paper, we denote this encoding technique as *vector-based representation (vbr)*:

$$x \xrightarrow[\textbf{vbr}]{} \overrightarrow{X} = [\underbrace{E(0), \ldots, E(0)}_{x \ times}, E(1), E(z), \ldots, E(z)]$$

$E(\cdot)$ denotes encryption using Paillier, and z a random number in the Paillier setting. (This is used for padding). Then, we raise each element of the vbr to the value of a tag – a power of 2. (The tagging is performed on ciphertexts, rather than plaintexts, as it will become clear later in the paper). After tagging, $E(0)$ remains $E(0)$, while $E(1)$ becomes $E(tag)$: this allows to identify which value corresponds to the minimum, after the homomorphic minimization is performed. (Note that that $E(\cdot)$ denotes encryption using Paillier, and z a random number in the Paillier setting).
Observe that vector $\overrightarrow{X}$ has to be large enough to contain each possible domain value. Finally, note that – since the Paillier cryptosystem is probabilistic – the elements $E(\cdot)$ (and the vectors too) are mutually computationally indistinguishable and do not reveal any information about plaintext values.

[1] E.g., the computation of homomorphic argmin may be useful for privacy-preserving data aggregation in sensor networks or urban sensing systems [43].

4.2.1 PrivSched-v1 Protocol Specification

To compute the homomorphic argmin, we use a tagging system over a vector-based representation, performed by each participant. First, the Initiator creates (and transfer to the server S) a vector-based representation of her costs, for each timeslots. Then, S sequentially asks other participants to update vectors with their own costs. (Recall that vectors do not reveal any information about the underlying inputs). Finally, S computes (and transfer to the Initiator) the homomorphic argmin and the Initiator learns the suitable timeslots (if any) upon decryption.

One crucial goal is to minimize computation overhead on the smartphones. Note that vbr's are relatively short, as we deal with small integers if small costs are chosen (e.g., $c_{max} = 10$). Nonetheless, we still want to minimize the number of exponentiations to compute the vbr. To this end, we compute single encryptions of $0, 1, z$, and a random $rand = E(0)$, where encryption of 0 is performed using a random number w, chosen with the same size as the Paillier modulus. We then re-randomize the first element in vbr with a multiplication by $rand$. Next, we update $rand \leftarrow (rand)^{exp}$, where exp is a relatively small random exponent, and we continue with the next element. We describe the details of PrivSched-v1 below. The protocol is also illustrated in Figure 2.

[1. Initialization] First, the Initiator P_1 generates Paillier public and private keys, denoted with PK_1 and SK_1, respectively. (In the rest of this section, all encryptions/decryptions are always performed using these keys, thus, to ease presentation, we omit them in our notation. Note that, if we need to specify the randomness used by the encryption algorithm, we use the following notation: $E(M, R)$ to denote encryption of M under PK_1 using the random value R).
Next, P_1 computes, for each time slot $j \in [1, m]$, the vbr $\overrightarrow{v_j}$:

$$\overrightarrow{v_j} = [\underbrace{E(0), \ldots, E(0)}_{c_{1:j}}, \underbrace{E(1), E(z), \ldots, E(z)}_{\tau - c_{1:j}}]$$

Finally, P_1 sends $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$, along with the identities of the other participants, to S.

[2. Aggregation] After receiving the initial input from P_1, the server S sequentially forwards $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$ to each participant involved in the protocol.
Next, each P_i (for $i \in [2, N]$) adds her cost $c_{i:j}$ to each vector $\overrightarrow{v_j}$ (for $j \in [1, m]$) by shifting the elements of each vector $c_{i:j}$ positions right, and replacing them by $E(0)$:

$$\overrightarrow{v_j} \leftarrow \overrightarrow{v_j} >> c_{i:j} \stackrel{\text{def}}{=} [\underbrace{E(0), \ldots, E(0)}_{c_{i:j}}, v_{j,1}, \ldots, v_{j,\tau - c_{i:j}}]$$

To mask her modifications, P_i re-randomizes the vectors $\overrightarrow{v_j}$'s by multiplying the generic element $v_{j,k}$ by a random $E(0)$. Finally, she sends the updated $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$ back to S.
Remark that this phase is repeated, sequentially, for each participant, $P_2, \ldots, P_N$: at the end S obtains the final $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$ where, for $j \in [1, m]$:

$$\overrightarrow{v_j} = [\underbrace{E(0), \ldots, E(0)}_{ac_j}, \underbrace{E(1), E(z), \ldots, E(z)}_{\tau - ac_j}]$$

[3. Minimization] Upon receiving the final $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$, S computes the homomorphic argmin: first, S raises each element of $\overrightarrow{v_j}$ to 2^j; then, it computes a vector $\overrightarrow{q}$. (The sum of all tags should not exceed the size of the Paillier modulus):

$$\overrightarrow{v'_j} = (\overrightarrow{v_j})^{2^j} = [(v_{j,1})^{2^j}, (v_{j,2})^{2^j}, ..., (v_{j,\tau})^{2^j}]$$

$$\overrightarrow{q} = [q_1, q_2, \ldots, q_\tau] \stackrel{\text{def}}{=} [\prod_{j=1}^{m} v'_{j,1}, \ldots, \prod_{j=1}^{m} v'_{j,\tau}] =$$

$$= [\underbrace{E(0), \ldots, E(0)}_{min}, q_{min+1}, \ldots, q_\tau]$$

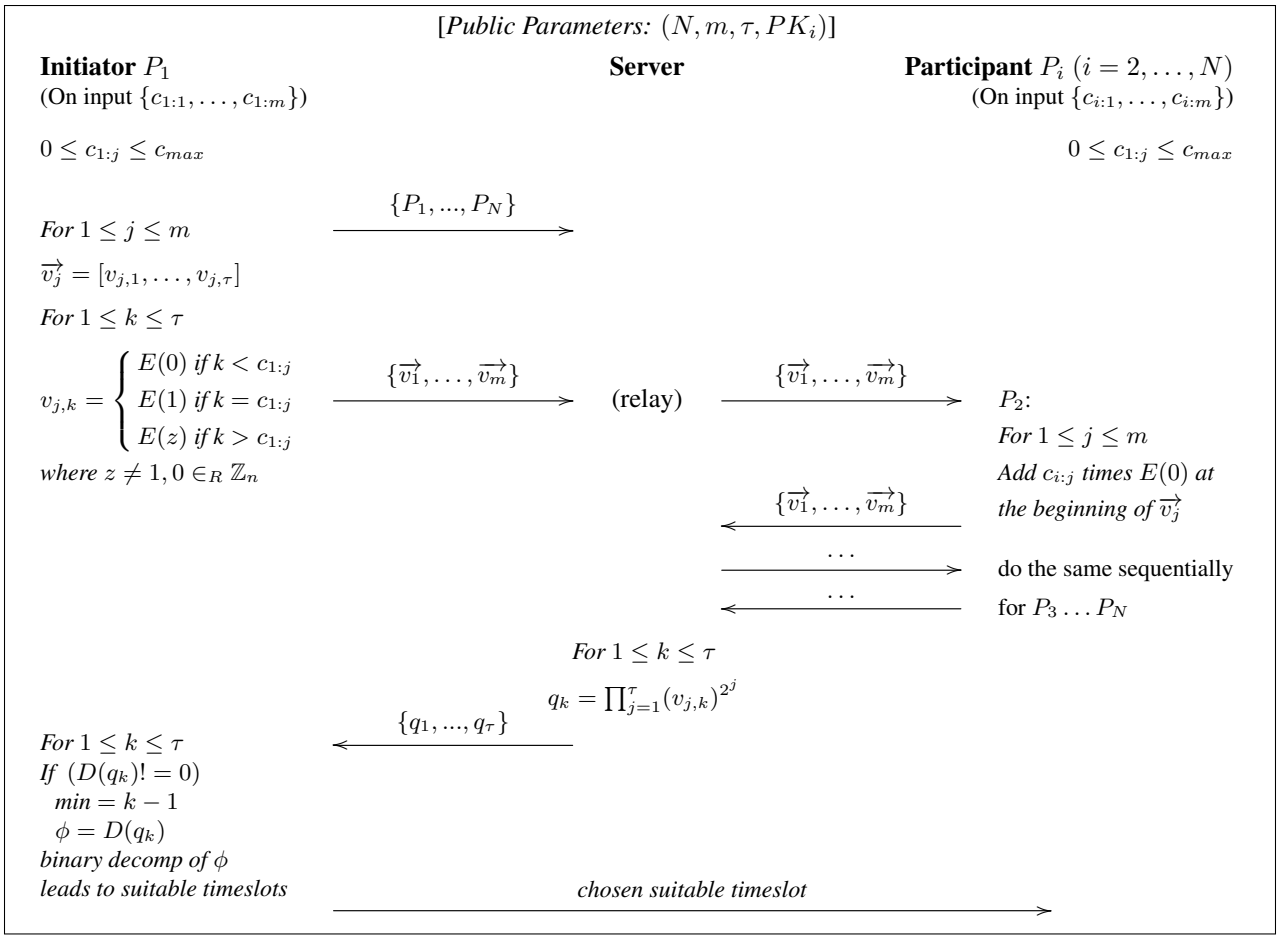


Figure 2: Our PrivSched-v1 Protocol.

Next, S sends $\overrightarrow{q}$ to the Initiator P_1, that decrypts each element of $\overrightarrow{q}$ using SK_1. The minimum aggregated cost corresponds to the number of consecutive 0's in the first positions of $\overrightarrow{q}$. Also, q_{min+1} decrypts to the sum of tags corresponding to the timeslot(s) producing the minimum aggregated cost. We denote this sum with ϕ. P_1 retrieves the index of this timeslot by observing which bits are equal to 1 in the binary decomposition of ϕ. P_1 may additionally retrieve the 2^{nd} minimum timeslot by subtracting $(\phi \cdot z)$ from the non-null decrypted elements of $\overrightarrow{q}$. Iterating this method leads to retrieval of all timeslots with aggregated cost less than τ.
Observe that τ is a system parameter and can be tuned to meet different requirements. Smaller values of τ result into a smaller $\overrightarrow{q}$ vector: this would reduce computations performed by participants and by the server, as well as the total bandwidth overhead. Also, the knowledge of P_1 on aggregated cost values will be limited to fewer timeslots, while the likelihood that the protocol execution terminates with no suitable timeslot would be increased. Therefore, an appropriate choice of τ depends on the specific setting and should be agreed on by the participants.

At the end of the protocol, only P_1 learns the timeslots with aggregated cost smaller than the threshold, and takes appropriate actions to schedule a meeting. Standard encryption techniques can be used by P_1 to multi-cast the meeting invitation to the other participants.

Complexity of PrivSched-v1. During each protocol execution, the Initiator performs 4 Paillier encryptions: $E(1)$, $E(0)$, $E(z)$ and $rand = E(0, w)$, where w is a random value chosen with the same size as the Paillier modulus. To create vector $\overrightarrow{v_1}$, the Initiator selects the encryptions $E(0)$, $E(1)$ or $E(z)$, and multiplies them by

a different $rand$ to perform re-randomization. Thus, the Initiator performs $(m \cdot \tau)$ multiplications and small exponentiations (to create $\{\overrightarrow{v_1}, \ldots, \overrightarrow{v_m}\}$), and at most τ decryptions (to retrieve suitable timeslots). Alternatively, to create the vector, a pool of pre-computed $E(0)$'s can be used: in this case, the Initiator performs 3 encryptions and $(m \cdot \tau)$ multiplications with pre-computed $E(0)$'s. All other participants perform 2 encryptions ($E(0)$ and $rand$), and $(m \cdot \tau)$ multiplications and small exponentiations (to update the vectors). If pre-computations are used, they perform 1 encryption and $(m \cdot \tau)$ multiplications. The server performs $(m \cdot \tau)$ exponentiations for the tagging and $(m \cdot \tau)$ multiplications to create vector $\overrightarrow{q}$. The communication overhead amounts to $(m \cdot \tau)$ ciphertexts (i.e., 2048-bit each) for all participants. Additionally, the Initiator receives τ ciphertexts (in $\overrightarrow{q}$).

4.2.2 Privacy of PrivSched-v1

All vectors are encrypted under the Initiator's public key, thus, neither the participants nor the server can violate the privacy requirements described in Section 2.2, by virtue of the CPA-security of the Paillier cryptosystem [40]. In other words, if *Privacy w.r.t. Server* is not guaranteed, then one can construct an adversary violating the Decisional Composite Residuosity assumption [40].

Then, *Privacy w.r.t. Other Participants* is straightforwardly guaranteed, since participants only get (from the server) the minimum of the aggregated costs and no other information about other participants' inputs.

Given that the server and the Initiator do not collude, the server computes the minimization, *blindly*, i.e., over encrypted data. Collusion between the server and the Initiator may lead to violate other participants' privacy, while a collusion between the server and other participants would be irrelevant, as they could not decrypt. However, if $N-1$ participants colluded with the server against the Initiator, they could recover (potentially) valuable information from the output of the protocol. Collusions can be thwarted using Threshold Cryptography [42] and, specifically, the threshold version of Paillier cryptosystem, presented in [19]. Recall that in a (t, N)-threshold cryptosystem, the private key to decrypt is shared between all the participants. Hence, to decrypt a ciphertext, t over N participants should agree and execute some computations to jointly perform the decryption. Using a threshold version of Paillier cryptosystem ensures that the Initiator, even if colluding with the server, cannot maliciously decrypt more information than she should. Note, however, that at this stage our protocol implementations do not address collusion between participants.

4.3 PrivSched-v2

A potential slow-down in PrivSched-v1 may result from *each* participant computing or updating the vector-based representation (vbr) of her costs. This increases the communication overhead and requires each participant to compute operations sequentially, one after the other. Therefore, we introduce a new protocol variant, called *PrivSched-v2*. Participants encrypt directly their costs without using the vbr, thus, we can perform the aggregation in parallel, instead of sequentially, since we no longer use the vbr at each participant's side. This modification reduces server's waiting time and improves the overall performance. The server relies on a mapping, pre-computed by the Initiator, to transform each aggregated cost into its vbr and perform the homomorphic argmin. Again, our improvements might be of independent interest in the context of homomorphic computation.

4.3.1 PrivSched-v2 Protocol Specification

We present our modified protocol—namely, PrivSched-v2—below. The PrivSched-v2 protocol is also illustrated in Figure 3.

[1. Setup] First, each participant P_i computes public/private keypairs (PK_i, SK_i). Public keys, PK_i, are distributed, before protocol execution, using the server.
The Initiator P_1 computes a mapping, MAP, and sends it to the server S. S will use it during aggregation to

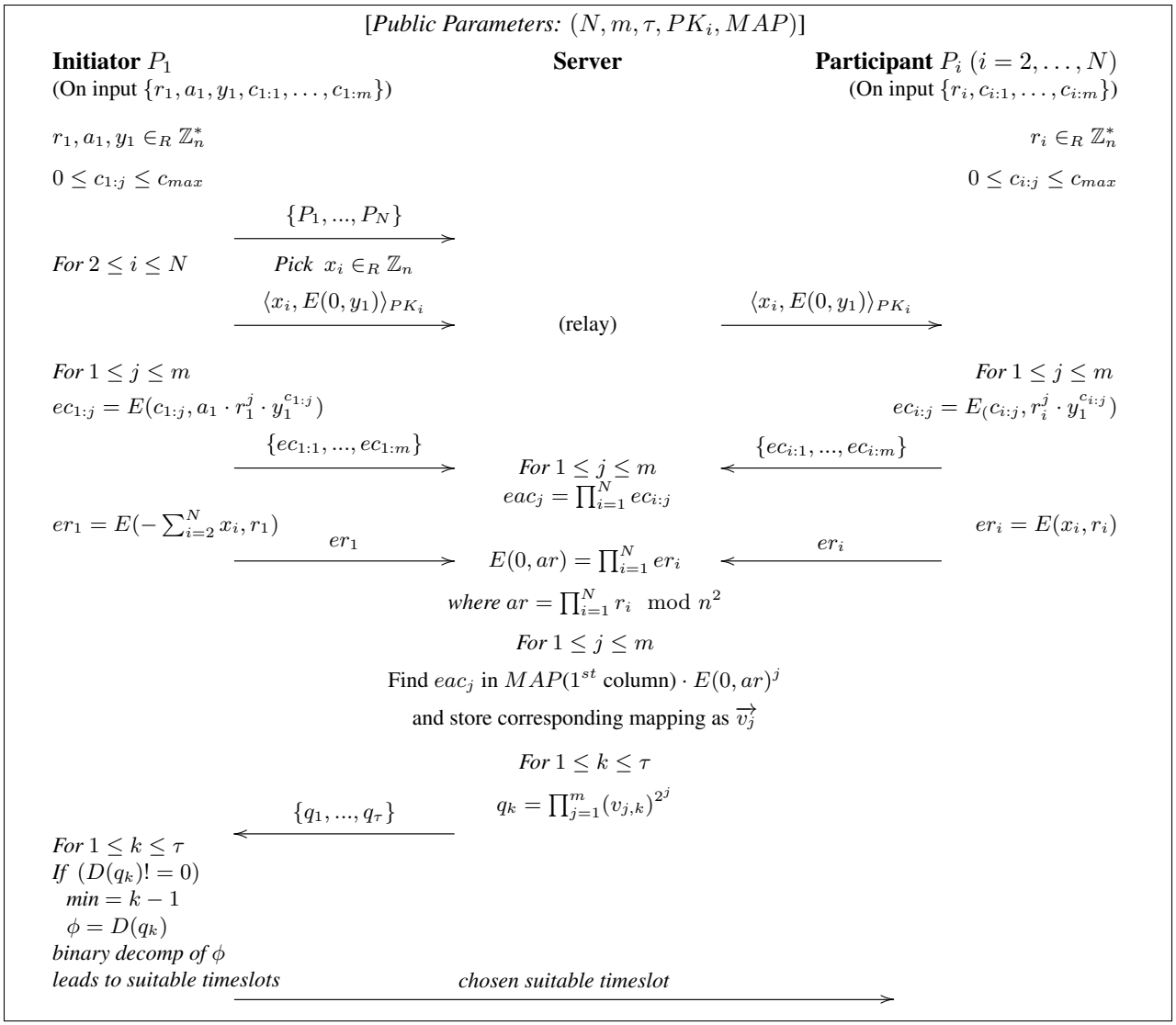


Figure 3: Our PrivSched-v2 Protocol.

transform each aggregated cost into the corresponding vbr. Assuming N_{max} is the maximum number of participants, $\tau_{max} = f(c_{max}, N_{max})$, and (a_1, y_1) are random values in the Paillier setting generated by P_1, MAP is pre-computed by P_1 as follows:

$$\begin{aligned}
E(0, a_1) &\longrightarrow [E(1), E(z), E(z), E(z), \ldots, E(z)] \\
E(1, a_1 y_1) &\longrightarrow [\underbrace{E(0)}_{1}, E(1), E(z), E(z), \ldots, E(z)] \\
\ldots &\longrightarrow \ldots \\
E(\tau_{max}, a_1 y_1^{\tau_{max}}) &\longrightarrow [\underbrace{E(0), E(0), E(0), E(0), \ldots, E(0)}_{\tau_{max}}]
\end{aligned}$$

Note that y_1 and a_1 randomize the mapping and prevent S from learning any private information. y_1 is raised to the value of the aggregated costs in order to randomize the costs differently. Since $(y_1)^0 = 1$, we add the random value a_1 to address the special case where an aggregated cost equals 0. In addition, a_1 simplifies future updates: P_i would only need to change this random value to re-randomize the mapping without performing exponentiations

again. Finally, we let P_1 shuffle the mapping to randomize the position of each aggregated cost. (Recall that encryptions are again computed using PK_1).

[2. Initialization] First, P_1 picks a random r_1, then, for each time slot $j \in [1, m]$, computes and sends S the value $ec_{1:j}$:

$$\begin{aligned} ec_{1:j} &= E\left(c_{1:j},\ a_1 \cdot (r_1)^j \cdot (y_1)^{c_{1:j}}\right) = \\ &= E(c_{1:j}, a_1) \cdot (E(0, r_1))^j \cdot (E(0, y_1))^{c_{1:j}} \end{aligned}$$

Next, P_1 picks $N-1$ random values, $(x_2, \ldots, x_N)$. For each $i \in [2, N]$, P_1 encrypts $\langle x_i, E(0, y_1)\rangle$ under the public key PK_i of each participant P_i and sends them to S, which forwards to the corresponding participant. (Note that we need this encryption to hide these values from the server).
Finally, P_1 computes $er_1 = E(-(\sum_{i=2}^{N} x_i), r_1)$ and sends it to S.

[3. Aggregation] First, each participant P_i generates a random value r_i in the Paillier setting generated by P_1. Then, for each timeslot $j \in [1, m]$, P_i encrypts her cost $c_{i:j}$, using as randomness the decrypted $E(0, y_1)$:

$$\begin{aligned} ec_{i:j} &= E\left(c_{i:j},\ r_i \cdot (y_1)^{c_{i:j}}\right) \\ &= E(c_{i:j}, 1) \cdot (E(0, r_i))^j \cdot (E(0, y_1))^{c_{i:j}} \end{aligned}$$

Then, P_i sends S $\{ec_{i:1}, \ldots, ec_{i:m}\}$ and $E(x_i, r_i)$.
Next, S computes the (encrypted) aggregated cost of each timeslot $j \in [1, m]$, using Paillier's homomorphism:

$$eac_j \stackrel{\text{def}}{=} \prod_{i=1}^{N} ec_{i:j} = E\left(ac_j,\ (ar)^j \cdot a_1 \cdot (y_1)^{ac_j}\right)$$

where $ar \stackrel{\text{def}}{=} \prod_{i=1}^{N} r_i \bmod n^2$ (n being the public Paillier modulus of PK_1). Finally, S reconstructs:

$$E(0, ar) = E(-(\sum_{i=2}^{N} x_i), r_1) \cdot E(x_2, r_2) \cdot \ldots \cdot E(x_N, r_N)$$

[4. Minimization] In this phase, the server S computes the (encrypted) minimum aggregated cost and sends it to P_1. To this end, S first transforms each encrypted aggregated cost (eac_j) into its vbr. Next, S computes the vbr using the mapping MAP and the value $E(0, ar)$, namely, for each timeslot $j \in [1, m]$, S:

(i) Multiplies the 1^{st} column of MAP by $E(0, ar)$ and gets: (recall that the position of each aggregated cost is shuffled in the mapping stored by the server).

$$\begin{aligned} E(0, ar^j a_1) &\longrightarrow [E(1), E(z), E(z), \ldots, E(z)] \\ E(1, ar^j a_1 \cdot y_1) &\longrightarrow [\underbrace{E(0)}_{1}, E(1), E(z), \ldots, E(z)] \\ E(2, ar^j a_1 \cdot y_1^2) &\longrightarrow [\underbrace{E(0), E(0)}_{2}, E(1), E(z), \ldots, E(z)] \\ \ldots &\longrightarrow \ldots \\ E(\tau_{max}, ar^j a_1 \cdot {y_1}^{\tau_{max}}) &\longrightarrow [\underbrace{E(0), E(0), E(0), \ldots, E(0)}_{\tau_{max}}] \end{aligned}$$

(ii) Finds eac_j in MAP and stores the right side of the mapping as $\overrightarrow{v_j}$.

(iii) Increments j and goes back to (i).

Then, S starts the homomorphic argmin using vectors $\overrightarrow{v_j}$, i.e., the vbr of each aggregated cost, leveraging the following tagging technique. The server raises each element of $\overrightarrow{v_j}$ to 2^j (for $j \in [1, m]$):

$$\overrightarrow{v'_j} = (\overrightarrow{v_j})^{2^j} = [(v_{j,1})^{2^j}, (v_{j,2})^{2^j}, ..., (v_{j,\tau})^{2^j}]$$

Next, the server computes the vector $\overrightarrow{q}$ and sends it to P_1:

$$\begin{aligned}\overrightarrow{q} = [q_1, q_2, \ldots, q_\tau] &\stackrel{\text{def}}{=} \left[\prod_{j=1}^{m} v'_{j,1}, \ldots, \prod_{j=1}^{m} v'_{j,\tau}\right] \\ &= [\underbrace{E(0), \ldots, E(0)}_{min}, q_{min+1}, \ldots, q_\tau]\end{aligned}$$

Finally, P_1 decrypts each element of $\overrightarrow{q}$ using SK_1. As in *PrivSched-v1*, observe that the minimum aggregated cost corresponds to the number of consecutive 0's in the first positions of $\overrightarrow{q}$. q_{min+1} decrypts to the sum of tags corresponding to the timeslot(s) producing the minimum aggregated cost. Again, we denote this sum with ϕ. P_1 retrieves the index of this timeslot by observing which bits are equal to 1 in the binary decomposition of ϕ. P_1 may additionally retrieve the 2^{nd} minimum timeslot by subtracting $(\phi \cdot z)$ from the non-null decrypted elements of $\overrightarrow{q}$. Iterating this method leads to retrieval of all timeslots with aggregated cost smaller than τ.

At the end of the protocol, only P_1 learns the timeslots with aggregated cost smaller than the threshold, and takes appropriate actions to schedule the meeting. Again, standard encryption techniques can be used by P_1 to multi-cast meeting invitation to the other participants.

Complexity of PrivSched-v2. During each protocol execution, the Initiator performs, in the worst case, $(c_{max} + 3)$ Paillier encryptions and $(c_{max}+3m)$ multiplications to create $ec_{1:j}$ for $j \in [1, m]$. In addition, the Initiator needs $N-1$ Paillier encryptions to protect $\langle x_i, E(0, y_1)\rangle$ and at most τ decryptions to retrieve suitable timeslots. Each participant performs one decryptions to get $\langle x_i, E(0, y_1)\rangle$, and $(c_{max} + 2)$ Paillier encryptions plus $(c_{max} + 3 \cdot m)$ multiplications to create encrypted costs. The server performs $(m \cdot \tau)$ exponentiations for tagging and $(m \cdot \tau)$ mults to create $\overrightarrow{q}$. The communication overhead amounts to m ciphertexts (i.e., 2048-bit each) for all participants. Additionally, the Initiator receives τ ciphertexts (in $\overrightarrow{q}$).

4.3.2 Privacy of PrivSched-v2

Participants only receive $E(0, y_1)$ and $E(x_i)$, encrypted under the Initiator's public key, thus, similar to PrivSched-v1, they cannot violate the privacy requirements described in Section 2.2, by virtue of the CPA-security of the Paillier cryptosystem [40]. The Initiator gets the vector $\overrightarrow{q}$ containing only suitable timeslots (i.e., whose aggregated cost is smaller than τ). Considerations about possible collusion are the same as in PrivSched-v1, thus, we do not repeat them here.

4.3.3 Potential Disclosure of Aggregated Costs

Compared to PrivSched-v1, PrivSched-v2 trades off some privacy guarantees (w.r.t. the server) for increased efficiency. Since equal aggregated costs may appear at the same line of the mapping, the server could detect repeated aggregated costs. Considering that participants provide weekly or monthly calendar, the highest costs is probably used for nights, weekends, holidays, and busy timeslots are most likely the ones appearing the most. Therefore, the server could easily infer timeslots with the highest cost counting the number of collisions in the mapping. One possible countermeasure is to use a secret permutation of the timeslots, known only to the participants. This way, information obtained by the server is obfuscated and she cannot get any information about date/time of repeated timeslots. Note that removing nights, weekends, or holidays, will also modify the distribution

of busy timeslots which will tend to be closer to the most available ones, thus, reducing the probability of correctly guessing which timeslots correspond to the highest cost.

We have already pointed out the above trade off in [13], nonetheless, in this version, we also discuss another way that a malicious server could infer all aggregated costs. Note, in fact, that the ciphertexts in MAP, published by P_1, depend from each other; specifically, in (un-shuffled) MAP, all pairwise consecutive values have the same ratio. Therefore, the server can attempt to un-shuffle MAP by trying all possible shuffling until it finds one where all ratios are equal. Once, it has reversed the permutation, the server can identify aggregated costs for each timeslot. While the above approach would require an exponential amount of work (i.e., try all possible shuffling), the server could instead mount the following un-shuffling attack. Observe that, for each pair (ρ_i, ρ_j) in the un-shuffled vector of aggregated costs, it holds $\rho_i/\rho_j = (g * {y_1}^n)^{i-j} \mod n^2$ (due to Paillier encryption properties). There are $O(\tau_{max}^2)$ of such pairs. When the server receives the shuffled vector, it can compute all pairs as described above. Note that the server finds out that 2 values occur once ($i-j = \tau_{max}$ or $-\tau_{max}$), 2 that occur twice ($i-j = \tau_{max}-1$ or $1-\tau_{max}$), and so on, up to 2 values that occur τ_{max} times ($i-j = 1$ or -1). The server also recognizes that pairs (ρ'_i, ρ'_j) in the shuffled vector whose quotients are a value occurring τ_{max} times and for a chain such that $1 = \rho'_{j0}\rho'_{i1} = \rho'_{j1}/\rho'_{i2} = \ \ldots \ = \rho'_{j\tau_{max}-1}/\rho'_{i\tau_{max}}$. Then, it either holds $\rho'_{ik} = \rho_k$ or $\rho'_{ik} = \rho_{\tau_{max}-k}$ (for $k \in [0, \ldots, \tau_{max}]$), thus, the server can identify either the permutation or its reverse. Again, note that a possible countermeasure it to simply use a secret permutation of the timeslots, known only to the participants, so that the server could only infer aggregated cost but not the corresponding timeslots.

4.4 Symmetric-Key Private Scheduling: S-PrivSched

The PrivSched-v1 and v2 protocols involve a limited yet non-negligible number of public-key cryptographic operations. Although our experimental analysis (presented in Section 5) provides encouraging performance results for PrivSched-v1 and v2, we now present yet another construction, *S-PrivSched* (Symmetric PrivSched), which only involves symmetric-key operations, and reduces computational and communication overheads. Our intuition is the following: Participants establish a shared secret, e.g., the Initiator broadcasts it using an "out-of-band" channel, such as SMS, or she encrypts it using the public key of each of the other participants. Then, we use an additively homomorphic symmetric-key cryptosystem (proposed in [10]) to perform cost aggregation.

Preamble. S-PrivSched leverages the cryptosystem in [10], a tailored modification of the Vernam cipher [44] to allow plaintext addition to be done in the ciphertext domain, proven to achieve security (more precisely, indistinguishability) against chosen plaintext attacks (IND-CPA). Below, we review its algorithms:

- *Encryption:*
  1. Represent the message msg as an integer $m \in [0, M-1]$, where M is the modulus. (See below).
  2. Let k be randomly generated keystream, where $k \in [0, M-1]$.
  3. Compute $ct = Enc_k(m) = (m + k) \mod M$.
- *Decryption:* $Dec_k(ct) = (ct - k) \mod M = m$
- *Addition of ciphertexts*: Let $ct_1 = Enc_{k_1}(m_1)$, $ct_2 = Enc_{k_2}(m_2)$. Aggregated ciphertext: $ct = ct_1 + ct_2 \mod M = Enc_k(m_1 + m_2)$ (where $k = k_1 + k_2$).

Observe that M needs to be sufficiently large: $0 \leq msg < M$, or $\sum_{i=1}^{N} m_i < M$ if N ciphertexts are added. Also note that, although above we assume k to be randomly generated at all times, one can generate a single master key K and derive successive keys as the output of an appropriate pseudorandom function [24], or the output of a length-preserving hash function (as shown in [10]).

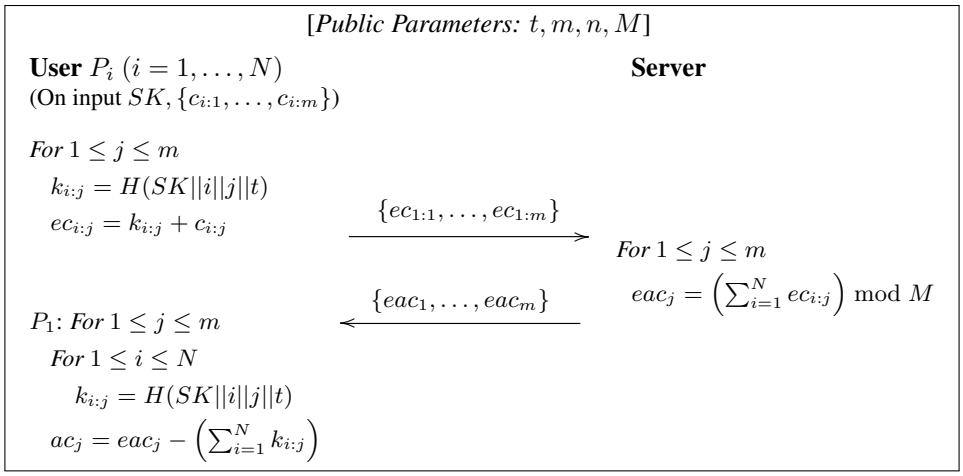


Figure 4: Our S-PrivSched Protocol.

4.4.1 S-PrivSched Protocol Specification

We now present S-PrivSched (also illustrated in Figure 4). We use the same system model introduced for the PrivSched-v1 and v2 constructs, thus, we do not repeat it here.

[1. Setup] The Initiator, P_1, selects $M > (c_{max} \cdot N_{max})$, where c_{max} is the maximum cost participants may associate to a timeslot and N_{max} the maximum number of participants.
P_1 also selects a random $SK \in [0, M-1]$ and broadcasts SK and a nonce t to participants $P_2, \cdots, P_N$ over an out-of-band channel. In this version of the protocol, we assume that SK is sent over SMS, thus, we assume the cellular network operator does not eavesdrop SMS traffic and collude with the (scheduling) service provider. One can also relax this assumption by using the public keys of all other participants to encrypt SK and share it over insecure channels. However, this comes at the cost of $N-1$ additional asymmetric encryptions.

[2. Initialization] Each participant (Initiator included), P_i (for $i \in [1, N]$), for each timeslot $j \in [1, m]$, computes:

- $k_{i:j} = H(SK||i||j)$
- $ec_{i:j} = k_{i:j} + c_{i:j}$

and sends $\{ec_{i:1}, \ldots, ec_{i:m}\}$ to the server S.

[3. Aggregation] Upon receiving $\{ec_{i:j}\}$ $i \in [1, N], j \in [1, m]$, S aggregates the costs, i.e.:

- Computes $eac_j = (\sum_{i=1}^{N} ec_{i:j}) \bmod M \; \forall j \in [1, m]$
- Sends $\{eac_1, \ldots, eac_m\}$ to P_1

[4. Minimization] P_1 obtains the aggregated costs upon decryption:

- $k_{i:j} = H(SK||i||j) \; \forall i \in [1, N], j \in [1, m]$
- $ac_j = ec_j - (\sum_{i=1}^{N} k_{i:j}) \; \forall j \in [1, m]$

Finally, P_1 obtains aggregated costs for each timeslot: she computes the timeslot(s) with minimum cost and takes appropriate actions to schedule the meeting.

Complexity of S-PrivSched. All the participants only perform symmetric key operations, specifically, $(N \cdot m)$ decryptions/subtractions (Initiator) and m encryptions/additions (participants). Communication overhead amounts

to m ciphertexts for each participant and $N \cdot m$ for the server.

4.4.2 Privacy of S-PrivSched

Privacy of S-PrivSched stems from the security of the cryptosystem in [10]. *Privacy w.r.t. the Server S* is guaranteed as S only receives ciphertexts. Also, *Privacy w.r.t. Other Participants* is straightforward since participants never receive other participants' costs. Also, P_1 only obtains aggregated costs. Compared to Paillier-based PrivSched constructs, however, P_1 obtains aggregated costs for *all* timeslots. Indeed, we do not know how to let the server compute the minimization using the homomorphic symmetric-key cryptosystem. Considerations and possible countermeasures about potential collusion are somewhat similar to PrivSched-v1, thus, we do not repeat them here.

5 Performance Analysis

All proposed protocols were implemented on Nokia N900 smartphones. We developed a prototype application for Private Scheduling, instantiating PrivSched-v1 and v2, S-PrivSched, and we used PIS to implement a threshold binary version of Private Scheduling (see scenario (4) in Section 3). Recall that the four algorithms provide (slightly) different privacy properties (reviewed below) and require somewhat different system settings (e.g., key management), which lead to differing computational and communication costs. Thus, our goal is not to compare their performance, rather, to assess their practicality for real-world deployment and to provide an indication of the overhead experienced by users.

In our prototypes, we used the Qt framework [38] and the open-source cryptographic libraries libpaillier [6] and libgmp [20], on several Nokia N900 devices (equipped with a 600 MHz ARM processor and 256 MB of RAM). We also used a Dell PC with 2.27 GHz CPU (16 cores) and 50 GB of RAM to instantiate the semi-trusted server. We ran tests with an increasing number of participants (ranging from 2 to 8) and a fixed number of timeslots, i.e., $7 \cdot 24 = 168$ to cover one week with one-hour granularity, with randomly-generated calendars. We define $c_{max} = 10$ and $\tau = \frac{c_{max}}{2} \cdot N = 5N$. In our tests, we used a local 802.11 Wi-Fi network.

For every algorithm, we measured the *processing time* of the server, the Initiator, and of a single participant. The latter is computed as the average of processing times of all participants, excluding the Initiator. We also evaluated the *communication overhead*. Results are averaged over 100 iterations. We also ran tests for a baseline protocol providing no privacy (i.e., calendars were transmitted to the server, which computed the minimization in the clear).

Results are plotted in Figure 5, 6, and 7. The top row measures bytes exchanged (using logarithmic scale), bottom row – processing times in milliseconds. Confidence intervals were very small and omitted for visibility. (Standard deviation was smaller than 282 bytes and $280ms$).

We observe that:

1. As expected, PrivSched-v1, compared to its most similar counterpart (PrivSched-v2), incurs an increased bandwidth overhead, that grows with the number of participants. Therefore, we recommend its use only in settings where participants do not want the server to learn that some timeslots may have equal aggregated costs.

2. PrivSched-v2 incurs a reasonable overhead and scales efficiently even when the number of participants is increasing. Nonetheless, recall that the mapping creates a potential privacy degradation. Also, some limited information has to be pre-exchanged among the participants.

3. S-PrivSched incurs very low computational and communication overhead — almost negligible compared to the baseline protocol with no privacy. However, it always reveals aggregated costs of *all* timeslots to the Initiator, and requires the distribution of a shared secret (e.g., via SMS).

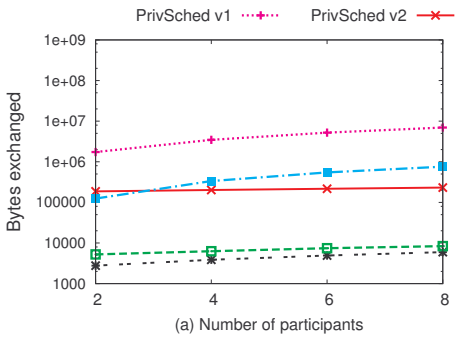


Figure 5: Initiator

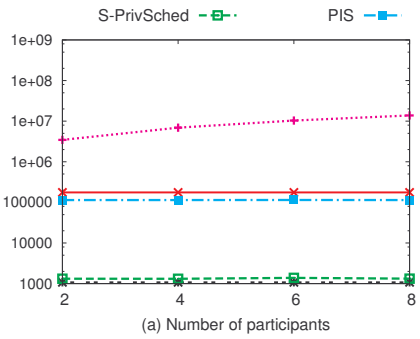


Figure 6: Each participant

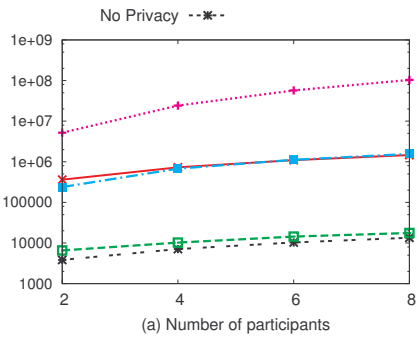
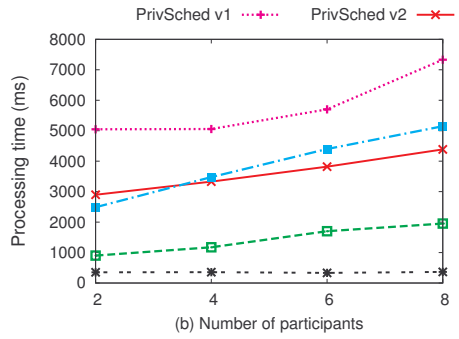
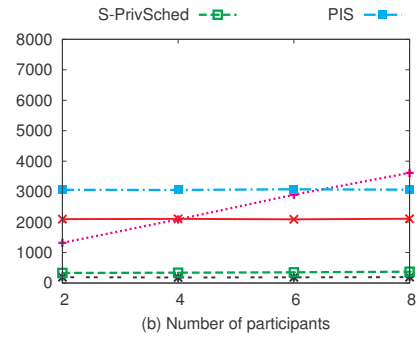
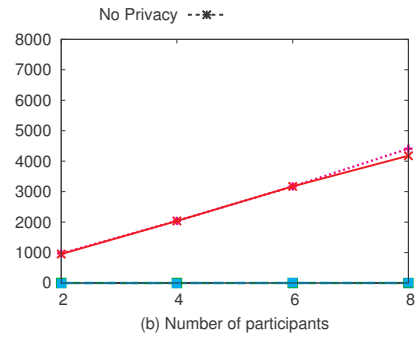


Figure 7: Server







4. PIS addresses several different applications beyond scheduling; nonetheless, it is worth observing that it is practical enough for actual deployment. Indeed, the computational and communication overheads are independent of the number of participants, and incur a constant overhead, except for the Initiator and the server — a reasonable assumption considering that the Initiator is the one willing to start off the protocol.

We conclude that our implementations, though achieving different privacy properties in different system settings, are practical enough for deployment. Indeed, even the most computationally demanding protocols, such as PrivSched-v1-2, require only a few seconds in most realistic settings.

6 Related Work

In the analysis of related work, we first discuss related cryptographic constructs addressing multi-party computation problems. Next, we analyze techniques employing third-party services, and, finally, solutions for the problems of private scheduling and privacy-preserving interest sharing.

Related Cryptographic Primitives. *Secure Multi-Party Computation* (MPC) [25] allows several players, each equipped with a private input, to compute the value of a public function f over all inputs. Players only learn the output of f, and nothing else (beyond what revealed by the computation). Generic MPC would indeed solve all the problems we consider, however, it is well-known that MPC involves several rounds of computations, as well as computational and communication costs far too high to be deployed on smartphones, while special-purpose protocols are generally much more efficient.[2] *Multi-party Private Set Intersection* [35] allows several players to privately compute the intersection of their private sets, such that they learn nothing beyond the intersection. Thus far, however, only solutions limited to the two-party set intersection have achieved practical efficiency [21, 35, 27, 15, 31, 14]. Whereas, constructs for multiple players require a number of long exponentiations quadratic in the

[2] For instance, remark that the communication complexity of MPC grows quadratically with the number of participants.

size of the sets [35] – well beyond the requirements of our smartphone setting. Further, PSI constructs cannot be used for the non-binary private scheduling problem. Finally, note that, to the best of our knowledge, there is no known construct for a private *threshold* set-intersection problem, which would be relevantly close to PIS, but only for threshold set-union [35].

Server-assisted Computations. Over the last years, semi-trusted third parties have been employed to assist privacy-preserving computations. We do not consider naïve solutions using fully trusted third parties (whereto all players surrender their inputs), or requiring the existence of specific hardware or devices, such as a Trusted Platform Modules. (Semi-trusted parties are only assumed not to collude with other players). Following Beaver's intuition [4], [9] introduces a semi-trusted third party to obliviously compare two numbers (e.g., to solve the millionaire's problem [45]). [16] uses the same intuition to solve the scalar product problem. Note, however, that these solutions are only designed for two parties and it is not clear how to adapt them to a multi-party setting. Also, to the best of our knowledge, there is no work exploring such intuition in the context of smartphone applications.

Private Scheduling. To the best of our knowledge, there is no solution targeting the private scheduling problem in the setting of smartphone users. Prior work includes distributed constraint satisfaction in a fully-distributed approach [49, 46, 36, 34, 28]. These solutions incur high computation and communication overhead and are unpractical for mobile environments. Recently (and independently from our work), a solution to a ***binary*** Private Scheduling problem (based on homomorphic encryption) has been presented for smartphone users leveraging a semi-trusted server [7].

Private Interest Sharing. Besides primitives discussed above, several techniques have focused on problems similar to Private Interest Sharing. The work in [48] proposes solutions to the *nearby-friend* problem, i.e., to let two users learn whether their distance is smaller than a given radius. It uses homomorphic encryption [40] to compute algebraic operations over encrypted data. One of the proposed solutions, Louis, relies on a third user to assist computation and reduce overhead, i.e., it acts as a semi-trusted party. Thus, Louis appears very similar to applying PIS to the nearby-friend problem. However, it is not clear how to extend Louis to a multi-party setting, e.g., to learn whether (at least) τ friends are nearby. Next, although in Louis input locations are two-dimension coordinates, while we only consider locations mapped to tags or cells, Louis involves more communication steps (four vs. two in PIS) and more expensive cryptographic operations (Paillier encryptions vs 160-bit exponentiations). Solutions that do not involve a semi-trusted party, such as Lester [48], incur much higher overhead and do not scale to multiple users, even if locations are mapped to cells, such as in Pierre [48], Wilfrid [47], and NFP [11].

Also, the recent position paper in [41] discusses how Location-Based Social Applications (LBSAs) should process friends' location coordinates only in their encrypted form. Further, some recent results addressed location privacy concerns in geo-aware social networks [22] and proximity-based services [37]. Finally, note that some of the applications envisioned in the context of PIS or Private Scheduling (e.g., polls) also resemble *e-voting* problems [26]. In reality, however, e-voting involves several different entities with specific roles and has very different requirements.

7 Conclusion & Future Work

In this paper, we showed how privacy can be efficiently and effectively protected in smartphone applications that deal with users' personal information. We described an architecture that allows a number of users to perform privacy-preserving computations assisted by a semi-trusted server and minimize related overhead. The server is not trusted with the data and performs computation blindly over encrypted data.

We presented several application scenarios, and provided for each of them an appropriate solution. In doing so, we encountered and addressed a number of challenges, which may be of independent interest (e.g., the computation of homomorphic argmin). Experimental analysis attested to the practicality of our techniques and led to interesting observations to design next generation's privacy-enhanced smartphone applications.

Our current and future research includes deployment of other applications beyond those discussed in this paper, as well as the optimization of the cryptographic operations.

Acknowledgments. This research has been done when Emiliano De Cristofaro was an intern in Nokia Research Center, Lausanne. We would like to thank Valtteri Niemi for some useful hints in the early stage of this work. Also, we gratefully acknowledge Urs Hengartner for his very helpful feedback and editorial suggestions. Finally, we thank the anonymous reviewer pointing out how to un-shuffle aggregated costs in PrivSched-v2 (see Section 4.3.3).

References

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2, 2010.

[2] Ovi. http://ovi.com/, 2010.

[3] G. Ateniese, E. De Cristofaro, and G. Tsudik. (If) Size Matters: Size-Hiding Private Set Intersection. In *PKC*, 2011.

[4] D. Beaver. Commodity-based cryptography. In *STOC*, 1997.

[5] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2008.

[6] J. Bethancourt. Paillier Library. http://acsc.cs.utexas.edu/libpaillier/, 2010.

[7] I. Bilogrevic, M. Jadliwala, J.-P. Hubaux, I. Aad, and V. Niemi. Privacy-Preserving Activity Scheduling on Mobile Devices. In *Codaspy*, 2011.

[8] C. Blundo and E. De Cristofaro. A Bluetooth-based JXME infrastructure. In *DOA*, 2007.

[9] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *CCS*, 1999.

[10] C. Castelluccia, A. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(3):20, 2009.

[11] S. Chatterjee, K. Karabina, and A. Menezes. A New Protocol for the Nearby Friend Problem. *Cryptography and Coding*, 2009.

[12] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. AnonySense: Privacy-aware people-centric sensing. In *Mobisys*, 2008.

[13] E. De Cristofaro, A. Durussel, and I. Aad. Reclaiming privacy for smartphone applications. In *PerCom*, 2011.

[14] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *Asiacrypt*, 2010.

[15] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, 2010.

[16] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW*, 2002.

[17] L. Ertaul and Vaidehi. Computing Aggregation Function Minimum/Maximum using Homomorphic Encryption Schemes in Wireless Sensor Networks. In *ICWN*, 2007.

[18] Federal Information Processing Standards. Secure Hash Standard. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf, 2002.

[19] P.-A. Fouque, G. Poupard, J. Stern, and cole Normale Suprieure. Sharing Decryption in the Context of Voting or Lotteries. In *Financlal Cryptography*, 2000.

[20] Free Software Foundation. The GNU MP Bignum Library. http://gmplib.org/, 2011.

[21] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt*, 2004.

[22] D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, 2010.

[23] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2004.

[24] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *JACM*, 33(4):792–807, 1986.

[25] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[26] D. Gritzalis. *Secure electronic voting*. Springer, 2003.

[27] C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC*, 2010.

[28] T. Herlea, J. Claessens, B. Preneel, G. Neven, F. Piessens, and B. De Decker. On securely scheduling a meeting. In *IFIP SEC*, 2001.

[29] J. Hong, E. Suh, and S. Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4), 2009.

[30] B. Hore, J. Wickramasuriya, S. Mehrotra, N. Venkatasubramanian, and D. Massaguer. Privacy-preserving event detection in pervasive spaces. In *PerCom*, 2009.

[31] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, 2010.

[32] A. Karlson, S. Iqbal, B. Meyers, G. Ramos, K. Lee, and J. Tang. Mobile task flow in context. In *CHI*, 2010.

[33] A. Karlson, B. Meyers, A. Jacobs, P. Johns, and S. Kane. Working Overtime: Patterns of Smartphone and PC Usage in the Day of an Information Worker. In *Pervasive*, 2009.

[34] B. Kellermann and R. Bohme. Privacy-enhanced event scheduling. In *CSE*, 2009.

[35] L. Kissner and D. Song. Privacy-preserving set operations. In *Crypto*, 2005.

[36] T. Leaute and B. Faltings. Privacy-preserving multi-agent constraint satisfaction. In *CSE*, 2009.

[37] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia. Privacy-aware proximity based services. In *Mobile Data Management*, 2009.

[38] Nokia Qt Development Frameworks. Qt – Cross Platform application and UI framework. http://qt.nokia.com/, 2011.

[39] Nokia Research Center. Nokia Instant Community. http://tinyurl.com/nokiainstcomm, 2010.

[40] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, 1999.

[41] K. Puttaswamy and B. Zhao. Preserving privacy in location-based mobile social applications. In *HotMobile*, 2010.

[42] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[43] J. Shi, R. Zhang, Y. Liu, and Y. Zhang. PriSense: Privacy-Preserving Data Aggregation in People-Centric Urban Sensing Systems. In *Infocom*, 2010.

[44] G. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *J. Amer. Inst. Elect. Eng.*, 45, 1926.

[45] A. C. Yao. Protocols for secure computations. In *FOCS*, 1982.

[46] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *Principles and Practice of Constraint Programming*, 2006.

[47] G. Zhong. Distributed approaches for Location Privacy. Masters Thesis, University of Waterloo, 2008.

[48] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three protocols for Location Privacy. In *PET*, 2007.

[49] A. Zunino and M. Campo. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 2009.

Appendix

A The Paillier Cryptosystem

The Paillier Cryptosystem [40] relies on the decisional composite residuosity assumption. Given a number $n = pq$ (where p and q are two large prime numbers), we define z as a n-th residue modulo n^2 if there exists a number $y \in \mathbb{Z}^*_{n^2}$ s.t. $z = y^n \mod n^2$. The problem of deciding if z is a n-th residue is believed to be computationally hard. The Paillier Cryptosystem involves the following algorithms:

- *Key generation:* Select $n = pq$ where p and q are two random large prime numbers. Pick a random generator $g \in \mathbb{Z}^*_{n^2}$ s.t. $\mu = (L(g^\lambda \bmod n^2))^{-1} \mod n$ exists, given that $\lambda = lcm(p-1, q-1)$ and $L(x) = \frac{(x-1)}{n}$. The public key is (n,g) and the private key is (λ,μ).
- *Encryption:* To encrypt a message $m \in \mathbb{Z}_n$, pick a random $r \in \mathbb{Z}^*_n$ and compute the ciphertext: $c = E_r(m) = g^m \cdot r^n \bmod n^2$. The resulting ciphertexts are then elements of $\mathbb{Z}_{n^2}$.
- *Decryption:* Given a ciphertext $c \in \mathbb{Z}_{n^2}$, the decryption is performed by: $m = D(c) = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$ where $L(x) = \frac{(x-1)}{n}$.

Indistinguishable encryption. The Paillier Cryptosystem is a probabilistic encryption scheme. Consequently, two encryptions ($c_1 = E_{r_1}(m)$ and $c_2 = E_{r_2}(m)$) of the same value m encrypted with different random values (r_1 and r_2) are computationally indistinguishable, i.e., one cannot tell whether or not they encrypt the same two plaintexts (without decrypting them).

Homomorphic properties. Paillier is also additively homomorphic. Hence, given $E_{r_1}(m_1) = (g)^{m_1} \cdot (r_1)^n \bmod n^2$ and $E_{r_2}(m_2) = (g)^{m_2} \cdot (r_2)^n \bmod n^2$, one can easily compute:

$$E_{r_1 r_2 \bmod n^2}(m_1 + m_2 \bmod n) = E_{r_1}(m_1) \cdot E_{r_2}(m_2) \bmod n^2 =$$
$$= (g)^{m_1+m_2} \cdot (r_1 r_2)^n \bmod n^2$$

It then follows that Paillier also allows the private multiplication. Given $E_r(m) = g^m \cdot r^n \bmod n^2$ and $z \in \mathbb{Z}_n$, we obtain:

$$E_{r^z \bmod n^2}(m \cdot z \bmod n) = E_r(m)^z \bmod n^2 = g^{mz} \cdot r^{nz} \bmod n^2$$

B Private Set Intersection from [31]

We now review the Private Set Intersection (PSI) construction, recently proposed by Jarecki and Liu in [31], on which our Private Interest Sharing protocol from Section 3 is based. This protocol is proven secure in the Random Oracle Model (ROM), under the One-More Gap Diffie-Hellman (OMGDH) assumption [5], in presence of malicious adversaries.

We present Jarecki and Liu's PSI in Figure 8. Note that the protocol executes on public input $p, q, H_1(\cdot), H_2(\cdot)$; specifically, p is a large prime, q is a large divisor of $p-1$, while $H : \{0,1\}^* \to \mathbb{Z}_p^*$, and $H' : \{0,1\}^* \to \{0,1\}^\gamma$ (given a security parameter γ) are two cryptographic hash functions. All computation is performed "$\bmod p$". Compared to Figure 2 in [31], we slightly modify some notation, to fit presentation of our Private Interest Sharing protocol from Section 3.

Observe that our description of the PSI protocol is simplified, as we only need semi-honest security, while the original protocol in [31] is geared for malicious adversaries. Specifically, to prevent the sender from maliciously deviating from the protocol, [31] let sender and receiver interact in a zero-knowledge proof $\pi = PoK\{\alpha | z_i = (y_i)^\alpha, i = 1, \ldots, v\}$. If π does not verify, the receiver aborts execution. Also, to facilitate input extraction during simulation (needed by the proofs in the malicious model), the protocol in [31] actually computes $t_{s:j} = H_2[H_1(s_j), H_1(s_j)^\alpha]$ and $t_{c:i} = H_2[H_1(c_i), (z_i)^{(1/r_i)}]$.

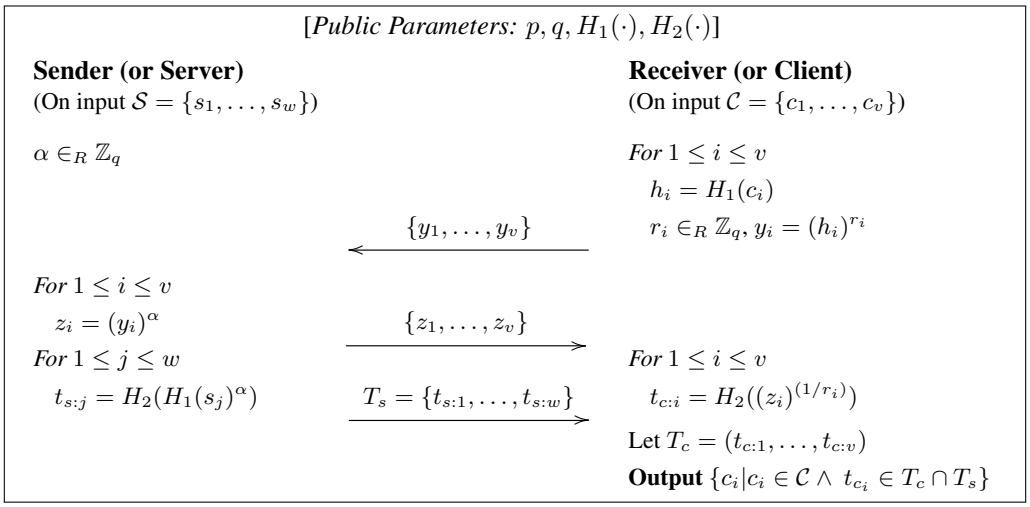


Figure 8: Private Set Intersection Protocol by Jarecki and Liu [31].