

# Reclaiming Privacy for Smartphone Applications

Emiliano De Cristofaro<sup>1</sup>, Anthony Durussel<sup>2</sup>, Imad Aad<sup>2</sup>

<sup>1</sup> University of California, Irvine – edecrist@uci.edu

<sup>2</sup> 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.

## I. 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 [20]. Smartphone applications are progressively enabling sophisticated interactions among people. Nowadays, users share information, for instance, for real-time social networking [27], to discover nearby friends (e.g., using Google Latitude or Foursquare), to exchange recommendations [17], or even for work-related purposes [21].

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. 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 [15] or Private Set Intersection (PSI) [11], [23], [8], [19], [7]. While some of these solutions are quite efficient for the two-party case,

the multi-party variants are not practical enough for large-scale deployment. For instance, the most efficient multi-party PSI, [23], incurs computational complexity *quadratic* in the size of participants' inputs. Alternatively, statistical methods for protecting privacy have also been used [18], [5], 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 private information is concentrated at this party. Beyond obvious privacy implications, this modus operandi imposes severe concerns about data loss, subpoena, or information sale.

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 [26] or Amazon EC2 [1]), without disclosing any data in the clear. 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.

### A. Roadmap

Motivated by the arguments above, we introduce an appropriate architecture that involves several smartphone users and a semi-trusted server (Sec. II). We then present the design and the implementation of several privacy-enhanced applications. We guarantee privacy by modeling efficient cryptographic tools, to 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 (Sec. III). 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 (Sec. IV). We are stimulated by ongoing projects aiming at automatically filling mobile phones' calendars with users' activities (e.g., based on emails, text messages,

routes). We propose three solutions that target different scenarios, corresponding to different efficiency, privacy, and system requirements.

Finally, we analyze the performance of proposed protocols (Sec. V), discuss related work (Sec. VI), and conclude with some future work items (Sec. VII). The Appendix reviews the Paillier Cryptosystem [28], used in two of our protocols.

## II. PRELIMINARIES

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

**Players.** Our system consists of  $N$  smartphone users,  $P_1, \dots, P_N$ , called *participants*, and a public *server*,  $S$ . Participants  $P_1, \dots, P_N$  hold private information, denoted, resp., with  $x_1, \dots, x_N$ , whereas,  $S$  does not generate any 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.

**Privacy-preserving Computation.** Participants,  $P_1, \dots, P_N$ , wish to jointly compute some function  $f(x_1, \dots, x_N)$ , without revealing any information about their private input, 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  $(x_1, \dots, x_N)$  or  $f(x_1, \dots, x_N)$ , as it receives and processes only encryptions of participants' inputs. Specifically, we define:

**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. Due to space limitation, we describe adversarial games in the extended version of the paper [6]. As  $S$  only receives encrypted inputs, privacy is 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  $f(x_1, \dots, x_N)$ . Privacy is considered as the probabilistic advantage that  $P_j$  gains from the protocol execution in learning  $P_i$ 's input. Again, adversarial games are discussed in [6].

**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 [13]. 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.

## III. 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 Sec. II—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).

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. Indeed, our efforts are currently focused on the deployment (on our Nokia devices) of actual privacy-enhanced applications relying on PIS.

**Informal Definition of PIS.** We assume  $N$  participants,  $P_1, \dots, P_N$ , 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  $\{b_{i:1}, \dots, b_{i:m}\}$  – 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  $\tau$  participants. Participants' privacy needs to be guaranteed by ensuring that, after protocol execution, the participants do not learn anything beyond matching interests. Also, server  $S$  learns no information about any participant's input. (Recall definitions in Sec. II).

**Protocol Specification.** Our PIS protocol 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  $q$ ,  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*$ , and  $H' : \{0, 1\}^* \rightarrow \{0, 1\}^\gamma$  (given a security parameter  $\gamma$ ) 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 mod  $p$ .)

[2. Interaction] Each participant  $P_i$  (for  $i \in [2, N]$ ), on input  $\{b_{i:1}, \dots, b_{i:m}\}$ , 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*  $\tau$  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 address 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: as a result, random exponents are only 160-bit and modular exponentiation are very efficient (with no loss of security). 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), whereas, for the other participants, the overhead amounts to  $m$  group elements and hash values.

**Privacy of PIS.** Our PIS construction provides provable privacy guarantees. Privacy w.r.t. the Server is guaranteed as  $S$  only receives outputs of the one-way function  $H, H'$ , whose inputs cannot be “forged” unless  $S$  knows  $\alpha$  (secret to  $P_1$  only) or some  $r_{i:j}$  (secret to  $P_i$ 's). Privacy w.r.t. Other Participants immediately stems from security arguments of the Private Set Intersection technique in [19], proven secure under the One-More-DH assumption [2], on which our PIS

protocol is based. Recall, however, that [19] 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 [23] require several rounds of computation and computational complexity at least quadratic in the size of participants' inputs. We discuss, in more details, arguments of PIS privacy guarantees in the extended version of the paper [6].

#### IV. PRIVATE SCHEDULING

In this section, we explore the concept of *Private Scheduling* for smartphones. Recall example (4) from Sec. III: 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/unavailable), 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 Sec. II to aggregate users' encrypted inputs, but prevent the server from learning any information. 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.

##### A. The Private Scheduling Problem

Private Scheduling involves  $N$  different participants,  $P_1, \dots, P_N$ .  $P_1$  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,  $\tau$ , 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  $\tau$  (if any).

### B. 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 [28], 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 the Appendix for a detailed description.

Following the intuition of [10], 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.<sup>1</sup>

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*):

$$x \xrightarrow{\text{vbr}} \vec{X} = [\underbrace{E(0), \dots, E(0)}_{x \text{ times}}, E(1), E(z), \dots, E(z)]$$

$E(\cdot)$  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  $E(\text{tag})$ : this allows to identify which value corresponds to the minimum, after the homomorphic minimization is performed.

Observe that vector  $\vec{X}$  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.

**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.

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  $\text{rand} = E(0)$ , 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  $\text{rand} \leftarrow (\text{rand})^{\exp}$ , where  $\exp$  is a

<sup>1</sup>E.g., the computation of homomorphic argmin may be useful for privacy-preserving data aggregation in sensor networks or urban sensing systems.

relatively small random exponent, and we continue with the next element. We describe the details of *PrivSched-v1* below.

[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  $\vec{v}_j$ :

$$\vec{v}_j = [\underbrace{E(0), \dots, E(0)}_{c_{1:j}}, E(1), \underbrace{E(z), \dots, E(z)}_{\tau - c_{1:j}}]$$

Finally,  $P_1$  sends  $\{\vec{v}_1, \dots, \vec{v}_m\}$ , 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  $\{\vec{v}_1, \dots, \vec{v}_m\}$  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  $\vec{v}_j$  (for  $j \in [1, m]$ ) by shifting the elements of each vector  $c_{i:j}$  positions right, and replacing them by  $E(0)$ :

$$\vec{v}_j' \xleftarrow{\text{def}} \vec{v}_j >> c_{i:j} = [\underbrace{E(0), \dots, E(0)}_{c_{i:j}}, v_{j,1}, \dots, v_{j,\tau - c_{i:j}}]$$

To mask her modifications,  $P_i$  re-randomizes the vectors  $\vec{v}_j'$ 's by multiplying the generic element  $v_{j,k}$  by a random  $E(0)$ . Finally, she sends the updated  $\{\vec{v}_1, \dots, \vec{v}_m\}$  back to  $S$ .

Remark that this phase is repeated, sequentially, for each participant,  $P_2, \dots, P_N$ : at the end  $S$  obtains the final  $\{\vec{v}_1, \dots, \vec{v}_m\}$  where, for  $j \in [1, m]$ :

$$\vec{v}_j' = [\underbrace{E(0), \dots, E(0)}_{ac_j}, \underbrace{E(1), E(z), \dots, E(z)}_{\tau - ac_j}]$$

[3. Minimization] Upon receiving the final  $\{\vec{v}_1, \dots, \vec{v}_m\}$ ,  $S$  computes the homomorphic argmin: first,  $S$  raises each element of  $\vec{v}_j'$  to  $2^j$ ; then, it computes a vector  $\vec{q}$ . (The sum of all tags should not exceed the size of the Paillier modulus).

$$\begin{aligned} \vec{v}_j' &= (\vec{v}_j')^{2^j} = [(v_{j,1})^{2^j}, (v_{j,2})^{2^j}, \dots, (v_{j,\tau})^{2^j}] \\ \vec{q} &\xleftarrow{\text{def}} [q_1, q_2, \dots, q_\tau] = [\prod_{j=1}^m v_{j,1}', \dots, \prod_{j=1}^m v_{j,\tau}'] = \\ &= [\underbrace{E(0), \dots, E(0)}_{\min}, q_{\min+1}, \dots, q_\tau] \end{aligned}$$

Next,  $S$  sends  $\vec{q}$  to the Initiator  $P_1$ , that decrypts each element of  $\vec{q}$  using  $SK_1$ . The minimum aggregated cost corresponds to the number of consecutive 0's in the first positions of  $\vec{q}$ . 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  $\phi$ .  $P_1$  retrieves the index of this timeslot by observing which bits are equal to 1 in the binary decomposition of  $\phi$ .  $P_1$  may additionally retrieve the 2<sup>nd</sup> minimum timeslot by subtracting  $(\phi \cdot z)$  from the non-null decrypted elements of  $\vec{q}$ . Iterating this method leads to retrieval of all timeslots with aggregated cost less than  $\tau$ . Observe that  $\tau$  is a system parameter and can be tuned to meet different requirements. Smaller values of  $\tau$  result into a

smaller  $\vec{q}$  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  $\tau$  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  $\vec{v}_1$ , 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  $\{\vec{v}_1, \dots, \vec{v}_m\}$ ), and at most  $\tau$  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-computation is 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  $\vec{q}$ . The communication overhead amounts to  $(m \cdot \tau)$  ciphertexts (i.e., 2048-bit each) for all participants. Additionally, the Initiator receives  $\tau$  ciphertexts (in  $\vec{q}$ ).

**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 Sec. II, by virtue of the CPA-security of the Paillier cryptosystem [28]. 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. For more details on collusion, as well as possible countermeasures, such as threshold secret sharing [30], we refer to the paper's extended version [6].

### C. 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 now 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 the 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. We discuss the modified protocol below.

[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 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:

$$\begin{aligned} E(0, a_1) &\longrightarrow [E(1), E(z), E(z), E(z), \dots, E(z)] \\ E(1, a_1 y_1) &\longrightarrow [\underbrace{E(0), E(1), E(z), E(z), \dots, E(z)}_1] \\ &\dots \longrightarrow \dots \\ E(\tau_{max}, a_1 y_1^{\tau_{max}}) &\longrightarrow [\underbrace{E(0), E(0), E(0), E(0), \dots, E(0)}_{\tau_{max}}] \end{aligned}$$

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  $ec_{1:j}$ :

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

Next,  $P_1$  picks  $N-1$  random values,  $(x_2, \dots, x_N)$ . 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  $E(0, y_1)$ :

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

Then,  $P_i$  sends  $S$   $\{ec_{i:1}, \dots, ec_{i:m}\}$  and  $E(x_i, r_i)$ .

Next,  $S$  computes the (encrypted) aggregated cost of each timeslot  $j \in [1, m]$ , using Paillier's homomorphism:

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

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:

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

[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<sup>st</sup> 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).

$$\begin{aligned} E(0, ar^j a_1) &\longrightarrow [E(1), E(z), E(z), \dots, E(z)] \\ E(1, ar^j a_1 \cdot y_1) &\longrightarrow [\underbrace{E(0)}_1, E(1), E(z), \dots, E(z)] \\ &\dots \longrightarrow \dots \\ E(\tau_{max}, ar^j a_1 \cdot y_1^{\tau_{max}}) &\longrightarrow [\underbrace{E(0), E(0), E(0), \dots, E(0)}_{\tau_{max}}] \end{aligned}$$

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

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

Next,  $S$  starts the homomorphic argmin using vectors  $\vec{v}_j$ , i.e., the vbr of each aggregated cost, leveraging the following tagging technique. The server raises each element of  $\vec{v}_j$  to 2 <sup>$j$</sup>  (for  $j \in [1, m]$ ):

$$\vec{v}_j' = (\vec{v}_j)^{2^j} = [(v_{j,1})^{2^j}, (v_{j,2})^{2^j}, \dots, (v_{j,\tau})^{2^j}]$$

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

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

Finally,  $P_1$  decrypts each element of  $\vec{q}$  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  $\vec{q}$ .  $q_{min+1}$  decrypts to the sum of tags corresponding to the timeslot(s) producing the minimum aggregated cost. Again, we denote this sum with  $\phi$ .  $P_1$  retrieves the index of this timeslot by observing which bits are equal to 1 in the binary decomposition of  $\phi$ .  $P_1$  may additionally retrieve the 2<sup>nd</sup> minimum timeslot by subtracting  $(\phi \cdot z)$  from the non-null decrypted elements of  $\vec{q}$ . Iterating this method leads to retrieval of all timeslots with aggregated cost smaller than  $\tau$ .

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  $\tau$  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  $\vec{q}$ . The communication overhead amounts to  $m$  ciphertexts (i.e., 2048-bit each) for all participants. Additionally, the Initiator receives  $\tau$  ciphertexts (in  $\vec{q}$ ).

**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*, neither participants nor the server can violate the privacy requirements described in Sec. II, by virtue of the CPA-security of the Paillier cryptosystem [28]. The Initiator gets the vector  $\vec{q}$  containing only suitable timeslots (i.e., whose aggregated cost is smaller than  $\tau$ ). Considerations about possible collusion are the same as in *PrivSched-v1*, thus, we do not repeat them here. Remark, that since equal aggregated costs may appear at the same line of the mapping, the server could detect repeated aggregated costs.

#### D. 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 Sec. V) provides encouraging performance results for *PrivSched-v1* and v2, we now present yet another construction, *S-PrivSched* (Symmetric Private Scheduling), which only involves symmetric-key operations, and reduces computational and bandwidth overheads. Our intuition is as follows: 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 [4]) to perform cost aggregation.

**Preamble.** *S-PrivSched* leverages the cryptosystem in [4], a tailored modification of the Vernam cipher [31] 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. (2) Let  $k$  be randomly generated keystream, where  $k \in [0, M - 1]$ . (3) Compute  $ct = Enc_k(msg) = (msg + k) \bmod M$ .
- **Decryption:**  $Dec_k(ct) = (ct - k) \bmod M$
- **Addition of ciphertexts:** Let  $ct_1 = Enc_{k_1}(msg_1)$ ,  $ct_2 = Enc_{k_2}(msg_2)$ . Aggregated ciphertext:  $ct = ct_1 + ct_2 \bmod M = Enc_k(msg_1 + msg_2)$  (where  $k = k_1 + k_2$ ).

Observe that  $M$  needs to be sufficiently large:  $0 \leq msg < M$ , or  $\sum_{i=1}^N msg_i < M$  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 [14], or the output of a length-preserving hash function (as shown in [4]).

#### Protocol Specification.

We now present S-PrivSched.

[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  $P_2, \dots, P_N$  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 or collude with the (scheduling) service provider. One can 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  $\{ec_{i:1}, \dots, ec_{i:m}\}$  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  $\{eac_1, \dots, eac_m\}$  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.

**Privacy of S-PrivSched.** Privacy of S-PrivSched stems from the security of the cryptosystem in [4]. 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 protocols, 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 similar to PrivSched-v1, thus, we do not repeat them here.

## V. 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 Sec. III). 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 and the open-source cryptographic libraries libpaillier, openssl, and libgmp, 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 1, 2, and 3. 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 small 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 trades-off some of the privacy properties, as it reveals aggregated costs of *all* timeslots to the Initiator, and requires the distribution of a shared secret (e.g., via SMS).
- (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

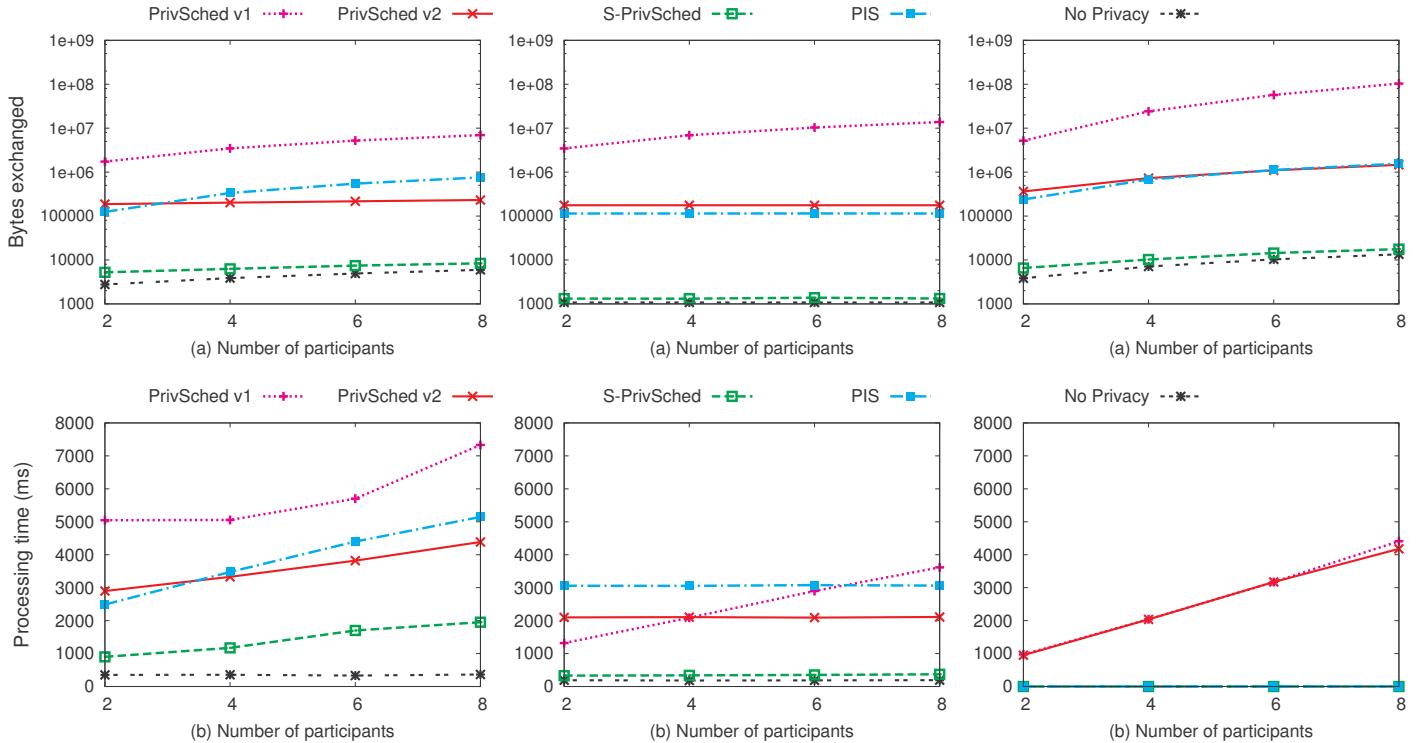


Fig. 1: Initiator

Fig. 2: Each participant

Fig. 3: Server

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 v2, require only a few seconds in most realistic settings.

## VI. 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) [15] 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. *Multi-party Private Set Intersection* [23] 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 [11], [23], [8], [19], [7]. Whereas, techniques for multiple players require a number of long exponentiations quadratic in the

size of the sets [23] – well beyond the requirements of our smartphone setting. 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 [23].

**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). For instance, [3] introduces a semi-trusted third party to obliviously compare two numbers, and [9] 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.

**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 [24], [22]. These solutions incur high computation and communication overhead and are unpractical for mobile environments.

**Private Interest Sharing.** Besides primitives discussed above, several techniques have focused on problems similar to Private Interest Sharing. Due to space limitation, we only overview the most closely related ones. The work in [32] 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 [28] 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)  $\tau$  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). Also, the recent position paper in [29] 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 [12] and proximity-based services [25]. Finally, note that some of the applications envisioned in the context of PIS or Private Scheduling (e.g., polls) also resemble *e-voting* problems [16]. In reality, however, e-voting involves several different entities with specific roles and has very different requirements.

## VII. 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., 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.** We would like to thank PerCom 2011's reviewers and Valtteri Niemi for the very precious comments.

## REFERENCES

- [1] Amazon Elastic Compute Cloud. <http://aws.amazon.com/ec2>, 2010.
- [2] 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.
- [3] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *CCS*, 1999.
- [4] 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.
- [5] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. AnonySense: Privacy-aware people-centric sensing. In *Mobisys*, 2008.
- [6] E. De Cristofaro, A. Durussel, and I. Aad. Reclaiming Privacy for Smartphone Applications (Full Version). <http://eprint.iacr.org/2011/041>.
- [7] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *Asiacrypt*, 2010.
- [8] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, 2010.
- [9] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW*, 2002.

- [10] L. Ertaul and Vaidehi. Computing Aggregation Function Minimum/Maximum using Homomorphic Encryption Schemes in Wireless Sensor Networks. In *ICWN*, 2007.
- [11] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt*, 2004.
- [12] 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.
- [13] O. Goldreich. *Foundations of cryptography*. Cambridge Univ Pr, 2004.
- [14] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *JACM*, 33(4):792–807, 1986.
- [15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.
- [16] D. Gritzalis. *Secure electronic voting*. Springer, 2003.
- [17] J. Hong, E. Suh, and S. Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4), 2009.
- [18] B. Hore, J. Wickramasuriya, S. Mehrotra, N. Venkatasubramanian, and D. Massagué. Privacy-preserving event detection in pervasive spaces. In *PERCOM*, 2009.
- [19] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN*, 2010.
- [20] A. Karlson, S. Iqbal, B. Meyers, G. Ramos, K. Lee, and J. Tang. Mobile task flow in context. In *CHI*, 2010.
- [21] 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.
- [22] B. Kellermann and R. Bohme. Privacy-enhanced event scheduling. In *CSE*, 2009.
- [23] L. Kissner and D. Song. Privacy-preserving set operations. In *Crypto*, 2005.
- [24] T. Leaute and B. Faltings. Privacy-preserving multi-agent constraint satisfaction. In *CSE*, 2009.
- [25] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia. Privacy-aware proximity based services. In *Mobile Data Management*, 2009.
- [26] Nokia. Ovi. <http://ovi.com/>, 2010.
- [27] Nokia Research Center. Nokia Instant Community. <http://tinyurl.com/nokiainstcomm>, 2010.
- [28] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, 1999.
- [29] K. Puttaswamy and B. Zhao. Preserving privacy in location-based mobile social applications. In *HotMobile*, 2010.
- [30] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [31] G. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *J. Amer. Inst. Elect. Eng.*, 45, 1926.
- [32] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three protocols for location privacy. In *PET*, 2007.

## APPENDIX

The Paillier Cryptosystem [28] 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.  $z = y^n \pmod{n^2}$ . 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.  $\mu = (L(g^\lambda \pmod{n^2}))^{-1} \pmod{n}$  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  $(\lambda, \mu)$ .

*Encryption:* To encrypt a message  $m \in \mathbb{Z}_n$ , pick a random  $r \in \mathbb{Z}_n^*$  and compute the ciphertext:  $c = E_r(m) = g^m \cdot r^n \pmod{n^2}$ . 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:  $m = D(c) = L(c^\lambda \pmod{n^2}) \cdot \mu \pmod{n}$  where  $L(x) = \frac{(x-1)}{n}$ .

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).

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

$$\begin{aligned} E_{r_1 r_2 \pmod{n^2}}(m_1 + m_2 \pmod{n}) &= E_{r_1}(m_1) \cdot E_{r_2}(m_2) \pmod{n^2} \\ &= g^{m_1+m_2} \cdot (r_1 r_2)^n \pmod{n^2} \end{aligned}$$

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

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