SPA: Inviting Your Friends to Help Set Android Apps

Zeqing Guo¹, Weili Han¹,², Liangxing Liu¹, Wenyuan Xu¹, Ruiqi Bu¹, Minyue Ni¹
¹. Software School, Fudan University, Shanghai, China
². Department of Electronic Engineering, Zhejiang University, Hangzhou, China
³. Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
{wlhan@fudan.edu.cn, xuwenyuan@zju.edu.cn}

ABSTRACT
More and more powerful personal smart devices take users, especially the elder, into a disaster of policy administration where users are forced to set personal management policies in these devices. Considering a real case of this issue in the Android security, it is hard for users, even some programmers, to generally identify malicious permission requests when they install a third-party application. Motivated by the popularity of mutual assistance among friends (including family members) in the real world, we propose a novel framework for policy administration, referring to Socialized Policy Administration (SPA for short), to help users manage the policies in widely deployed personal devices. SPA leverages a basic idea that a user may invite his or her friends to help set the applications. Especially, when the size of invited friends increases, the setting result can be more resilient to a few malicious or unprofessional friends. We define the security properties of SPA, and propose an enforcement framework where users’ friends can help users set applications without the leakage of friends’ preferences with the supports of a privacy preserving mechanism. In our prototype, we only leverage partially homomorphic encryption cryptosystems to implement our framework, because the fully homomorphic encryption is not acceptable to be deployed in a practical service at the moment. Based on our prototype and performance evaluation, SPA is promising to support major types of policies in current popular applications with acceptable performance.

Categories and Subject Descriptors
D.4.6 [Software]: Security and Protection—Access controls

General Terms
Security

Keywords
Policy based Management, Policy Administration, Socialized Policy Administration, Android, Social Computing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SACMAT’15, June 1–3, 2015, Vienna, Austria.
Copyright © 2015 ACM 978-1-4503-3556-0/15/06 ...$15.00.
http://dx.doi.org/10.1145/2752952.2752974.

1. INTRODUCTION
Recently, users are faced with smarter and smarter personal devices, e.g., smartphones, which contain large amounts of information and powerful sensors. According to the report of Technasia [26], 1.3 billion smartphones, containing over 0.7 million third-party applications, have been shipped to customers since 2014. That is, over one out of six persons in the world got a new smart phone in the past year. Unfortunately, these users, even applications’ developers, are usually unprofessional to manage these applications in devices [6][2][8][17][7].

Friends (including family members) are indeed supports for security in both the real world and the cyberspace. In real life, people who are not good at security management may invite their friends (including family members) who are professional to help set their devices and applications. For example, a woman could invite her husband to set electronic devices; an elder could ask grandson to provide a configured smart phone. The invitation would be possible among friends if they agree to share their privacy. This is very common in current social network services. In this paper, we extend this idea into the cyberspace, which means, a user may ask for help from his or her friends to set their devices via Social Network Services (SNSs for short) on the Internet. Although this extension looks very intuitive, some technical challenges, such as privacy preservation, should be resolved. When a user asks his or her friends some sensitive settings and the friends cannot control the responses, or the framework cannot protect the responses from personal information leakage, the framework can be maliciously used to gather privacy from the friends.

Existing methods of collaborative policy authoring [30] or administration [12] are far from the protection of friends’ privacy. The work of [30] aims to protect the privacy with the supports of nominated friends (co-owners of content) when a user shares his or her personal content on a SNS, such as Facebook. It requires friends’ interactions, and does not consider friends’ privacy. In addition, CPA (Collaborative Policy Administration) proposed in the work of [12] leverages other similar settings of applications to set users’ application. CPA is not required to consider the privacy protection issue of the applications, because the settings of these applications are in public. However, in our extension, the privacy of these involved friends can be very critical, because the responses would implicitly include some sensitive information. Once the information is automatically gathered, more sensitive personal profiles would be exposed to the public.
This paper, therefore, articulates the problem of socialized policy administration and proposes an enforcement framework. The main contributions are as follows:

- We articulate the methodology of socialized policy administration where a user can request his or her friends to help automatically set sensitive policies. We design a model family, including Basic SPA which includes essential elements of SPA, m-SPA which supports multiple friends groups of a user, w-SPA which supports the user labels weights for friends, and Composite SPA which merges the features of m-SPA and w-SPA. We also design a framework for a mobile scenario where the privacy of friends is preserved. To the best of our knowledge, this is the first paper to enable unprofessional users with the help of their friends to set sensitive policies with privacy preservation.

- We find out that partially homomorphic encryption algorithms which we leverage in SPA can support major policy types in popular mobile applications. In SPA, homomorphic encryption will run in a semi-trusted service, where the SPA responses from friends will be merged together. A thin client, such as a smart phone, can decrypt the merged result, then set the policy in the client. Due to the performance of current fully homomorphic encryption, although the under-developed fully homomorphic encryption may support more types of policies, several practical algorithms which support one or several types of homomorphic encryption are leveraged in our implementation. We analyze the possible supported types of policies in SPA by these homomorphic encryption algorithms based on our downloaded popular third-party applications.

Note that, although the policies of devices can be automatically set by SPA, users can modify the policies in their devices by themselves. As a result, professional users may also obtain useful references from their friends.

The rest of this paper is organized as follows: Section 2 introduces the background knowledge and our target problem. Section 3 describes the security assumptions and defines SPA models. Section 4 describes the design and implementation of SPA. We then present our experimental process and evaluation results in Section 5. Next, we have a discussion about vulnerabilities of SPA and security of homomorphic encryption in Section 6. Section 7 introduces related work. Finally, Section 8 summarizes this paper and outlines our future work.

2. BACKGROUND AND MOTIVATION

2.1 Homomorphic Encryption

Homomorphic encryption is a form of encryption which allows specific types of computations to be carried out on ciphertext and generate an encrypted result which, when decrypted, matches the result of operations performed on the plaintext [28]. That is, A may encrypt his message \( m \) and send the ciphertext \( E(m) \) to B. B may then take the ciphertext \( E(m) \) and evaluate a function \( F \) on the underlying \( m \) obtaining the encrypted result \( E(F(m)) \). B may decrypt this result, and achieve the wanted functionality on data \( m \), but B learns nothing about the data that it has operated on.

Although the fully homomorphic encryption (FHE) which supports an arbitrary function \( F \) on ciphertexts was proposed several years ago [3][25][27], its performance is hard to meet the requirements for a practical business service. Gentry showed the first fully homomorphic encryption scheme using lattice-based cryptography in 2009 [10][11]. Such a scheme allows one to compute arbitrary functions over encrypted data without the decryption key, i.e., given encryptions \( E(m_1), \ldots, E(m_n) \), one can efficiently compute a composite ciphertext that encrypts \( F(m_1, \ldots, m_n) \) for any efficiently computable function \( F \).

As a result, partially homomorphic cryptosystems are the good choices in practical. Some partially homomorphic cryptosystems are as follows.

- **Paillier (Additive):** The Paillier cryptosystem, invented by Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography [18]. The cryptomorphic algorithm generates a key pair, consisting of a public key and a private key. The public key is used to encrypt plaintext; whereas the private key is used to decrypt ciphertext.

The scheme is an additive homomorphic cryptosystem [4]. The additive homomorphic properties can be represented as follows:

\[
F(E(m_1), E(m_2)) = E(m_1 + m_2)
\]

Here, \( E \) refers to encryption function, and \( F \) is a function defined by the partially homomorphic cryptosystem whose value is the encryption of the sum of \( m_1 \) and \( m_2 \). And \( m_1, m_2 \) are two plaintexts.

- **RSA (Multiplicative):** RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977 [19]. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret.

The scheme is a multiplicative homomorphic cryptosystem [9]. The multiplicative homomorphic properties can be represented as follows:

\[
F(E(m_1), E(m_2)) = E(m_1 \cdot m_2)
\]

Here, the definitions of \( E, F, m_1, m_2 \) are the same as those in **Paillier**.

- **ElGamal (Multiplicative):** The ElGamal encryption system is an asymmetric key encryption algorithm for public-key cryptography. It was described by Elgamal in 1985 [5]. The scheme is also a multiplicative homomorphic cryptosystem, and its multiplication homomorphic properties is the same with RSA’s.

2.2 Motivated Scenario in Mobile Application Setting

A user, e.g., Alice, may set her applications with little professional knowledge, assisted by her friends whose settings can be kept secret without privacy leakage:

> Alice, Bob, Cindy, Dale, Eric are in a Facebook group named “Classmates”. Alice downloads Instagram, which is recommended by her classmates.
However, after she finishes the initially installing process, she is confused about how to configure these settings, because she never used Instagram. At the moment, Alice has no choice but to finish these settings independently, which is obviously very hard and time-consuming to understand all guides to set Instagram's policies.

What Alice expects is the following scene. She sends a request to her classmates in the Facebook group respectively. Her classmates receive the request, and tell her their configurations CONFIDENTIALLY. Then Alice can set her Instagram according to her classmates' configurations. E.g., if the majority of her classmates set “Like Notification” as “off”, then she sets it “off”. In addition, if Alice applies a high-level policy, e.g., follow major settings in “Classmates”, the settings in Instagram can be automatically finished without click by click.

Alice can modify the setting by herself if she has her different preferences after the automatically setting by SPA. But she can immediately set Instagram without the deep professional knowledge of application settings.

We define this process as “socialized policy administration” where the social relations are leveraged to help users set sensitive policies. This requirement would become more and more important when the mobile technologies, especially when smartphones and wearable smart devices are widely equipped by those who lack security awarenesses and experiences of policy administration.

3. SPA MODELS

3.1 Security Assumptions

In the above scenario, SPA is designed to protect Alice from malicious or harmful policies in her devices. That is, her unprofessional settings in security management seriously affect on the system security. We can conclude the following security assumptions:

- The major friends (Bob, Cindy, Dale, Eric) set correct security and management policies of applications. And their settings should be protected during SPA processes. That is, the user and the cloud service should know the least information of the user’s friends’ settings.

- The cloud service which helps SPA to merge the responses from friends (Bob, Cindy, Dale, Eric) is semi-trust. That is, details of the friends’ responses cannot be viewed through the cloud service. But when the cloud service colludes with the user, they can get privacy data of friends. The assumption is reasonable in the major secure cloud-based applications[13].

The privacy concern in SPA will lead that if Alice has only one friend, SPA will be ceased.

3.2 SPA Models

As is shown in Figure 1, we define a family of SPA models where (1) Basic SPA includes the essential elements of SPA; (2) m-SPA supports multiple friend groups when a user launches SPA requests; (3) w-SPA allows the user to set the weight according to each friend’s ability of managing policy; (4) Composite SPA supports multiple features of m-SPA and w-SPA. The above four models are proposed to meet different requirements of application scenarios.

3.2.1 Basic SPA Model

As is shown in Figure 2, the formal definition of Basic SPA is as follows.

**Definition 1. Basic SPA Model**

$$\text{Basic SPA} := (E, g, R, P, SPAPolicy)$$

Here, $E$, $g$, $R$, $P$, $SPAPolicy$ refer to a set of entities, a group, a set of roles, a set of processes, a high level policy respectively:

- **Entity (E)** refers to the user who has their customized setting for applications (such as Android apps). Let $e := (id, policies)$ denote an entity, where $id$ is the unique identification of $e$, policies consist of every application’s setting policy, each of which includes attributes as well as values. E.g., $e_{Alice} = (id_{Alice}, policies)$ represents that $e_{Alice}$ is an entity assigned to Alice with identification $id_{Alice}$, and her setting policies are represented by policies. In the motivated scenario, there are five entities: Alice, Bob, Cindy, Dale, Eric. We hide the applications’ policies of these entities here.

- **Group (g)** is created by a user (e.g., Alice in the motivated scenario) to implement the Socialized Policy Administration. Let $g = \{e_1, e_2, \ldots, e_n\}$ denote a
group, where each element $e_i$ is an entity. In Basic SPA model, a user, e.g., Alice, can send requests to a sole selected $g_i \subseteq \{g | e_{Alice} \in g\}$. In the motivated scenario, $g = \{Alice, Bob, Cindy, Dale, Eric\}$. Note that, $|g| \geq 3$.

- **Role (R)** refers to the entity’s role in Basic SPA Model. There are two types of roles: requester, and respondent. The requester refers to the one who sends SPA request and asks his or her friends for help about policy setting, and the respondent refers to the one who receives SPA request. E.g., if Alice asks her friends for help, Alice is a requester and her friends are respondents.

- **Process (P)** denotes either an SPA request or an SPA response in Basic SPA model. There are two types of processes: request and response. Both request and response can be described as a tuple $(e, r, policy)$, where $e$ refers to an entity, $r$ refers to a role, and policy, which is the setting policy that the user asks for, is a part of $e$.policies. E.g., in the motivated scenario, Alice is confused about how to configure settings on Instagram. Thus, she sends request to entities in her classmate group, including Bob, Cindy, Dale, Eric, for help. This request can be represented by (Alice, requester, Instagram). And these responses responded by her friends in the group can be represented by (Bob, respondent, Instagram), (Cindy, respondent, Instagram), etc.

- **SPAPolicy** refers to the high level policy supported by Basic SPA model. For instance, Basic SPA supports Average value policy where the requester can obtain the average number based on all respondents’ settings. For a sound volume setting, each respondent can report their value to a requester with confidence. In addition, requester knows only “a little”, “some”, “most” of the respondents instead of the exact number of respondents who respond his request, which can avoid the requester knowing the choices of all respondents when all respondents happen to make the same responses to a request.

### 3.2.2 Multi-group – m-SPA Model

As is shown in Figure 3, m-SPA is extended from Basic SPA. It allows requesters to send requests to multiple groups. The formal definition of m-SPA is as follows.

**Definition 2. m-SPA Model:**

$$m-SPA := (E, MG, R, P, SPAPolicy)$$

Here, $E$, $MG$, $R$, $P$, SPAPolicy refers to a set of entities, a multi-group, a set of roles, a set of processes, a high level policy respectively.

- **Multi-group (MG)** is a set of several groups of friends. Let $MG = \{g_1, \ldots, g_n\}$, where $n \geq 1$, denote a multi-group. If and only if an entity $e \in g_1 \cup \ldots \cup g_n$, where $g_1, \ldots, g_n \in MG$, roles($e$) can be a respondent of MG. If and only if an entity $e \in g_1 \cap \ldots \cap g_n$, where $g_1, \ldots, g_n \in MG$, roles($e$) can be a requester of MG. E.g., Alice owns two groups that are named highschool and university in m-SPA Model. The set $\{ \text{highschool, university} \}$ is a multi-group. She is allowed to be a requester in this multi-group and other entities in this multi-group are respondents.

Comparing with Basic SPA, the definitions of entity, role, process, and SPAPolicy do not change in m-SPA, whereas multi-group is a set of several groups selected by the requester.

### 3.2.3 Weighted entity – w-SPA Model

The structure of w-SPA is extended from Basic SPA shown in Figure 2, while the structure of entity is different. Considering the facts that respondents’ knowledge about managing privacy determines the reliability of the result and a requester is familiar with respondents, the result is more reliable if the requester gives each respondent a weight corresponding to each friend’s ability of managing policy. The following definition reflects this idea.

**Definition 3. w-SPA Model:**

$$w-SPA := (E_w, g, R, P, SPAPolicy)$$

w-SPA Model is unchanged from Basic SPA model except the structure of entity $E_w$.

- **Weighted Entity ($E_w$)** defines an entity, which includes an additional attribute weight (the value of weight is 100 by default), and a requester can change it before sending requests. Let $e_w := (id, policies, weight)$ denote an entity in w-SPA, where id and policies are the same as those in Basic SPA, and weight is a number determined by a requester.
3.2.4 Composite SPA

Composite SPA contains all features of both multi-group and weighted entity, as it combines \(m\)-SPA and \(w\)-SPA. Composite SPA remains unchanged from \(m\)-SPA except the structure of entity \(E\), which is the same as the entity in \(w\)-SPA.

3.3 Correctness and Robustness of SPA Models

We begin with Composite SPA, which is the most general case of these four SPA models, and formalize the correctness and robustness of Composite SPA model. Here, we only analyze the high level policy of Average value. Some hypotheses during the analysis are as follows:

- A user is a requester in \(g_1, \cdots, g_m\), and \([g_1 \cup \cdots \cup g_m] \cap \{\text{requester}\} = n\). That is, there are \(n\) different friends in the requester’s groups. These friends of the requester may be professional users, unprofessional users, or malicious users. And let \(n_{\text{pro}}, n_{\text{unpro}}, n_{\text{mal}}\) denote the number of professional, unprofessional, and malicious users respectively, so \(n_{\text{pro}} + n_{\text{unpro}} + n_{\text{mal}} = n\).

- The requester is confused about a setting policy, which is a continuous type in the range \([0, 100]\), for a certain application. SPA Policy for the continuous type is Average value in Composite SPA. Suppose that there exist four non-negative integers \(e_1, e_2, e_3, e_4\) where \(e_1, e_2, e_3, e_4 \in [0, 100]\) and \(e_1 < e_2 < e_3 < e_4\), and the setting values in \([e_1, e_2]\) may be harmful for the requester, while \([e_3, e_4]\) is the appropriate choice interval for him or her.

- Each friend may respond to the request with a setting value, and can be attached with a weight value. To simplify the proof, we assume that each friend will respond with a setting value, and setting values of professional, unprofessional, malicious users are in \([e_3, e_4]\), \([e_2, e_3]\), \([e_1, e_2]\) respectively. The setting value function and weight function can be defined as follows:

\[
\text{setting}(e) = \begin{cases} x, & \text{if } e \text{ is professional} \\ y, & \text{if } e \text{ is unprofessional} \\ z, & \text{if } e \text{ is malicious} \end{cases} \tag{1}
\]

\[
\text{weight}(e) = \begin{cases} w_{\text{pro}}, & \text{if } e \text{ is professional} \\ w_{\text{unpro}}, & \text{if } e \text{ is unprofessional} \\ w_{\text{mal}}, & \text{if } e \text{ is malicious} \end{cases} \tag{2}
\]

Here, \(x \in [e_3, e_4]\), \(y \in (e_2, e_3]\), \(z \in [e_1, e_2]\) refer to expectations of setting values of professional users, unprofessional users, malicious users respectively. The correctness of Composite SPA can be described by the proximity of the expectation of setting values to \(z\):

\[
\text{correctness}(n_{\text{pro}}, n_{\text{unpro}}, n_{\text{mal}}) = \frac{\sum_{e \in [e_3 \cup \cdots \cup e_4]} \text{weight}(e) \cdot \text{setting}(e)}{\sum_{e \in [e_1 \cup \cdots \cup e_n]} \text{weight}(e) \cdot x} \tag{3}
\]

Replacing each \(\text{setting}(e)\), \(\text{weight}(e)\) with equation (1) and (2), we get

\[
\text{correctness}(n_{\text{pro}}, n_{\text{unpro}}, n_{\text{mal}}) = 1 - \frac{(x - y) \cdot w_{\text{unpro}} \cdot n_{\text{unpro}} + (x - z) \cdot w_{\text{mal}} \cdot n_{\text{mal}}}{x \cdot (w_{\text{pro}} \cdot n_{\text{pro}} + w_{\text{unpro}} \cdot n_{\text{unpro}} + w_{\text{mal}} \cdot n_{\text{mal}})} \tag{4}
\]

Then, we calculate the partial differential of correctness function:

\[
\frac{\partial \text{correctness}}{\partial n_{\text{pro}}} (n_{\text{pro}}, n_{\text{unpro}}, n_{\text{mal}}) = \frac{(x - y) \cdot w_{\text{unpro}} \cdot n_{\text{unpro}} + (x - z) \cdot w_{\text{mal}} \cdot n_{\text{mal}}}{x(w_{\text{pro}} \cdot n_{\text{pro}} + w_{\text{unpro}} \cdot n_{\text{unpro}} + w_{\text{mal}} \cdot n_{\text{mal}})^2} \tag{5}
\]

Considering \(x > y\) and \(x > z\), the inequality \(\frac{\partial \text{correctness}}{\partial n_{\text{pro}}} > 0\) holds, which means when the proportion of the number of professional users increase, the correctness of Composite SPA will increase.

The robustness of Composite SPA can be described by the expectation on the expectation of setting value. The relation of \(n_{\text{unpro}}\) and the expectation of setting value can be expressed similarly by the following equation.

\[
\text{robustness}(n_{\text{pro}}, n_{\text{unpro}}, n_{\text{mal}}) = \frac{(x - z)w_{\text{pro}} \cdot n_{\text{pro}} + (y - z)w_{\text{unpro}} \cdot n_{\text{unpro}} \cdot w_{\text{mal}} \cdot n_{\text{mal}}}{(w_{\text{pro}} \cdot n_{\text{pro}} + w_{\text{unpro}} \cdot n_{\text{unpro}} + w_{\text{mal}} \cdot n_{\text{mal}})^2} \tag{6}
\]

Let \(a, b, c\) denote the percentage of professional, unprofessional, malicious users, where \(a, b, c\) are non-negative constants and \(a + b + c = 1\). Then we have \(a \cdot n = n_{\text{pro}}, b \cdot n = n_{\text{unpro}}, c \cdot n = n_{\text{mal}}\). The robustness can be converted to a function with independent variable \(n\):

\[
\text{robustness}(n) = \frac{(x - z)w_{\text{pro}} \cdot a + (y - z)w_{\text{unpro}} \cdot b \cdot w_{\text{mal}} \cdot n}{(w_{\text{pro}} \cdot a + w_{\text{unpro}} \cdot b + w_{\text{mal}} \cdot c)^2 \cdot n} \tag{7}
\]

It is easy to see that \(\frac{\partial \text{robustness}}{\partial n} > 0\) and \(\text{robustness} < 0\) for any positive integers \(n\). Therefore, when the \(n\) increases, the robustness increases. That is, when the \(n\) increases, the decreasing trend of expectation of setting value will decrease.

Composite SPA is the most general SPA model among these four SPA models defined in section 3.2, and \(m\)-SPA, \(w\)-SPA, Basic SPA are the special cases of Composite SPA. When the \(m\) of \(g_1, \cdots, g_m\) equals to 1 in Composite SPA, Composite SPA and \(w\)-SPA will be equivalent. In the case where \(w_{\text{pro}} = w_{\text{unpro}} = w_{\text{mal}}\) in Composite SPA, Composite SPA are equivalent to \(m\)-SPA. When \(m = 1\) and \(w_{\text{pro}} = w_{\text{unpro}} = w_{\text{mal}}\) in Composite SPA, Composite SPA and Basic SPA are equivalent. So the correctness and robustness of these three SPA models can also be represented by above equations.
### Table 1: Mappings from Policies and Merging Policies to Homomorphic Properties

<table>
<thead>
<tr>
<th>Privacy Policies</th>
<th>Merging Policies</th>
<th>Homomorphic Properties</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Majority/Minority preferred</td>
<td>Additive</td>
<td>Paillier</td>
</tr>
<tr>
<td>Single Select</td>
<td>Majority/Minority preferred</td>
<td>Additive</td>
<td>Paillier</td>
</tr>
<tr>
<td>Multiple Select</td>
<td>Majority/Minority preferred</td>
<td>Multiplicative</td>
<td>RSA</td>
</tr>
<tr>
<td>Continuous</td>
<td>Average value</td>
<td>Additive</td>
<td>Paillier</td>
</tr>
<tr>
<td></td>
<td>Maximum/Minimum value</td>
<td>Fully Homomorphic</td>
<td>Not practical</td>
</tr>
</tbody>
</table>

In our SPA framework, we define four types of policies as follows.

- **Switch.** The status of a setting consists of “on” and “off”, e.g., “find me through email address”.

- **Single Select.** There is more than one choice for a setting, but only one choice can be selected. E.g., there are 3 choices ( “Off”, “From People I Follow”, “From Everyone”) for the setting “Comment Notifications” on Instagram.

- **Multiple Select.** There is more than one choice for a setting, and more than one choice can be selected at the same time.

- **Continuous.** The numeric value of a setting is continuous.

We also define three types of high-level policies to merge responses as follows.

- **Majority/Minority preferred.** This type of merging policies applies to Switch, Single Select and Multiple Select types of policies. When a Majority preferred merging policy is set, the final merging result is the majority of choices made by friends.

- **Average value.** This type applies to the Continuous type of policies. The final merging result is the average value of choices made by friends.

- **Maximum/Minimum value.** This type also applies to the Continuous type of policies. When a Maximum value merging policy is set, the final merging result is the maximum value of choices made by friends.

Table 1 shows the mappings from a combination of policies and merging policies to required homomorphic properties and example algorithms.

### 4. DESIGN OF BASIC SPA FRAMEWORK

#### 4.1 Basic SPA Framework

The key flow of the proposed framework is illustrated in Figure 4. Before the processes, a key management server will disseminate the public and private keys of homomorphic encryption algorithms, such as Paillier to the entities in the group. To simplify the implementation, we only propose the framework to support Basic SPA. The framework is easily to extend to other models proposed in Section 3. The security concerns of these models are different from to be extend to other models proposed in Section 3. The framework is easily to extend to other models proposed in Section 3. The security concerns of these models are different from Basic SPA and we analyze their security in Section 6.1.

A socialized policy administration process includes the following steps:

1. A requester sends a request to all friends in the friend group.
2. A respondent receives the request, and sends a response encrypted with the public key of homomorphic encryption to a semi-trusted cloud service.
3. The cloud merges corresponding responses. Then the cloud sends an encrypted result to the requester. If the number of corresponding responses is one, the cloud will return an error to the requester.
4. The requester receives data that cloud sends and decrypts it with the private key of homomorphic encryption to get the final result.

We also define three types of high-level policies to merge responses as follows.

- **Majority/Minority preferred.** This type of merging policies applies to Switch, Single Select and Multiple Select types of policies. When a Majority preferred merging policy is set, the final merging result is the majority of choices made by friends.

- **Average value.** This type applies to the Continuous type of policies. The final merging result is the average value of choices made by friends.

- **Maximum/Minimum value.** This type also applies to the Continuous type of policies. When a Maximum value merging policy is set, the final merging result is the maximum value of choices made by friends.

Figure 4: SPA Framework

<table>
<thead>
<tr>
<th>Privacy Policies</th>
<th>Merging Policies</th>
<th>Homomorphic Properties</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Majority/Minority preferred</td>
<td>Additive</td>
<td>Paillier</td>
</tr>
<tr>
<td>Single Select</td>
<td>Majority/Minority preferred</td>
<td>Additive</td>
<td>Paillier</td>
</tr>
<tr>
<td>Multiple Select</td>
<td>Majority/Minority preferred</td>
<td>Multiplicative</td>
<td>RSA</td>
</tr>
<tr>
<td>Continuous</td>
<td>Average value</td>
<td>Additive</td>
<td>Paillier</td>
</tr>
<tr>
<td></td>
<td>Maximum/Minimum value</td>
<td>Fully Homomorphic</td>
<td>Not practical</td>
</tr>
</tbody>
</table>
4.2.2 Response

The structure of a response is shown in Table 3. The id field works as an identifier of a response. The respondent field describes the entity which makes the response, and the request field is used so that cloud knows which responses are supposed to be merged together to get a final result.

Table 3: Structure of a Response

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the unique identifier of a response</td>
</tr>
<tr>
<td>respondent</td>
<td>the entity that responds the request</td>
</tr>
<tr>
<td>request</td>
<td>the corresponding request</td>
</tr>
<tr>
<td>policy</td>
<td>app’s policy that the requester confuses with</td>
</tr>
</tbody>
</table>

4.2.3 Result

The structure of an SPA result is shown in Table 4. An SPA result is the result of merging responses. The cloud sends an SPA result to the corresponding SPA requester. The policy field includes the name of application that need to be set as well as the setting that describes how to set the application concretely.

Table 4: Structure of an SPA Result

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique identifier of a result</td>
</tr>
<tr>
<td>requester</td>
<td>the entity that starts the request</td>
</tr>
<tr>
<td>policy</td>
<td>app’s policy that the requester confuses with</td>
</tr>
</tbody>
</table>

4.3 Key Algorithms

In this section, we introduce two key algorithms in SPA: a response encryption algorithm and a response merging algorithm.

4.3.1 Response Encryption Algorithm

The pseudocode of response encryption algorithm is shown in Algorithm 1.

Input: r: a response

Output: $r_{encrypted}$: an encrypted response

1: $m$ = getSettingSize(r)
2: for $i$ = 1 to $m$ do
3: $r_{encrypted}.policy.setting.value$ ← homomorphicEncrypt($r.policy.setting.value$)
4: end for
5: return $r_{encrypted}$

4.3.2 Response Merging Algorithm

The pseudocode of response merging algorithm is shown in Algorithm 2.

Input: $r_1$: a encrypted response; $r_2$: another encrypted response

Output: result: a merging result

1: if $r_1.request.id = r_2.request.id$ then
2: result.requester ← $r_1.requester$
3: Create a new policy element $p$
4: $p.application$ ← $r_1.application$
5: $m$ = getSettingSize($r_1$)
6: for $i$ = 1 to $m$ do
7: Create a new setting element $s$
8: $s.name$ ← $r_1.setting.name$
9: $s.type$ ← $r_1.setting.type$
10: $s.content$ ← $r_1.setting.content$
11: $s.value$ ← homomorphic
12: ($r_1.setting.value$, $r_2.setting.value$)
13: Append $s$ to $p$
14: end for
15: Append $p$ to result
16: end if
17: return result

When a response comes, the cloud deals with the response as follows:

- When the cloud receives a response corresponding to a new request, cloud generates a new record in the database, storing the response related to the specific request.
- When the cloud receives a response corresponding to an existing request, the cloud starts a merging process, which merges the newcomer response with an existing response, and updates the record in the database.

Algorithm 2 shows how to merge two responses. If the number of responses from friends over two, algorithm 2 can process them iteratively. The method getSettingSize() returns the number of setting elements in a response. The method homomorphic() operates a homomorphic addition/multiplicative on two encrypted values. The time complexity of Algorithm 2 is $O(m)$, where $m$ is the number of setting elements in a response.

5. EVALUATION

5.1 Prototype Implementation

In our work, we implemented the SPA framework which builds on a partially homomorphic encryption algorithm,
The SPA framework consists of the client and server application.

- The server application forwards requests and responses between entities. It also functions as a semi-trusted cloud service to merge responses corresponding to a request.
- The client application is released as an .apk file. After installing the application on a smartphone, a user can log on to it using a registered account, and perform the SPA functionality when he or she wants to set privacy-aware policies.

### 5.2 Completeness of Supporting Policy Types

When the implementation of SPA leverages a partial homomorphic encryption algorithm due to performance concerns, we concern how many types of policies in mobile applications can be supported. In this section, we try to find out the completeness of supporting policy types of the implemented SPA.

We conduct experiments to evaluate the completeness of the implemented SPA by measuring the fraction of policies it can handle over the total number of policies. Firstly, we download the top 50 applications of each category on the ranking list of Google Play Store. These 22 categories include social network services, sports, finance and so on. Then we install these 1,100 applications on Android smartphones. For each application, we click to find out the settings related to privacy or security and calculate the number of policies in each type manually.

We conduct the first experiment to learn the percentage of policies that the implemented SPA can support in these 1,100 applications. We do the statistics work in the following two ways. One is to count the total number of applications that contain policies of each type (denoted as m), and we calculate the percentage as \( \frac{m}{M} \), where \( M \) is the number of usable applications for statistical analysis among 1,100 applications. The other is to count the total number of policies in each type (denoted as n), and we calculate the percentage as \( \frac{n}{N} \), where \( N \) is the total number of policies appearing in the 1,100 applications.

Not all the 1,100 applications contain privacy or security settings. For example, applications related to books and references rarely contain settings. There are also some applications we cannot even open due to the issue of Internet. There are 479 applications of these two kinds in total, and the rest 721 applications are usable for statistical analysis.

Table 5 shows the number and percentage of applications that contain policies of each type. About half of the usable applications contain policies of Switch type. Since the implemented SPA can support the defined four types of settings, the percentage of applications that SPA cannot completely support is 27.18%.

Table 6 shows the number and percentage of policies of each type. Nearly half of the total policies are of Switch type. As the implemented SPA supports the defined four types of settings, the percentage of policies that the implemented SPA cannot support accounts for 16.48%.

In the second experiment, we select a specific SNSs application as the test case, to learn the percentage of policies that the implemented SPA supports. We conduct the experiment on WeChat, a mobile text and voice messaging service developed by Tencent in China [29]. It is a very popular instant messaging service in Chinese community. The version we employ here is WeChat 5.3 for Android. The policies appear in WeChat are shown in Table 7. We simulate these settings in the implemented SPA.

In our experiment, the SPA works well except when the type of policies is List. There are two types of policies in WeChat: Switch and List. We have not defined the List type, so the implemented SPA cannot deal with this kind of policies. However, we think this kind of policies is not appropriate for policy recommendation. The reason is that users can take advantage of blocked list or hidden friends to know who are less popular in the friend group, which leads to privacy problems.

### 5.3 Performance Evaluation

In order to provide an acceptable security strength of the SPA framework, there is much computational work on both the client and server application. For the client application, it needs to encrypt a response before sending it to the server; For the server application, it takes time and space to do a homomorphic merging operation.

We conduct the first experiment to evaluate the performance of the server application. The server application runs on a PC running Windows 7 professional with 4.00 GB memory and Intel(R) Core i5-2400 CPU @ 3.10GHz processor. We vary the number of responses that the server application receives at the same time, and record the time that the server application needs to process these responses. The experimental result is shown in Figure 5.

In Figure 5, the horizontal axis represents the number of responses that the server receives at the same time, and the vertical axis represents the time that the server needs to deal with these responses. The main time costs lie on merging responses.

We conduct the second experiment to evaluate the performance of the client application. The two metrics of performance are: time to process requests and CPU occupancy rate on Android smartphones. We conduct the experiment on different kinds of smartphones running Android 4.2. These smart phones include a Nexus 4 with a 8.00G memory, a Galaxy S4 with a 16.00G memory, and an HTC Desire 816w.
The implemented SPA supports major types of policies of the current popular applications with acceptable performance.

### Table 7: Policies in WeChat

<table>
<thead>
<tr>
<th>Setting</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friend Confirmation</td>
<td>Confirmation before becoming friends</td>
<td>Switch</td>
</tr>
<tr>
<td>Find QQ Contacts</td>
<td>Recommend friends on QQ to me</td>
<td>Switch</td>
</tr>
<tr>
<td>Find Me by QQ ID</td>
<td>Find me by searching my QQ ID</td>
<td>Switch</td>
</tr>
<tr>
<td>Find Me by Phone Number</td>
<td>Find me by searching my phone number</td>
<td>Switch</td>
</tr>
<tr>
<td>Find Mobile Contacts</td>
<td>Recommend friends in mobile contacts to me</td>
<td>Switch</td>
</tr>
<tr>
<td>Find Me by WeChat ID</td>
<td>Find me by searching my WeChat ID</td>
<td>Switch</td>
</tr>
<tr>
<td>Blocked List</td>
<td>List of friends who cannot send messages to me</td>
<td>Others</td>
</tr>
<tr>
<td>Do Not Share My Moments</td>
<td>List of friends who cannot see my WeChat status</td>
<td>Others</td>
</tr>
<tr>
<td>Hidden Friends</td>
<td>List of friends whose WeChat status I do not want to see</td>
<td>Others</td>
</tr>
<tr>
<td>Public Moments</td>
<td>Whether strangers can visit my WeChat status</td>
<td>Switch</td>
</tr>
</tbody>
</table>

### Figure 5: Server Performance

Figure 5 shows the server performance with a 8.00G memory. The experimental result is shown in Figure 6.

In Figure 6, the horizontal axis represents the number of requests that a client application receives at the same time, and the vertical axis represents the time that the client application needs to deal with the requests. The main time costs lie on encrypting a response before sending it out.

We further measure the performance of the client by connecting the smartphone device to a computer, and executing “adb shell top” command to view ongoing tasks running on the device. We monitor the status on a Google Nexus 4 running Android 4.2 several times. We can see that the peak value of CPU usage of the client application does not exceed 7%, including encrypting a response before sending it out.

Based on the above experiments, we can see that the implemented SPA supports major types of policies of the current popular applications with acceptable performance.

### 6. DISCUSSION

#### 6.1 Vulnerabilities in SPA

One vulnerability exists when the requester acts as an attacker. The requester sends a request to only one friend and other accounts which are also managed by the requester himself. When the only friend makes a response, it is possible for the requester to reason the settings of the friend based on the final merging result and his own settings. The vulnerability also exists when a requester colludes with other friends to crack into the settings of a targeted friend.

There exist several vulnerabilities in $w$-SPA. According to the SPA framework in subsection 4.1, a requester sends requests to respondents directly. Thus, weights, which are the privacy of the requester, is sent to respondents. If respondents exploit this, they could get requester’s personal judgments of them. It is also possible for a requester to get all settings from respondents who send responses respectively. The requester can use carefully designed weights to store all responded respondents’ setting values in different digits of weighted sum. E.g., Alice wants to get each friend’s setting value, which ranges 0 to 100, from Bob, Cindy, Dale, who are her friends. She can set Bob’s weight to 1, Cindy’s weight to 1000, and Dale’s weight to 1000000, and send requests to them. These three weight values ensure Alice’s friends’ setting values store in weighted sum respectively. (If there are some friends who do not respond Alice’s request, the value will be 0, and it will not affect others setting values) The cloud server will send the encrypted average value of weighted sum to Alice. Then Alice can get setting values from respondents who responded her request. As a result, the mechanism to support $w$-SPA requires more concerns to protect friends’ privacy.

#### 6.2 Using Fully Homomorphic Encryption to Support More Types of Policies

Although there is still no effective and efficient solution, we believe the SPA framework can support more types of policies using fully homomorphic encryption. Our work has implemented SPA framework using a partially homomorphic encryption algorithm, Paillier. Experimental results show that the implemented SPA supports the majority of policies in current popular Android applications. However, the lim-
6.3 Security of Homomorphic Encryption

The security of homomorphic encryption affects the security strength of the SPA framework. Shannon formalized the security of encryption schemes for the first time in the literature [21]. Shannon introduced the notion of perfect secrecy/unconditional security, which characterized encryption schemes in which the knowledge of a ciphertext does not give any information about either the corresponding plaintext or the key [9]. The highest security level a homomorphic encryption can reach is IND-CPA [9]. IND stands for indistinguishability whereas CPA are acronyms for chosen plaintext attack. A chosen plaintext attack (CPA) is an attack model for cryptanalysis which presumes that the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts [1].

RSA cannot achieve a security level of IND-CPA, but Paillier and ElGamal achieve the highest security level for homomorphic encryption schemes. However, RSA is still considered strong enough.

7. RELATED WORK

This paper proposes a novel policy administration framework, socialized policy administration (SPA) to manage personal policies, where users can invite their friends to help set sensitive policies.

Policy administration is an effective approach to protect and operate information systems [23][15]. The literature [16] specifies four core components in a traditional framework of policy-based management: Policy Decision Point (PDP), Policy Enforcement Point (PEP), Policy Administration Point (PAP), and Policy Repository (PR). In the traditional administration model, a professional expert or group will take charge of the policy administration, whose functions include policy design, policy verification, and policy deployment [12]. Many researchers proposed their policy administration methods [20][14]. However, smarter phones and mobile applications challenge the existing trust model in the policy administration, where common users do not possess professional knowledge of policy-based management.

Therefore, much work has been done to meet new requirements in policy administration. In the literature [24], Squicciarini et al. pointed out, in spite of the fact that content sharing represents one of the prominent features of existing Social Network sites, Social Networks yet do not support any mechanism for collaborative management of settings for shared content. Squicciarini et al. modeled the problem of collaborative enforcement of privacy policies on shared data by using game theory. In particular, they proposed a solution that offers automated ways to share images based on an extended notion of content ownership. The approach makes use of the concept of shared ownership of data. This is achieved by having the originator of the data, that is the user responsible for uploading the data, specify other potential owners of that data. The system then holds an auction on the possible privacy policy to apply to the data in which all the owners submit a vote for their desired policy. The literature claims to be the first research to discuss a novel model for privacy management across social networks, where data may belong to many users.

In the literature [30], Wishart et al. pointed out content sharing on social network services may lead to privacy problems. The literature proposes a privacy-aware social networking service and then introduced a collaborative approach to authoring privacy policies for the service. The approach permits the originators of content on the social network to specify policies for the content they upload. The conditions under which the policy applies can then be edited by nominated users of the social networking service.

The literature [22] proposes policy recommendation. Mohamed Shehab and Said Marouf proposed a multicriteria recommendation model that utilizes application-based, user-based, and category-based collaborative filtering mechanisms. Collaborative filtering mechanisms are based on previous user decisions, and application permission requests to enhance the privacy of the overall site’s user population.

In the literature [12], Han et al. proposed a policy administration mechanism, referred to as collaborative policy administration (CPA for short), to simplify the policy administration. In CPA, a policy administrator can refer to other similar policies to set up their own policies to protect privacy and other sensitive information. To obtain similar policies more effectively, a text mining-based similarity measurement method is presented.

Existing approaches of collaborative policy authoring or administration involve cooperators or friends. However, these methods rarely focus on protection of privacy of who participate in the collaborative policy administration process. Instead, this paper pays attention to the privacy of friends who help set our applications. We implemented an enforcement SPA framework by using homomorphic encryption, and experimental results show that the proposed mechanism can supports major types of policies with acceptable performance.

8. CONCLUSION AND FUTURE WORK

To the best of our knowledge, this paper is the first research to study privacy-preserving solutions to enable unprofessional users to set sensitive policies with the help of their friends on social network services. We firstly articulate the problem of socialized policy administration (SPA for short), where a user can request his or her friends to help set sensitive policies. We then propose an SPA framework for a mobile scenario, and implement the SPA framework using a partially homomorphic encryption, Paillier. We conduct experiments to evaluate the completeness and performance of the proposed SPA framework. The results show that the framework supports the majority of policies in current popular Android applications, and the performance is promising to support these policies.

In our future work, we will leverage some formal methods to analyze the security of the SPA framework. In addition, we will design a new schema to support w-SPA with privacy preserving. We will focus on the implementation of a practical SPA component for users to integrate into their applications. Last but not least, we will investigate a method for a user to find other users who may have similar usage scenario with him or her to make the result more suitable for the requester’s usage scenario.
Acknowledgement

This paper is supported by National Key Science and Technology Program (P01-029-2014(10)-2-4-02-T-C), Natural Science Foundation of Shanghai (12ZR1402600), Twelve-Five National Development Foundation for Cryptography (MMJJ 201301008), and Shanghai Science and Technology Development Funds (13dz2260200, 13511504300). We thank anonymous reviewers for their comments.

9. REFERENCES