MoRePriv: Mobile OS Support for Application Personalization and Privacy

Drew Davidson
University of Wisconsin
davidson@cs.wisc.edu

Matt Fredrikson
University of Wisconsin
mfredrik@cs.wisc.edu

Benjamin Livshits
Microsoft Research
livshits@microsoft.com

ABSTRACT

Privacy and personalization of mobile experiences are inherently in conflict: better personalization demands knowing more about the user, potentially violating user privacy. A promising approach to mitigate this tension is to migrate personalization to the client, an approach dubbed *client-side personalization*. This paper advocates for *operating system support* for client-side personalization and describes MOREPRIV, an operating system *service* implemented in the Windows Phone OS. We argue that personalization support should be as ubiquitous as location support, and should be provided by a unified system within the OS, instead of by individual apps.

We aim to provide a solution that will stoke innovation around mobile personalization. To enable easy application personalization, MOREPRIV approximates users' interests using *personae* such as technophile or business executive. Using a number of case studies and crowd-sourced user studies, we illustrate how more complex personalization tasks can be achieved using MOREPRIV.

For privacy protection, MOREPRIV distills sensitive user information to a coarse-grained *profile*, which limits the potential damage from information leaks. We see MOREPRIV as a way to increase end-user privacy by enabling client-side computing, thus minimizing the need to share user data with the server. As such, MOREPRIV shepherds the ecosystem towards a better privacy stance by *nudging* developers away from today's privacy-violating practices. Furthermore, MOREPRIV can be *combined* with privacy-enhancing technologies and is complimentary to recent advances in data leak detection.

Keywords

Security, privacy, personalization

1. INTRODUCTION

Mobile applications are becoming increasingly *personalized*, fluidly adapting themselves to the needs and preferences of their users. Today, personalization typically aggregates user data in the *cloud* and uses it for large-scale data mining. This approach has significant advantages for data aggregators, allowing them to present targeted services and ads. However, as the ongoing DoNotTrack debate illustrates, not all users are comfortable with sharing information with large companies such as Facebook or Apple to obtain personalized results. Recently, a series of papers has highlighted the severity of data leak problems in mobile apps [16, 32, 20]. However, existing work in this domain usually only *detects* when apps use personalized data (in which case the user's only remedy is to not install the app at all), or falsifies the data that apps access [25]. In both cases, the user forgoes the benefits of personalization.

To balance the goals of privacy and personalization, we implemented mobile OS support for rich personalization in the form of MOREPRIV[1] Privacy-aware personalization is a rapidly expanding research area, often taking the form of client-side personalization. Encouragingly, a recent study has demonstrated users' generally positive attitudes towards client-side personalization [29]. Much of the state-of-the-art client-side personalization work focuses on solving a specific problem, such as privacy-conscious ad targeting [44, 22]. Similarly, we observe a trend towards platform-provided ad support in efforts such as Window 8's Ads in Apps[2] and Blackberry's integrated Advertising Service[3]. In contrast, the focus of MOREPRIV is on creating a platform to unleash developer creativity across a variety of personalization scenarios.

Client-side personalization: MOREPRIV encourages *leaving* user data on the mobile device, mediated by the operating system and under the control of the user. This approach has many benefits: the user retains control over data, and cloud providers do not have to worry about properly securing data, respecting the wide array of user privacy preferences, complying with local and international laws, running and powering expensive storage clouds, dealing with PR repercussions of data leaks and unauthorized tracking (such as the Apple location data scandal), etc.

OS support: When user preference information is collected from all apps and OS interactions, it is considerably more accurate compared to what any given app, even the mobile browser, can obtain individually. Furthermore, the user already implicitly trusts the OS, so performing personalization there does not expand the trusted computing base. In MOREPRIV, personalization functionality draws from a single unified, trustworthy, and protected source provided by the OS and exposed to developers via a simple API.

Personae: To enable easy application personalization, MOREPRIV approximates user's interests using *personae* such as technophile or business executive. Personae can be used for a basic level of personalization, which can be extended through the use of *custom classifiers*. We demonstrate how always-on user interest mining can effectively and accurately infer user interests in a mobile operating system. MOREPRIV inference is based on parsing and classifying

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.
ACSAC '14, December 08 - 12 2014, New Orleans, LA, USA
Publication rights licensed to ACM.978-1-4503-3005-3/14/12 ...$15.00
http://dx.doi.org/10.1145/2664243.2664266 .

[1] for Mobile OS for Reclaiming Privacy
[2] http://adsinapps.microsoft.com/en-us
[3] adservices.blackberry.com

multiple streams of (sensitive) information about the user within the OS, such as their email, SMS, Facebook stream, and network communications. For privacy protection, this sensitive information is distilled to a coarse-grained profile (as opposed to fine-grained location data, whose leaks have caused much chagrin among users and privacy advocates), without being exposed to apps. The use of personae within the user profile limits the potential for user tracking: while persona information can be shared by apps to perform server-based personalization, for instance, it is not sufficient to link an individual user across multiple discrete interactions.

1.1 Contributions

In this paper, we

- propose OS-wide personalization to achieve personalization and privacy on mobile devices, illustrate key opportunities for mobile operating system designers, and suggest ways in which mobile applications can be personalized;
- describe our implementation of MOREPRIV on the Windows Phone OS; and evaluate the use of personae, through use cases and end-user experiments;
- demonstrate automatic personalization of legacy applications through case studies and show how MOREPRIV APIs can be used by new apps for personalization and skinning;
- show how to support custom classifiers, the integrity of which can be verified using the machinery of zero-knowledge proofs to provide assurances to 3rd parties when the user has incentive to fake personalization measurements;
- establish that using zero-knowledge proofs in our context is practical in terms of computational demands experimentally, finding that a classifier using 100 feature words requires only 11 seconds of prover time, less than a second of transfer time on a 3G connection, and negligible battery usage;
- present crowd-sourced user studies showing how MOREPRIV personalization can be effective in a widely-used app such as Yelp.

1.2 Paper Organization

The rest of this paper is organized as follows: Section 2 provides the background for mobile privacy and personalization. Section 3 gives an overview of MOREPRIV. Section 4 describes how we implemented MOREPRIV by modifying the Windows Phone operating system and demonstrates the MOREPRIV API. Section 5 discusses support for custom classifiers and describes how to provide integrity of computed classification results using zero-knowledge proofs. Section 6 illustrates the benefits of mobile personalization through case studies and describes the result of our user studies. Section 7 summarizes the related work and Section 8 concludes.

2. BACKGROUND

The prevalence of smart phones and other mobile computing devices has opened up new avenues for personalized applications. As users carry these devices with them wherever they go, they are subject to a level of user interaction never before seen on a personal computing platform. Furthermore, they are typically equipped with cameras, microphones, GPS, and several forms of wireless networking, providing a constant stream of data from both the physical world and the Internet. Common examples of personalization include Siri, Google Now, and personalized local search. All of these currently move a considerable amount of user data into the cloud for processing. We begin by discussing compelling representative examples of mobile personalization and then proceed to generalize the requirements for a successful personalization platform.

Personal Assistants: Google Now and Siri are familiar examples of mobile personalization. Both attempt to perform high-level tasks on behalf of the user, such as planning commutes, selecting movies, and making wine recommendations. Over time, these systems learn the user's preferences and personalize recommendations accordingly, although the mechanisms by which this happens is proprietary. If the user invokes the services frequently, they become privy to a sizable portion of the user's day-to-day activities, giving a valuable window into the user's life when targeting content and advertisements. However, this data is processed in the cloud and is potentially visible to third parties affiliates for analysis. As such, both the information given directly by the user such as "set a birthday reminder" and secondary information such as gender and age may be obtained in bulk.

Shopping: Shopkick (`shopkick.com`) integrates location-awareness with user preference data. When a Shopkick user enters a brick-and-mortar store, the app consults the user's preferences to offer discounts and recommendations. Users can scan the barcodes of interesting items and are rewarded with discounts within the store. This personalization is interesting in that it allows traditional merchants to to target content, advertisements, and discounts towards users in much the same manner as online retailers do.

Summary: The apps above share the following set of personalization steps [28]: 1. Acquire personalization signal from user interactions with the app; 2. Address the *cold start* problem of not having data to base personalization on [38]; often, this is resolved using a secondary source. Many modern apps encourage the user to sign up with their Facebook credentials so they can "scrape" their Facebook data; 3. Refine personalization signal as a result of subsequent interactions.

3. OVERVIEW

The apps described in Section 2 are representative of the personalization needs of many apps: they benefit from an understanding of the user, but need not collect enough information about the user to personally identify them. In this Section, we describe how MOREPRIV provides this information through the key notion of *personae*.

3.1 User Interest Profile: Personae

Personae are custom representations of various walks of life, offering easy-to-understand targets for personalization [40]. In addition to being easy to understand, the advantage of this technique is the degree of *pseudonymity* that it provides [30]. In other words, using a persona provides a way to declassify sensitive information; persona data is all that is released about the user to the application, as opposed to personally-identifiable information like their name or unique identifiers such as device IMEI.

In our prototype implementation of MOREPRIV, we target *eight* personae. Each persona is represented by a Bayesian classifier C_p, trained on a manually curated list of keywords characteristic to profile p. For example, the business executive persona represents strong interest in business, finance, and national news. Thus, the corresponding classifier is trained on text from sites as the Financial Times. On the other hand, the technophile profile represents a strong interest in technology, so the corresponding classifier is populated with text from tech blogs.

Our prototype of MOREPRIV supports the following personae: activist, bachelor, business executive, sports buff, retiree, homemaker, technophile, tween. Note, however, that we do not advocate for this specific list of personae. Our goal in choosing the set of personae is neither completeness nor exclusivity: arguably, it is very difficult to be comprehensive; similarly, there may be intersections between these personae. Instead we optimize for ease of understanding by

the user and the developer. We believe these profiles are a reasonable proof of concept but note that our system is modular with respect to the personae that are used, with updates amounting to simply re-training a Bayesian classifier on a new list of keywords.

In practice, no user is likely to have interests that match exactly one persona. As such, each persona is assigned a *persona weight* that indicates how closely that persona matches the user. MOREPRIV keeps a vector of these weights, called the user's *profile*. Consider an example user who is very interested in technology news and somewhat interested in financial news. His or her profile would have a high persona weight for the technophile persona and a moderate persona weight the executive persona. We believe that many users understand the value of personae: we performed a brief anonymous survey of 179 crowd-sourced survey-takers, 39–45% of respondents are more willing to share *persona* data, whether in general or for the purposes of personalization, compared to the familiar baseline of location data.

3.2 Design Choices

The architectural of MOREPRIV is divided into three parts: User interactions are observed by *personal preference miners*, that can be as diverse as location information miners that tell whether the user is home or on a trip, or transit miners that discover the activity of the user (walking, riding a train, in a car, etc.). Miners also pore over the user's email, SMS, and Facebook streams to establish the user's interests and preferences in news and entertainment. These interactions are distilled to a user profile by *persona classifiers* that come pre-installed (we discuss details of extending the system via custom classifiers in Section 5). Finally, the profile is used by a variety of *personalizers* for different forms of personalization. user's interests and preferences for news and entertainment.

A key strength of MOREPRIV is that users remain in control of whether and to what extent to use it, in order to appeal to a wide spectrum of users. Previous studies have shown that users' attitudes towards mobile privacy vary greatly: conservative users may opt-out and turn off personalization or data collection *entirely* Section 4.4. Furthermore, conservative users will likely not to install *any* third-party classifiers. More adventurous users who want to experience extra functionality may install custom classifiers, but will be prompted for additional permissions at the time of installation (Section 5.2).

3.3 Data Protection and Privacy

So far, our discussion of MOREPRIV has primarily focused on the personalization benefits of our approach. In the rest of this section, we focus on privacy-related advantages. In the past several years, numerous projects have examined data leakage potential on the mobile phone, including mobile information leakage [16, 20, 14], leading to privacy-enhancing technologies designed to guard against such data leaks, such as AppFence [25].

We want to draw a contrast between that line of research and MOREPRIV, which is envisioned as a privacy-by-design approach. The goal of MOREPRIV is to *incentivize* the different constituents of the mobile ecosystem into compliance. MOREPRIV can and should be combined with privacy monitoring and privacy leak detection tools, but we see that as an orthogonal concern.

Use of personae limits information leaks: At the core of the MOREPRIV design is the use of personae, which provides a degree of pseudonymity [30]. Sensitive data is distilled by MOREPRIV to a restricted, deliberately coarse-grained persona, and is never given to user mode apps directly [37]. A natural direction to consider is implementing an information flow restriction approach to mitigate leaks of persona information. While many research efforts have argued for information flow and tainting approaches, in both static and runtime flavors at levels varying from hardware to the application runtime, practicable adoption has been slow. We argue that this is because it is very difficult to deploy these systems without causing numerous false positives or tolerating many false negatives. Consequently, we choose not to following this path in MOREPRIV.

Data representation: Our implementation of MOREPRIV maintains a vector of persona weights for the user. This vector is maintained in memory and it potentially serialized as part of the OS service that MOREPRIV implements. To make data maintenance incremental, our implementation records two values for each persona p (out of eight possible personae):

1. *relevance*: sum of interest scores from classifier C_p, s_p
2. *support*: number of elements that have been scored by C_p, called n_p.

Intuitively, s_p indicates a raw score of how closely the persona matches the user, while n_p indicates the amount of evidence to support that score. The persona weight is given by the fraction s_p/n_p. Thus, only two data items per persona are stored. Note that these vectors are never directly shared with user mode apps.

While this is not encouraged, in our model, apps are allowed to leak the most relevant persona of the user if they so desire. However, we consider the consequences of that to be relatively benign, especially when compared to the current practice of user monitoring built into many of today's privacy-leaking apps [24]. MOREPRIV is designed to use a limited number of personae such that each persona will be the most relevant for many users. Thus, leaking the top persona of a user has a negligible contribution towards identifying a user and does not identify any specific interests of the user. Those users that are particularly concerned about leaking their top persona can still disable apps from accessing this information.

Permissions: As we saw before, access to persona data in apps is guarded with a runtime permission prompt, similar to that used for obtaining location data. This is superior and more meaningful to the user compared to installation-time permissions [17]. We believe that this is part of responsible disclosure: the user is informed of persona data access and is given the opportunity to opt in. Furthermore, access to MOREPRIV's persona data requires statically declared permissions at the level of application manifest. Just like with location data, we can envision extra scrutiny, code reviews, and testing being applied by the app store to apps that request persona data.

Data synchronization: While the default storage strategy is to keep the interest profiles local, on the current device, it is entirely possible to synchronize them — in an encrypted form — with the cloud. This is not unlike the approach used in Apple's iCloud for synchronizing application settings, etc. However, unlike application settings, persona information encroaches on user privacy considerably less. In addition to synchronization across multiple devices, cloud synchronization also serves as a backup. This approach is used in several domains, including bookmark synchronization, etc.

3.4 Value Proposition

Our goal with MOREPRIV is to provide an attractive value proposition for all major categories of constituents within the mobile ecosystem: users, developers, and third party data and ad providers. We summarize the benefits below.

Users: Recent user studies indicate that users often prefer client-side personalization to alternatives [29]. Additionally, MOREPRIV learns an accurate model of the user's tastes and preferences with-

out the user having to "teach" or configure it. To make it easier for users to understand, we model the way persona information is surfaced to the user after GPS location data; the figure to the right shows an example of a persona prompt within an app. Users can *audit* persona information released to every application, to understand when each release has taken place. Users can opt of of data collection, either partially or entirely. Lastly, for users who are strongly concerned about privacy, MOREPRIV is designed to be combined with privacy-enhancing technologies.

Developers: The main advantage for app developers is the MOREPRIV API. It represents a single OS-level data source for personalization (as opposed to application-specific, ad-hoc information sources) allow seamless and uniform functionality for a single user across many applications, devices, and platforms. It is easy for developers to use for personalization and skinning, as illustrated in Section 6.1. The *cold start* problem [38] common in many personalization tasks is largely addressed, because of an OS-wide user personalization context, which apps can easily take advantage of. Finally, OS-wide data collection for a single repository allows richer and more accurate profile information about the user to be collected, compared to what any single application can accomplish.

Third-party and ad providers: Data providers are no longer responsible for storing "toxic" user data and cross-correlating it across a set of user interactions. Given the mounting pressure by legislators in the US and Europe to limit the impact of online tracking as well as mobile app tracking, this can be welcome news. Instead of trying to engage in user profiling, which is 1) costly, because it requires maintaining or buying data center capacity, developing custom software, and paying data mining and support personnel, and 2) cumbersome, because multiple laws of different localities must be respected, and there is always a danger of being limited later in time, a clear alternative is to use the information provided natively, by the mobile OS [2, 1]. While some third-party provides will certainly prefer to stick with the state of affairs, less entrenched ones will find the new approach attractive.

4. IMPLEMENTATION

In order to test the effectiveness of personal preference miners, we instrumented Windows Phone 7.5 (Mango) to capture several important *personalization signals*, sources of data that indicate likely user preferences. We then use these signals to locally classify the user with respect to the given personae. There are two facilities for personalizers in MOREPRIV: a privileged service to perform automatic personalization within the OS, and a set of APIs that give third-party applications limited access to the user interest profile. We chose to implement MOREPRIV on top of the Windows Phone operating system. Similar implementations on Android or iOS may be possible, and we point out areas in which we take advantage of features of the Windows Phone and possible analogues in other mobile operating systems. However, cross-platform results are not the focus of our work.

4.1 Personalization Signals

In order to assign relevance scores to each persona, MOREPRIV needs data to classify. Here, we leverage our position at the OS level: all user information must pass through the operating system in order to be consumed or produced by the user. However, one must be careful of *which* data stream to collect: a poor choice can slow down the device or introduce noise. One must also consider *how* data is collected: the collection mechanism should be positioned at a level of abstraction such that the data has appropriate context. For example, one may be interested in mining the text of web pages that the user views, but if the miner interposes at a low level of the protocol stack, bytes of text will be indistinguishable from bytes of an image.

In our implementation, we capture five distinct personalization signals: Facebook data, the Twitter feed, SMS messages, emails, and HTTP traffic. We briefly discuss our approach to mining each of these signals on Windows Phone, and we mention how equivalent signals could be captured for an Android Device.

Facebook and Twitter: A unique feature of the Windows Phone is that several popular networking features are integrated directly into the operating system and organized into the *People Hub*. The intention of the People Hub is to organize social updates in a single, unified feed called the social feed, which is updated automatically. The social feed is a good target for mining because it is a rich source of structured user data. We implemented a miner for Facebook by reading social feed data from the Facebook service, consisting of "likes", posts that the user made, and posts made to the user's wall.

There is no direct analogue to the People Hub on Android. However, since the account credentials are stored in the `AccountManager`, an Android device could make separate queries through the APIs exposed by high-value services like Facebook and Twitter, and classify the results of those queries. Note that this approach loses an advantages of MOREPRIV, namely that it does not consume additional network bandwidth.

SMS: due to the simplicity and inherent lack of structure in SMS messages, we implemented our miner by interposing on the SMS handler in native code underlying the application framework. Alternatively, similar modifications can be made to the C# core libraries to read SMS messages. To implement a similar miner in Android, one could periodically query the `ContentResolver` for SMS content, and classify each SMS message in turn.

Email: Sending email is exposed to third party users via the `Microsoft.Phone.Tasks` classes such as `EmailComposeTask`. However, in order to simplify capture of both outgoing and incoming email, we instead interpose on the internal implementation of SMTP. Although we do not treat fields such as *subject* and *sender* differently from text in the body, interposing on SMTP allows us to avoid classifying noise, e.g. attached images.

HTTP Traffic: Unlike SMS, HTTP traffic has structure that cannot be ignored. Fortunately, the Windows Phone passes information to an HTTP handler which parses the structure of the message. By interposing on the parser as it parses text, we can gather relevant web text without adding significant noise from non-textual traffic.

These signals demonstrate an advantage of performing signal capture at the Operating System level: since the OS and framework have a very high level of privilege, the user must already trust these components to handle personal data. As such, the signal capture mechanisms are already within the user's trusted computing base. Instrumentation at the OS level has the unique advantage of being able to integrate multiple data sources together. This is important for several reasons. Even very rich data sources can suffer from a cold-start problem, but are useful in aggregate.

To drive this point home, we conducted a study to see how strongly a user with a high interest in technology would be classified as a technophile. While we would expect this score to be high, the quality of different signals in isolation varies significantly. However, *combining* these signals together can boost the correct relevance score even in the face of irrelevant signal data.

4.2 Building MOREPRIV Classifiers

Each MOREPRIV persona is represented by a Naive Bayesian classifier. We trained our classifiers offline on manually curated lists of words obtained for web pages relevant to each profile, e.g., `techcrunch.com` for technophile and `espn.com` for sports buff. An

alternative source of such pages are taxonomies such as the Open Directory Project (ODP).

This resulted in thousands of words per persona, which were subsequently used to obtain the probabilities $P(w_i|C_j)$ for each attribute word w_i and each persona C_j. This classification data was then loaded into an OS-level service.

Although building each classifier can in principle be a difficult task, especially if a large training set is used, applying classification to a piece of text is very fast. For example, to find the most relevant profile for a piece of text, we tokenize it into words and perform a simple log-likelihood addition for each persona, maximizing over that value.

4.3 Universal Personalization

In the context of MOREPRIV, we implemented *automatic* universal personalization within the OS. To accomplish this, we modified the Windows Phone C# framework upon which apps are built. We focused on reordering lists such that elements that are of the most interest to the user are displayed at the top, while items of less interest to the user are kept at the bottom. Widget classes like `System.Windows.Controls.ListBox`, are (directly or via subclass) used in many third party apps to display lists. Our modification changed the way in which list items are displayed on screen, ordering them by their relevance to the user profile. In essence, this allows lists to be transparently personalized without any modification (or knowledge) of the app. An important consideration for universal personalization is not to personalize "too much". For example, if automatic personalization were to be applied to an alphabetized list, the alphabetic ordering would be lost. Thus, we detect if a list has been sorted, and if so we do not use the personalized list ordering to draw elements, instead relying on the ordering of the internal `ItemSource` list.

We have made a small number of changes to the Windows Phone core C# classes to inform our universal personalization mechanism that a list has been sorted, such as modifying the `List.Sort()` method to keep a `sorted` flag. When set, the flag causes UI classes like `ListBox` to display in their natural order. We emphasize that universal personalization is often effective, but recognize that automatically restructuring UI is not always guaranteed to work. For this reason, we allow universal personalization to be disabled through the MOREPRIV management menu.

4.4 OS-level Service

Positioning MOREPRIV within the OS provides an opportunity to collect a great deal of data to build a user interest profile. However, it also provides an opportunity to perform personalization on user-level apps without any modification of the app itself.

To explore automatic personalization, we altered the Windows Phone C# framework to reorder lists in the application UI, as described in Section 4.3, based on the persona weights. For legacy applications such as news readers, this has the effect of not only reordering the order in which stories are displayed (stories more relevant to the user's interests are shuffled to the top), but also reordering entire categories of subjects such that the "technology news" category page of a news reader app appears earlier in the menu than the "arts section" for a technophile. Finally, users can control how MOREPRIV is used by toggling two independent facilities:

Toggling personalization: When personalization is off, the user will have the regular Windows Phone experience. When personalization is on, OS-level personalization is enabled and apps have access to the user's persona.

Toggling collection: This allows users to to freeze their profile scores. This indicates to MOREPRIV that it should not track any behavior of the user until persona refinement is re-enabled. This is a form of a privacy mode, similar to those supported in modern browsers.

4.5 MOREPRIV APIs

MOREPRIV also exposes APIs to third party developers that allow application-specific personalization. We discuss three of these API functions:

- `IsMoRePrivEnabled()` returns true if personalization is enabled. We allow users to toggle personalization on and off as part of the MOREPRIV configuration UI.
- `TopProfile()` return the most relevant profile to the user if personalization is enabled, and `null` otherwise. This is useful for application *skinning*, for example by changing the background of the app based on the top profile of the user.
- `Ignore(Object o)` Informs MOREPRIV not to apply OS-level personalization to `o`. This allows developers to bypass the GUI features such as automatic list reordering.

We have also written a wrapper library that checks for these API functions and calls them if present. Thus, app developers can write a single app that will work on both MOREPRIV-enabled and plain OSes (albeit without personalization).

5. INTEGRITY AND EXTENSIBILITY

There is growing evidence that the need for trusted user identities in advertising is growing[4]. These cases are emblematic of the commonly occurring case in which a 3rd party requires knowledge that the classification of users is accurate and trustworthy. A signature-based scheme may prevent benign users from loading hacked classifiers, but when it comes to malicious users, no such recourse is available. While the operating system itself is trusted by the user, the 3rd party may not trust the integrity of apps and classifiers: malicious users can install or "sideload" a hacked classifier to send fabricated data.

Instead, we extended MOREPRIV to optionally use zero- knowledge proofs to provide a solution to the problem of *integrity* of user profiling, effectively guaranteeing to third parties that the personalization signal can be trusted through *checkable* proofs of computational integrity.

5.1 Design Choices

In some cases, the information provided by the default classifiers is not sufficient for personalization. However, while we want to support greater extensibility, we do not want to expose sensitive user data to untrusted 3rd parties.

In this paper, we consider two ways to architect the system we have described. The *monolithic* architecture provides only the built-in, default user classifier. However, we acknowledge that in some settings, more application-specific classifiers may be called for. Since classifiers may need to be upgraded as a result of retraining, the *customizable classifier* approach allows for custom classifiers to optionally be installed at the discretion of the user.

Monolithic: The model discussed in this paper thus far assumes a single monolithic set of MOREPRIV classifiers which are implemented as an operating system service. As such, they are part of the TCB. A consumer of the user profile can trust the profile computation under the assumption that the operating system has not been tampered with. We feel that this is a sensible assumption,

[4]See `http://bit.ly/13hJpNk`, `http://tcrn.ch/PR5ZDL`, `http://bit.ly/1bT5mFD`, `http://on.wsj.com/1e57ms1`, `http://bit.ly/1bT5mFD`, and [8] for discussions of advertising fraud and its implications.

since tampering with the OS can invalidate the integrity of the observations on the basis of which the user profile is calculated. The main shortcomings of this monolithic approach is the difficulty in upgrading the classifiers and the fact that that classifiers are part of the TCB.

Customizable: An alternative model discussed in the rest of this section allows for custom classifiers to be installed in the same way that regular apps are installed, without affecting the OS and adding to the TCB. The user is responsible for checking installation- and runtime permissions to ensure these classifiers do not leak data inappropriately, just as they are for any other mobile app. Alternatively, as proposed in several recent projects [46, 26, 14, 16, 25, 31], classifiers can be subjected to static or runtime analysis to gain further assurance.

5.2 Deploying Custom Classifiers

In our model, custom classifiers are distributed as third-party applications in the same manner as other third-party software on mobile platforms. When the user installs a classifier application, he is prompted for permission to let the app interact with MOREPRIV by monitoring keywords among the personalization signals discussed in Section 4.1; this permission is requested at the same time as the other platform-specific permissions supported by the operating system. When the classifier is installed, MOREPRIV registers the needed keywords that are listed in the classifier's manifest.

To perform classification, the application queries MOREPRIV via an exposed API that returns the number of times each feature word was observed in the signals as a vector $\mathbf{w} = [w_0, \ldots, w_n]$. After receiving an updated set of word counts, the classifier runs its custom algorithm, and uses the result as needed. Whether this classification step is performed *periodically* in the background, or *on-demand* when requested, is an application-specific design choice left up to the classifier developer. Similarly, the choice of when to provide a proof of integrity is left to the developer; it can happen each time a new classifier is constructed (for an absolute guarantee of integrity), or only when the classifier consumer decides to *challenge* the client (for a probabilistic guarantee).

5.3 Motivating Example

Suppose that a third-party service, such as AdMob, needs to determine a characteristic about the user that it cannot derive from the built-in personae. For example, it may want to determine if the user is a student, in order to determine whether to display advertisements about spring break vacation packages. In this situation, AdMob can utilize a classifier developed in-house, which is installed as a user-level application on the user's device much as a third-party ad library is on existing mobile platforms.

This is a compelling example of securely-managed cooperation between MOREPRIV and a third-party that wishes to personalize, but it exposes an important concern for the third party. Namely, because AdMob's algorithm is running on the user's device in an unprotected mode, AdMob has no assurance that the results it obtains correspond to the true results of its algorithm. For example, a user who wishes to consume ad-supported content without providing information about themselves may simply install a "dummy" application in place of AdMob's, that returns random results.

In this section, we discuss a strategy for balancing this concern with our central focus of privacy based on recent advances in *Non-Interactive Zero-Knowledge (NIZK) Proofs of Knowledge* [7]. We provide integrity using the same practical assumptions as Danezis *et al.* [13] and Schnorr [41], which roots trust in the operating system, but does not require trusted hardware.

We discuss an implementation of the AdMob scenario given above that uses zero-knowledge proofs based on the construction of [39], and show how it can be efficiently parallelized to achieve acceptable performance. While our implementation is application-specific, the approach can be generalized without requiring custom zero-knowledge proof development using one of the many zero-knowledge proof compilers [3, 4, 33].

5.4 Achieving Computational Integrity

In this section, we begin by giving a brief general overview of NIZK schemes, and then describe our realization in the context of MOREPRIV.

Background and Overview: In a NIZK scheme, one party (the *prover*) attempts to convince another (the *verifier*) that it has correctly performed a computation that is known to both parties, without revealing the inputs that it provides to the computation. Oftentimes, the inputs are generated signed by a *data provider* that is trusted by the verifier, in order to establish an initial root of integrity. In our setting, the prover corresponds to a classifier, the verifier to a service that consumes the classifier's output, and the data provider to the MOREPRIV core.

Our NIZK scheme makes use of a non-interactive commitment scheme, which consists of the algorithms *Commit* and *Open*[5]. Given a value x, $Commit(x)$ produces a *commitment* c_x as well as a piece of auxiliary data o_x called the *opening*. The commitment is *opened* by revealing (c_x, o_x), and checking that $Open(x, o_x, c_x)$ returns *true*. Commitments have two useful properties: *(1) binding*, meaning that *Open* only returns *true* when $(c_x, o_x) = Commit(x)$, and *(2) hiding*, meaning that a party who possesses only c_x should not be able to learn anything about x. Abusing notation somewhat, we use the phrase *signed commitment*, and write $Commit_{K_A}(x)$, to refer to the commitment of x signed with A's public key. MOREPRIV provides inputs to the classifier via API calls, which can later be converted into signed commitments by the operating system when the consumer service requires a proof of integrity for some output of the classifier. The integrity of these inputs comes from the consumer service's trust in the integrity of the operating system, i.e., that it has not been compromised.

The scheme that we use requires the following entities:

- The computation $f(x_{\mathsf{pub}}, x_{\mathsf{priv}})$, which takes input known to both parties x_{pub} as well as input private to the prover x_{priv}.
- The inputs x_{pub} and x_{priv}, as well as signed commitments $Commit_{K_{TP}}(x_{\mathsf{priv}})$ to the private inputs, using the data provider's public key K_{TP}.

As illustrated in Figure 1, it proceeds as follows:

- The data provider sends the prover its private input x_{priv}, as well as the signed commitment $S = \mathrm{Commit}_{K_{TP}}(x_{\mathsf{priv}})$.
- The prover runs the computation on both sets of inputs to learn $r = f(x_{\mathsf{pub}}, x_{\mathsf{priv}})$. It constructs a zero-knowledge proof of knowledge $P = \mathrm{ZKProof}(f, x_{\mathsf{pub}}, x_{\mathsf{priv}}, r, S)$ which demonstrates that $\exists\, x_{\mathsf{priv}}.r = f(x_{\mathsf{pub}}, x_{\mathsf{priv}})$ and $S = \mathrm{Commit}_{K_{TP}}(x_{\mathsf{priv}})$. Finally, the prover sends P and r to the verifier.
- The verifier checks P against S and r, which can be done in roughly the same amount of time taken by the prover to generate P and compute r.

We refer the reader to the explanation of these schemes given by [7] for more details.

[5] For simplicity, we omit certain details relating to security parameters and setup. For a detailed explanation, the reader should refer to [7, 39]

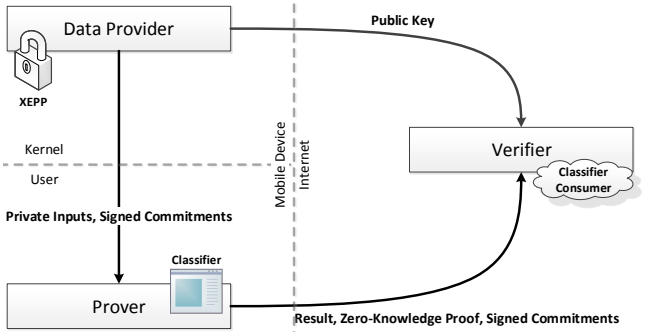


Figure 1: Zero-knowledge proofs for computational integrity.

5.5 Evaluation

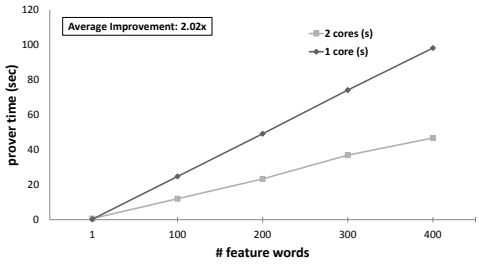
We evaluated our zero-knowledge classifier on a mobile device to determine its runtime, network utilization, and impact on battery life. In general, we found our implementation to be quite practical. Performance is parametrized by the number of feature words in the classifier.

For the student classifier in this section, based on our practical experience with Bayesian classification, we believe that using between 100 and 300 feature words gives sufficient precision. Note that we do not generally anticipate every single request requiring a proof; a scenario where a third party would periodically *audit* the result is more likely, making the numbers reported below a performance upper bound. Moreover, because the prover can run in idle time, when the phone is unused, and not on demand, we believe this represents more than acceptable performance.

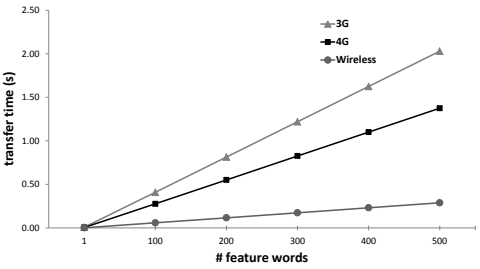
Setup: Unless stated otherwise, all experiments were performed on a Nokia Lumia 920 running Windows Phone 8. Experiments involving networking were performed on a small wireless network comprised of the mobile device, a laptop (with an Intel 4965AGN chipset), and a dedicated router not connected to an external network. The Zero-Knowledge protocol was implemented in F# and C#, using the .NET infinite-precision integer library. For commitments, we used a 1024-bit modulus; for signatures we used SHA1 with RSA, again with a 1024-bit modulus. All experiments used eight personae, unless otherwise noted.

Basic measurements: Figure 2(a) shows the time taken on the mobile device to compute the classifier result for all personae against the number of feature words in each persona's set of Bayesian classifier. Both cores of the mobile device were utilized without restriction. As shown, the time for a classifier size of 300 words is just under five minutes. Since our current personae classifiers are of approximately 300 words, we believe this to be a reasonable overhead. Profiling reveals that nearly all processing time is devoted to generating cryptographically-secure random noise (for commitment generation), and integer exponentiation (for the group operations). This means that while it is not feasible to produce a new, up-to-date zero-knowledge proof on-demand, one could perform these computations in the idle time after a period of user interaction with the phone.

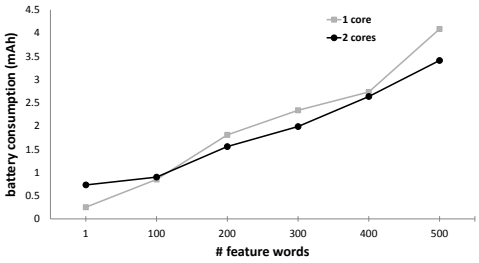
Figure 2(b) shows the network transfer time required to send the proof and input commitments against the number of feature words. To take these measurements, we simulated 3G and 4G cellular networks by modulating throughput, latency, and packet loss rate. Average throughput in each measured configuration (3G, 4G, and Wifi) was 134.4, 198.4, and 945.8 kilobytes per second, respectively. We observed proofs of less than 200 K that it takes 0.41–1.22 seconds to transmit the setup and proof for a 100 and 300-feature word classifier, respectively, on an "average" 3G network. We also



(a) Time to run the prover.



(b) Transmission time for the proof.



(c) Battery usage.

Figure 2: Custom classifier performance metrics

observed reasonably proof sizes of less than 200 KB for even the largest (300 feature-word classifier).

Figure 2(c) shows the amount of battery draw required by the classifier module, measured against the number of feature words in the classifier. This was computed based on CPU utilization, according to the model of Mittal *et al.* [34]. The power required to produce a proof goes down when multiple cores are available, and there is sufficient work available for the second core; this trend begins shortly after 100 feature words. To give these readings some perspective, we assume a 24-hour battery charge cycle in order to compute the amount of "battery time" consumed by the prover. Our test phone (Nokia Lumia 920) advertises a built-in 2,000 mAh battery, so producing a proof for a 300-word classifier consumes 0.099% of the available battery, about 40 seconds.

Summary: We believe our scheme to be practical with today's cryptographic primitives and mobile hardware in terms of computation and transmission time. With 100 feature words, the prover time is about 11 seconds. Transfer time of the proof for 100 feature words is under a second on an average 3G connection. Battery utilization (under 1 mAh) is negligible.

6. PERSONALIZATION

MoRePriv offers many opportunities for personalization. In this section, we focus on several personalization scenarios in depth, but note that this is only the tip of the iceberg. Other prominent examples include keying custom dictionaries for word completion,

News topic	activist	bachelor	business exec	sports buff	retiree	homemaker	technophile	tween
Health	3	3	4	4	10	7	2	1
Tech	4	6	5	5	3	4	10	8
US	9	4	7	6	6	4	3	2
Business	4	5	10	5	5	2	6	1
World	7	2	4	1	2	2	2	1
Entertainment	0	7	3	4	4	6	5	5
Science	2	3	3	1	3	2	6	4
Society	6	3	2	2	3	3	2	3
Politics	10	4	6	5	5	3	2	1
Sports	0	7	4	10	4	2	5	7

Figure 3: News personalization parameters.

spell-checking, and voice recognition on the persona.

6.1 Personalization Scenarios

RSS News Feed Personalization: To test the usefulness of the MOREPRIV APIs, we built a custom RSS reader called MoRSS. This app pulls stories from 10 RSS news feeds, and samples from these feeds to display a list of stories to the user. MoRSS disables the OS-level GUI enhancements described in Section 4. Instead, MoRSS relies on the built-in table in Figure 3 to rate how interesting each feed will be to each persona.

MoRSS can operate with no personalization, in which case stories from each RSS feed will be sampled uniformly and displayed to the user in the order in which they are sampled. When personalization is enabled, MoRSS queries the MOREPRIV API to determine the top persona of the user's profile, and then samples according to the column of Figure 3 for that persona.

MoRSS demonstrates the advantages of exposing limited information to third party applications: Developers have the flexibility to reinterpret the top profile in any way that they see fit. Apps such as MoRSS are free to sample tech stories for the homemaker, even though the built-in Bayesian classifier for that profile does not have tech keywords. Furthermore, the personalization can be done in a privacy-preserving way: MoRSS uses client-side personalization, so even the owner of the RSS feeds cannot learn the top profile of the user from the requests that MoRSS makes.

Application Skinning: OS-level personalization has broad applicability, even beyond networked device use. As a demonstration, we implemented a simple calculator that is re-skinned using the MOREPRIV API for each persona. For instance, when the top profile is retiree, the calculator goes into a high contrast mode with large text and buttons.

We highlight this example to show that MOREPRIV provides an alternative to forcing complicated configuration menus on users who nonetheless prefer different configurations. An app like the calculator could use profile data to provide an initial configuration that is likely to be close to what the user wants, and allow her to tweak configuration options from there.

6.2 Personalization Case Study: Yelp

The goal of the rest of the section is to demonstrate that persona-level personalization is useful and would be appreciated, were it provided by the operating system natively. However, effective personalization is not an easy task. It is both an art and a science, requiring a combination of technical and design skills, deep understanding of the target market, and the particulars of the application. Thus, we believe that a dedicated effort to create such a service would have even better results.

	Chicago	LA	Madison	NY	SF	Seattle
General	50%	33%	56%	50%	50%	39%
Executive	63%	76%	50%	53%	68%	47%
Boost	13%	43%	-6%	3%	18%	8%

Figure 4: Percentage of survey responders who prefer the executive personalization over the default one.

Personalizing Yelp: For our personalization target, we have chosen business recommendations using the Yelp API to get location-aware reviews and listings for businesses such as restaurants, hair salons, doctors, etc. popular in the US. The app includes a subset of the Yelp categories at `http://bit.ly/441TNj` relevant to a persona. When run, it pulls in a large number of Yelp listings, and prioritizes those in categories relevant to the top profile.

We have manually curated a list of 28 categories in the Yelp taxonomy (out of about 700) for business executive, intended to suggest an activity that an executive might bring business associates to. Categories fall into a more upscale niche, such as *steakhouses*, *wine bars*, or *auction houses*. To personalize, we sort the full set of Yelp results returned by reviewer rating "stars". We then search for a listing in a category relevant to the top persona not already selected for display. If there are no such listings, we select the highest ranked remaining listing in a category relevant to the top persona (even if we have already selected another listing in that category). We note that there are many additional features upon which to select listings, such as price, ambiance, availability of Wi-Fi, etc. Our experiment does not use these features, as doing so violates the Yelp API terms of use. Such restrictions could be ignored by an official Yelp app. In a general sense, we assume that the business executive persona applies at all times, which may not always be valid [12].

Methodology: Using the Instant.ly crowd-sourcing platform, we hired survey takers to answersix preference questions pertaining to the following cities: Chicago, LA, Madison, NY, SF, and Seattle. A sample survey can be seen at `http://www.instant.ly/s/B55pX`. One question was asked for every city, phrased as follows: *We are trying to understand which listing is more relevant for YOU. In other words, where are you more likely to go to and spend money at? Which listing below do you find more relevant and why?*

Of course, any such survey measures *expressed* preferences, not *actual* preferences, as noted in [36, 42]. However, short of deploying a modified app in the wild and watching users (who would have to opt into the study), this is a good initial assessment.

Selecting users: Using the Instant.ly built-in crowd, we created two survey groups *general* (21 respondents) and *executive* (41 respondents). Executives were selected based on the following criteria: age of 35–65, education of college graduate with a four-year degree or an advanced degree, employed full-time or self-employed, income in excess of $125,000, and married or living with a partner. This is a fairly simplistic way to reach that demographic, but we believe it suffices for our experiments. Given that these listings are somewhat location-specific, we tried to avoid location biases that might have been present in the recruiting process.

Results: Figure 4 presents the results of our study. We show preference for the executive treatment over the generic Yelp results for the general user group and the executive user group, reflecting the percentage of users who prefer the executive personalization. Overall, the *boost*, i.e. the difference in preference for executive in the executive demographics vs. the general population is most pronounced for Los Angeles (43%) and San Francisco (18%). The average boost across the cities is about 30%, which indicates that, on average, the executive personalization is quite effective.

7. RELATED WORK

Most efforts in the rapidly-growing space of mobile privacy research focus on detection and prevention techniques for information leaks, especially those in Android mobile apps [45, 26, 43, 15, 35]. TaintDroid [16] and Aquifer [35] are representative projects in this space, whose full overview we omit for space reasons. Our work takes a complementary position: leak detection tools prevent malicious use of private data, while MoRePriv offers devlopers a route to use private data in a way that does not damage the privacy of the user.

Resolving the tension between *extensible* personalization and privacy is not a new problem, which we attempt to address with our custom classifiers mechanism. Much work has been done on *verified extensions*, in contexts as diverse as OS drivers and browser extensions. The problem of checking extensions for privacy leaks can be addressed via static analysis [6], type systems [21], and runtime information flow tracking. This kind of validation can be integrated with MoRePriv or applied ahead of time by app store maintainers, but validating extensions is not the focus of our work. In contrast to many extensible systems, we do not require that custom classifiers be written in a special language, as, for example in [18].

MoRePriv falls into the category of efforts advocating client-side computing as a way to enhance user privacy. In the context of ad delivery, several researchers advocate remedying the problem by storing the necessary sensitive personal data on the client, along with all ads to be matched [23, 44, 27]. When an ad is displayed, it is matched to personal information locally, thus sidestepping the need to leak to the ad network. Accounting and click-fraud prevention are addressed using either additional semi-trusted parties, or homomorphic encryption. Although MoRePriv can be used in the context of personalized advertising, it has broader applicability than advertising alone.

RePriv [18] explored personalization opportunities in the context of a web browser by building a *user interest profile* based on the user's browsing history. RePriv profiles classify the sites the user visits based on the Open Directory Project (ODP) taxonomy of interests. Unlike RePriv, we focus on the mobile space. We go further by integrating our system directly into the operating system, drawing from more diverse data sources, and provide integrity through zero-knowledge proofs.

Both MoRePriv and RePriv are representative of an expanding line of work focusing on client-side computing for privacy. There are a number of notable differences between RePriv and MoRePriv: 1) MoRePriv user data is very coarse-grained, and its disclosure to apps is limited. This persona-based approach is substantially different than the taxonomy-based approach of RePriv; 2) MoRePriv does **not** impose a development language and an onerous-to-use type system on the developer; 3) MoRePriv alone provides automatic personalization for legacy apps. The focus on integrity and the use of Zero-Knowledge proof techniques is specific to MoRePriv as well; 4) Since MoRePriv is at the OS level, it has access to different, rich sources or data - personalization approaches for the browser and OS are substantially different.

Zero-knowledge proofs [5, 7, 13] have seen extensive use in the privacy and applied cryptography literature. Zero-Knowledge protocols have been developed for proving linear relations [9], equality and inequality [41], logical connectives [9], multiplication [10], division and modulo [11], and set membership [10]. Taken together, this work allows the construction of a zero-knowledge proof that expresses the functionality of an arbitrary circuit, as in the case of fully-homomorphic encryption [19].

8. CONCLUSIONS

This paper proposes operating system-level mechanisms that support automatic personalization of legacy applications, and simplify building new personalized applications. Our focus is primarily on mobile operating systems. Implementing a tool at this level allows us leverage a broad variety of user activity, and distill it into a trusted source of personalization information. We believe that at the level of the operating system these opportunities are largely untapped. We demonstrate that personalization can be done quickly and effectively using personae, which provide a degree of pseudonymity and are easy for users and developers to understand.

We show how both OS-wide universal personalization and custom personalization can be done with little effort by the developer, making us hope that persona and location information can become equally ubiquitous on mobile devices. We also demonstrate and experimentally evaluate how MoRePriv supports *integrity* of custom classifiers with the help of zero-knowledge proofs. With a classifier using 100 feature words, the prover time is about 11 seconds, the time to transfer the proof is under a second on an average 3G connection, and the battery utilization is negligible — under 1 mAh. Our results show that trusted classification is feasible with today's zero-knowledge techniques.

Finally, we performed a crowd-sourced user study showing that MoRePriv-style personalization improves the relevance of Yelp results, with business executives preferring MoRePriv personalized results most of the time (a boost of ~30%). In our experiments we have observed cases where the general population and users with a particular persona disagree on what is most relevant, highlighting the need for user profiling and personalization.

We believe MoRePriv is timely and can enable game-changing innovation around privacy through client-side computing.

9. REFERENCES

[1] Application privacy. `http://www.applicationprivacy.org/?page_id=39`.

[2] Mobile app privacy policies are now the law. `http://www.truste.com/blog/2012/03/02/mobile-app-privacy-policies-are-now-the-law`.

[3] J. B. Almeida, E. Bangerter, M. Barbosa, S. Krenn, A.-R. Sadeghi, and T. Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on σ-protocols. In *Proceedings of the European Conference on Research in Computer Security*, 2010.

[4] M. Backes, M. Maffei, and K. Pecina. Automated synthesis of privacy-preserving distributed applications. In *Proceedings of the Network and Distributed System Security Symposium*, 2012.

[5] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens. Pretp: privacy-preserving electronic toll pricing. In *Proceedings of the Usenix Conference on Security*, 2010.

[6] T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, and A. Ustuner. Thorough static analysis of device drivers. In *Proceedings of the European Conference on Computer Systems*, 2006.

[7] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *Proceedings of the International Cryptology Conference on Advances in Cryptology*, 1993.

[8] R. Bhaskar, S. Guha, S. Laxman, and P. Naldurg. Verito: A Practical System for Transparency and Accountability in Virtual Economies. In *Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb 2013.

[9] S. Brands. Rapid demonstration of linear relations connected by boolean operators. In *Proceedings of the International Conference on Theory and Application of Cryptographic*

Techniques, 1997.

[10] J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, 2008.

[11] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, 1999.

[12] L. F. Cranor. Designing personalized user experiences in ecommerce. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, 2004.

[13] G. Danezis, M. Kohlweiss, B. Livshits, and A. Rial. Private client-side profiling with random forests and hidden Markov models. In *Proceedings of the International Conference on Privacy Enhancing Technologies*, 2012.

[14] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *Proceedings of the Network and Distributed System Security Symposium*, Feb. 2011.

[15] S. Egelman, A. P. Felt, and D. Wagner. Choice architecture and smartphone privacy: There's a price for that. In *WEIS*, 2012.

[16] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the Usenix Conference on Operating Systems Design and Implementation*, 2010.

[17] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of SOUPS*, 2012.

[18] M. Fredrikson and B. Livshits. RePriv: Re-envisioning in-browser privacy. In *IEEE Symposium on Security and Privacy*, May 2011.

[19] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the ACM Symposium on Theory of computing*, 2009.

[20] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Conference on Security and Privacy in Wireless and Mobile Networks*, Apr. 2012.

[21] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. Verified security for browser extensions. In *IEEE Symposium on Security and Privacy*, May 2011.

[22] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *Proceedings of the Usenix Conference on Networked systems design and implementation*, 2011.

[23] S. Guha, A. Reznichenko, K. Tang, H. Haddadi, and P. Francis. Serving Ads from localhost for Performance, Privacy, and Profit. In *Proceedings of Hot Topics in Networking*, Nov. 2009.

[24] S. Han, J. Jung, and D. Wetherall. A study of third-party tracking by mobile apps in the wild. Technical report, University of Washington, Mar. 2012.

[25] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In *Proceedings of the International Symposium on Information, Computer, and Communications Security*, 2011.

[26] Y. Z. X. Jiang. Detecting passive content leaks and pollution in android applications. In *Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)*, Feb 2013.

[27] A. Juels. Targeted advertising ... and privacy too. In *Proceedings of the Conference on Topics in Cryptology*, Apr. 2001.

[28] A. Kobsa. Privacy-enhanced personalization. *Commununications of the ACM*, 50(8), Aug. 2007.

[29] A. Kobsa, B. Knijnenburg, and B. Livshits. Let's do it at my place? attitudinal and behavioral study of privacy in client-side personalization. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems Proceedings*, Apr. 2014.

[30] A. Kobsa and J. Schreck. Privacy through pseudonymity in user-adaptive systems. *ACM Transactions Internet Technologies*, 3(2), May 2003.

[31] B. Livshits and J. Jung. Automatic mediation of privacy-sensitive resource access in smartphone applications. In *Proceedings of the Usenix Conference on Security*, 2013.

[32] J. R. Mayer and J. C. Mitchell. Third-party Web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy*, May 2012.

[33] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. Zkpdl: a language-based system for efficient zero-knowledge proofs and electronic cash. In *Proceedings of the Usenix Conference on Security*, 2010.

[34] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the International Conference on Mobile Computing and Networking*, 2012.

[35] A. Nadkarni and W. Enck. Preventing accidental data disclosure in modern operating systems. In *ACM Conference on Computer and Communications Security*, pages 1029–1042, 2013.

[36] P. Norberg, D. Horne, and D. Horne. The privacy paradox: Personal information disclosure intentions versus behaviors. *Journal of Consumer Affairs*, 41(1), 2007.

[37] L. Pareschi, D. Riboni, A. Agostini, and C. Bettini. C.: Composition and generalization of context data for privacy preservation. In *In: Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), Proceedings of the Workshops, IEEE Computer Society*, 2008.

[38] A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: An information theoretic approach. *SIGKDD Explor. Newsl.*, 10(2):90–100, Dec. 2008.

[39] A. Rial and G. Danezis. Privacy-preserving smart metering. In *Proceedings of the Workshop on Privacy in the Electronic Society*, 2011.

[40] E. Rich. User modeling via stereotypes. *Cognitive Science*, 3, 1979.

[41] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4, 1991.

[42] S. Spiekermann, J. Grossklags, and B. Berendt. E-privacy in 2nd generation e-commerce: privacy preferences versus actual behavior. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, EC '01, 2001.

[43] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries. In *MoST 2012: Mobile Security Techologies (2012)*, 2009.

[44] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the Network and Distributed System Security Symposium*, Feb. 2010.

[45] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: characterizing mobile advertising. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, 2012.

[46] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. Taming information-stealing smartphone applications (on android). In *Proceedings of the 4th international conference on Trust and trustworthy computing*, TRUST'11, pages 93–107, Berlin, Heidelberg, 2011. Springer-Verlag.