Design, Realization, and Evaluation of xShare for Impromptu Sharing of Mobile Phones

Yunxin Liu, *Member*, *IEEE*, Ahmad Rahmati, *Student Member*, *IEEE*, Hyukjae Jang, Yuanhe Huang, Lin Zhong, *Member*, *IEEE*, Yongguang Zhang, *Member*, *IEEE*, and Shensheng Zhang

Abstract—Mobile phones are truly personal devices loaded with personal data such as photos, contacts, and call history. Yet it is often necessary or desirable to share our phones with others. This is especially true as mobile phones are integrating features conventionally provided by other dedicated devices, from MP3 players to games consoles. Yet existing phones assume a single user and provide little protection for private data and applications when a phone is shared. That is, when we lend our phones to others, we give away complete access. In this work, we present xShare, a protection solution to address this problem. xShare allows phone owners to rapidly specify what they want to share and place the phone into a restricted mode where only the data and applications intended for sharing can be accessed. We first present two formative user studies and derive the design requirements of xShare. We then offer the design of xShare based on file-level access control. We describe the implementation of xShare on Windows Mobile and report a comprehensive evaluation, including performance measurements, usability, and a one-month field trial.

Index Terms—Mobile phone, sharing, privacy, virtualization.

1 Introduction

Existing mobile phones provide inadequate support for sharing; there is no access control on private data and pay-per-use applications. Consequently, when the owner shares their phone, the borrower will have the same access rights as the owner. Some mobile phones use a password to prevent unauthorized access; yet it is for the entire system, and therefore, the access control is *nothing or everything*. The iPhone has a *restriction* feature that can disable some built-in applications. Yet it does not apply to third-party applications nor does it provide access control for data. Windows Mobile phones can boot into a less-known *Kiosk* mode, in which only certain applications can be run. However, it requires a reboot and does not provide access control to data.

- *Y. Liu is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 20030, and with Microsoft Research Asia, 4F Beijing Sigma Center, Zhichun RD, Haidian DC, Beijing 100190, P.R. China. E-mail: yunxin_liu@sjtu.edu.cn, yunliu@microsoft.com.*
- *A. Rahmati and L. Zhong are with the Department of Electrical and Computer Engineering, Rice University, 6100 Main St., MS-380, Houston, TX 77005. E-mail: {rahmati, lzhong}@rice.edu.*
- *H. Jang is with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 373-1, Guseong-dong, Yuseong-gu, Daejeon, Republic of Korea. E-mail: hjjang@nclab.kaist.ac.kr.*
- *Y. Huang is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100080, P.R. China. E-mail: thuhyh@gmail.com.*
- *Y. Zhang is with Microsoft Research Asia, 4F Beijing Sigma Center, Zhichun RD, Haidian DC, Beijing 100190, P.R. China. E-mail: ygz@microsoft.com.*
- *S. Zhang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 20030, P.R. China. E-mail: sszhang@sjtu.edu.cn.*

Manuscript received 19 Oct. 2009; revised 23 May 2010; accepted 12 Aug. 2010; published online 27 Aug. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org and reference IEEECS Log Number TMCSI-2009-10-0435. Digital Object Identifier no. 10.1109/TMC.2010.162.

To address such limitations, we present the results from a complete research and development cycle of xShare, a software solution for friendly, efficient, and secure phone sharing. xShare allows the owner to rapidly specify what they want to share and place the phone into a restricted mode where only specifically shared applications and data are accessible.

First, we present a thorough understanding of phone sharing obtained from two user studies, including interviews with existing smartphone users from four countries and long-term user studies with teenagers from the USA. Our user studies show that the majority of existing and potential users share their mobile phones. Our user studies provide insight into why, where, with whom, and for what applications mobile users share their phones.

Based on these findings, we propose the design of xShare based on file-level access control. xShare provides two modes of operation: Normal Mode and Shared Mode. When switching from Normal Mode to Shared Mode, the owner specifies which files and applications to share, or a *sharing policy*. xShare creates a virtual environment for Shared Mode from the sharing policy. The virtual environment contains only specifically shared files and applications and conceals the rest of the system. The borrower uses the phone in Shared Mode.

Furthermore, we report an implementation of xShare on Windows Mobile. Our implementation works atop of the existing systems without requiring changes to the OS source code or the phone ROM image. As Windows Mobile lacks built-in file-level access control, we implement one based on system API interception at the kernel level. We implement namespace virtualization for resource access and create a virtual environment to contain shared data and applications in Shared Mode. Through careful examination of the Windows CE kernel, we are able to address several practical challenges as well.

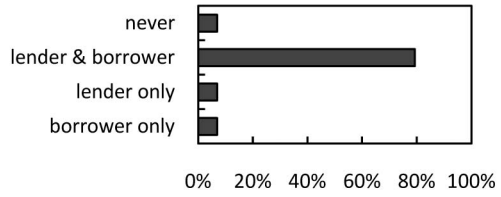


Fig. 1. Phone sharing statistics.

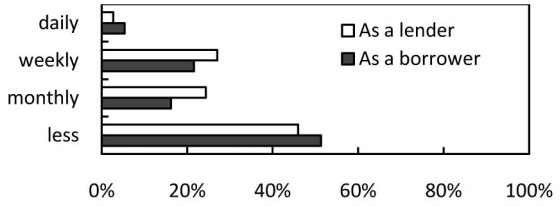


Fig. 2. Frequency of phone sharing among our sharing-type participants, as a borrower (top) and lender (bottom).

Finally, we provide a comprehensive evaluation of the xShare implementation through performance measurements, two user studies of usability, and a one-month field trial. Our measurements show that xShare barely affects the overall system performance in Shared Mode. Our first user study shows that phone owners' subjective opinions were extremely positive. Our second user study shows that xShare is resilient to attempts of unauthorized access. Our field trial shows that xShare is able to meet the needs of participants when sharing their mobile phones.

It is important to note that xShare is not intended to make mobile phones more secure than they already are. Instead, it is intended to limit temporary users to explicitly shared services and data. Therefore, even with xShare, temporary users may be able to exploit existing security flaws in the phone to overpower xShare.

The rest of the paper is organized as follows: We present the motivational user studies in Section 2. We provide the design, implementation, and evaluation of xShare in Sections 3-5, respectively. We address the limitations of xShare in Section 6, discuss related work in Section 7, and conclude in Section 8.

2 Understanding Phone Sharing

We next present findings from two user studies regarding mobile phone sharing. The first consists of interviews in four countries; the second is a four-month field trial with 14 teenagers in the USA. The findings motivate the need for privacy protection during phone sharing and provide the design requirements for xShare.

2.1 International Interview

To learn about the current global status of phone sharing, we designed and conducted an interview of about 35 questions, of which several are open-ended. The interviews were structured and lasted between 30 and 45 minutes. Participants were required to have phones that at a minimum have a built-in camera and support video and music playback. They were recruited through the authors' social networks, e.g., by advertising the study through friends and coworkers. We recruited 60 participants in total from China, South Korea, Iran, and the USA, with a minimum of 10 from each country. All but one were aged between 18 and 40 years old. While we recruited the participants to provide a broad as possible sense of how phone users share their phones, we do not claim that our participants were a balanced or accurate representation of any demography.

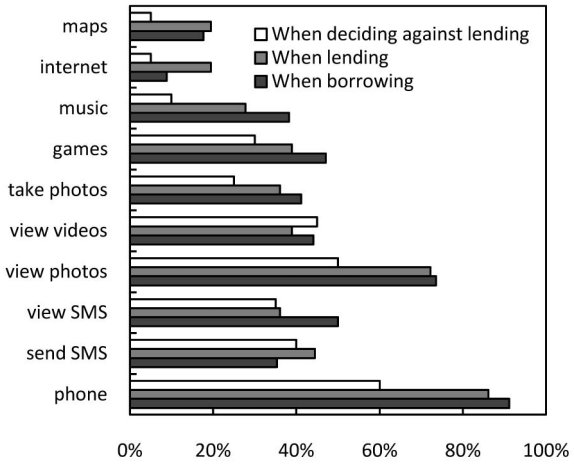


Fig. 3. Applications involved in phone sharing.

2.1.1 Nature of Sharing

Our international interviews provide evidence that phone sharing is very popular and involves a wide range of applications, reasons, social settings, and relationships. Ninety-three percent of our interviewees have participated in phone sharing, either as a lender or borrower, as shown in Fig. 1. Twenty-two percent of our participants only shared their phones when a borrower needed to make a phone call but did not have their phone at hand, or it was out of battery. We call them the *nonsharing type*. On the other hand, 64 percent of our participants lent and borrowed phones for other reasons as well, e.g., access to certain applications or data. We call them the *sharing type*. The nonsharing and sharing types had a significantly different frequency of sharing. All nonsharing type participants reported that they participate in sharing less than once a month. In contrast, more than half of the sharing type participants reported sharing their phones at least monthly, as shown in Fig. 2. In the rest of this section, we will focus on and report *only* the sharing type, and "participants" refers to sharing type participants. The significant number of sharing type users highlights the potential for a privacy-maintaining phone sharing solution.

What applications. The most common applications shared are shown in Fig. 3. As expected, placing phone calls was very popular. However, a wide range of other applications were also popular, and viewing photos was almost as popular as phone calls. We must note that many participants didn't have internet access on their phones, and data plans are very costly in some of the studied markets, therefore explaining the low usage. The range of applications that are shared indicates that a solution for phone sharing must be scalable and support a wide range of applications.

With whom. Our participants shared phones with people from a wide range of social relationships, as shown in Fig. 4. As the same data may be considered as confidential or

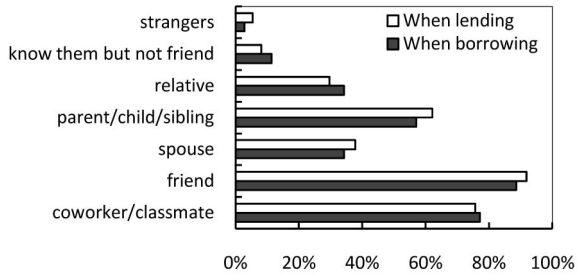


Fig. 4. Relationships of those involved in phone sharing.

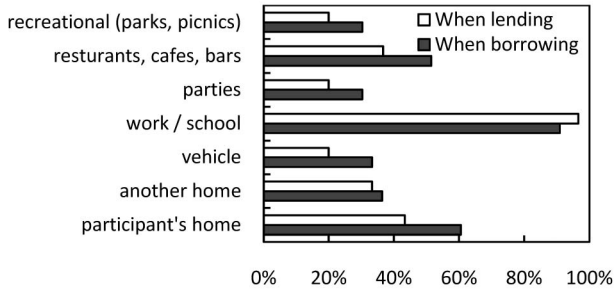


Fig. 5. Locations of phone sharing.

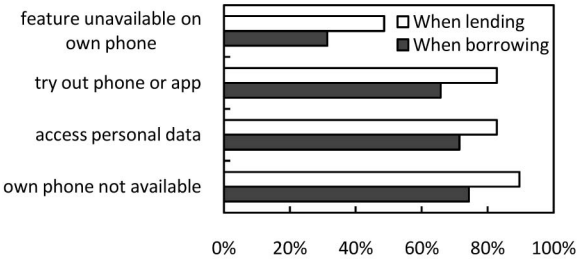


Fig. 6. Reasons for sharing phones.

sharable depending on the relationship between the parties, a solution for phone sharing must provide customizable access to phone functionalities and data.

Where. The most popular location for phone sharing was at school or in the workplace. However, sharing occurred at a variety of locations, both private and public, as shown in Fig. 5. Therefore, a solution for phone sharing must be configurable in an impromptu manner and any location. This also reinforces the need for a customizable solution, since the same data may be considered as confidential or sharable depending on the location.

Why. The most common reasons our participants shared their phones and used other people's phones are shown in Fig. 6. We can see that it is very common to use a shared phone when the borrower's own phone is not at hand or is out of battery. However, accessing the owner's personal data is also a very common purpose of phone sharing. Therefore, a solution for phone sharing must provide privacy for the owner-created data. Showing or trying out a feature or a different phone and accessing features unavailable on a user's own phone were also popular. Therefore, a solution for phone sharing must be able to support new applications installed by the owner.

Who is the initiator. When a borrower uses the lenders mobile phone, either the borrower or the lender may have

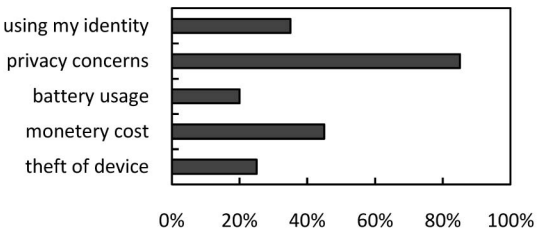


Fig. 7. Reasons for deciding against sharing.

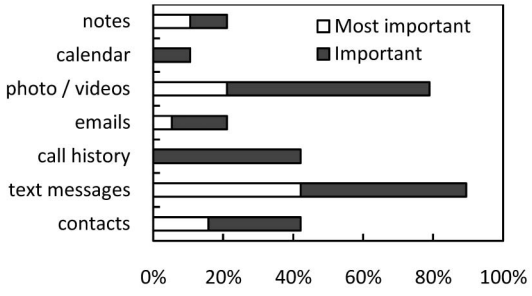


Fig. 8. Private information on owners' phones that prevents them from sharing their phones.

initiated the use. That is, the borrower may ask the lender to use their phone; or the lender may ask the borrower to use their phone. In the former case, it is crucial that a solution for phone sharing can be configured quickly to avoid embarrassment. While all our participants reported being asked to share their phones, 64 percent of our participants reported initiating the sharing of their phones as well. This shows that many participants used sharing for social purposes by offering their phones to others.

2.1.2 Privacy Concerns

Our interviews highlight that privacy remains a major concern for phone sharing: the most common reason mentioned by our participants for not sharing their phone was concerns regarding the third party's access to their private data. This was followed by concerns regarding the borrower assuming their identity, e.g., caller ID, introducing too much monetary cost, and using too much battery, as shown in Fig. 7.

Classified user data. We asked participants to select what types of information on their phones prevented them from sharing their phones, and which one was the most important. Photos and videos, text messages (SMSes), and contacts were mentioned by most participants, along with a wide range of other items, as shown in Fig. 8. The diversity in what types of personal information users consider private and most private confirms that we would need to support fine-grain access control for a wide range of data types in order to provide meaningful privacy to all users.

Existing protection inadequate. The built-in access control for most of our participants' phones was very simple: a password or PIN code for accessing the entire phone. A few of the phones had the ability to password protect certain functions, such as access to the user storage or to specific applications. However, all but one of participant told us that they rarely or never use it, and some told us that in order for borrowers to access the shared

data, they would have to disable the password protection anyway, so it is not useful.

How owners deal with concerns. Lacking useful privacy protection on phones, we found that our participants dealt with their concerns regarding phone sharing in three distinct ways. First, they simply share their phone less often. Fifty-six percent of them recalled instances where they refrained from sharing due to privacy concerns, and all but three of them told us that they would be more willing to share their phone if it supported better privacy protection. Second, the owners casually supervise the phone when sharing: 86 percent of our participants told us that they usually or always keep their phone in sight when sharing, and 60 percent told us that they usually or always keep an eye on what the borrower is doing on their phone. This enables the enforcement of mutually agreed privacy boundaries, but places an extra physical and mental burden on the lender and may make the borrower feel uncomfortable or mistrusted. Third, the owners prepared their phones before handing it to others. Fifty-four percent of our participants reported preparing their phone by moving or deleting private data, enabling or disabling certain features, and/or bringing the application being shared to the front. Of those, 55 percent reported that such preparation takes more than 30 seconds and 75 percent thought that it takes too much time. This highlights the need for minimum latency when entering Shared Mode.

2.2 Long-Term Field Study

While our interviews examined sharing behavior and concerns of existing phone owners, they provide little insight into the underlying reasons they share phones and how they have reached this level of sharing. We have recently concluded a four-month field trial of a Windows Mobile phone in Pecan Park, a low-income urban community in Houston, Texas. In the field trial, we distributed HTC Wizard phones to 14 high school students from the community, recruited through a local nonprofit organization. We conducted focus groups with the participants regularly throughout the study, on average, every three weeks. The focus groups were recorded using a voice recorder, and later transcribed, coded, and analyzed manually.

We made the following observations regarding their sharing behavior. First, initially, all participants actively shared the mobile phones as a social networking tool, at various locations, including home and school, and with people of different relationships with them, from family members to newly introduced peers. Second, sharing decreased as the study went on. This was in part due to the diminishing excitement of having a new phone; and more importantly, the increasing privacy concern as the phones became personalized with privacy-sensitive data such as contacts, photos, and SMSes. Their sharing circle eventually became very small, limited to very close friends or siblings. Third, sharing is mostly impromptu, taking place at time and location of convenience and social significance, and more importantly, without prior planning. This is because sharing is driven by a social purpose and mobile phones can be conveniently used at most locations and times. Fourth, sharing is usually application-driven and data-driven. That is, as observed in our international interviews, the phone owners lend their phone to others for accessing specific data with a specific application, e.g., playing a song with Media Player and viewing photos under a certain directory. Fifth, the sharing policy is dynamically determined, mutually agreed upon implicitly, and often enforced casually. The lender often keeps a watchful eye to ensure that the borrower does not go beyond the mutually agreed boundaries, similar to what we found out from the first user study.

2.3 Threat Model and Design Requirements

Our two user studies highlight the need for a solution to the privacy challenge of sharing phones. Further, they show that sharing serves social purposes. In order to retain the value of sharing, a privacy solution must support impromptu configuration for a temporary guest user with minimum latency. This was further confirmed by our participants' opinion on preparing the phone to maintain privacy while sharing (e.g., by deleting/moving private data). Seventy-five percent who had prepared the phone in this way told us that it takes too long.

From our user studies, it is apparent that the user often knows the borrower and lends the phone under a socially defined agreement as to the limits of the borrowers' access. Therefore, the threat in phone sharing is from a curious or careless borrower who may snoop into or accidentally get exposed to personal data. Further, the borrower may inadvertently consume excessive exhaustible or billable resources, such as battery and cellular minutes. Therefore, we aim at limiting the data and services accessible to the borrower by providing in situ configurable access control and resource accounting. We base our threat model on an attacker centric approach, considering the careless or curious borrower. It is important to note that our goal is not to make the system more resilient against existing security flaws that may be exploited by intentionally malicious borrowers, as well as worms and viruses.

Based on our user studies and the threat model described above, we distill the following design requirements for supporting phone sharing through xShare:

- **Impromptu policy creation:** xShare should allow the owner to specify a policy for what to share with minimum latency. This is more important than the latency for exiting Shared Mode, as it can impact the social value of sharing.
- **Access control:** xShare should provide access control for three different categories: individual applications, data files and folders, and system resources such as storage, battery, minutes, and cellular data usage.
- **Resource accounting:** To control exhaustible system resources and pay-by-use services, xShare should provide accounting for them and track their usage.
- **Borrower data reconciliation:** In many instances, the borrower may create or modify data files and system settings. For example, 65 percent of our participants shared their phone for a borrower to try out, and 36 percent shared their phone for the borrower to take photos. Therefore, xShare must track borrower changes and allow the owner to accept or reject them.

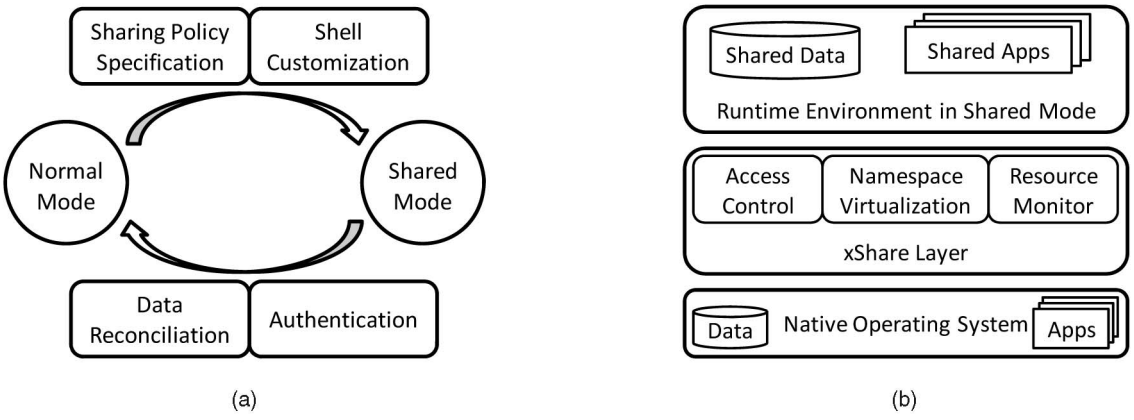


Fig. 9. Design overview of xShare. (a) Mode transition. (b) Architecture of shared mode.

In addition, xShare must be resource-efficient because mobile phones are highly resource-constrained. Further, the borrower's experience should be comparable to the owner's.

3 Designing xShare

Based on the findings from the user studies, we present our platform-independent design of xShare and provide rationales for each design decision.

3.1 Design Overview

xShare runs atop of the OS. It has two modes: Normal and Shared. In Normal Mode, it appears to be a regular application that can be launched and closed, and does not affect the operation of other applications. In Shared Mode, it runs in the background, customizes the system shell, and enforces the sharing policy based on the owner's specification. Fig. 9 provides an overview of xShare, in particular, transitions between the two modes and the architecture of Shared Mode. When switching from Normal Mode to Shared Mode, xShare provides a UI for the owner to specify the *sharing policy*: what files and applications to share. xShare then creates a virtual environment and runs in the background enforcing the policy. When switching from Shared Mode to Normal Mode, user authentication is required. After that, xShare provides another UI for the owner to either accept or reject changes made in Shared Mode. We call this *borrower data reconciliation*.

File-based access control. A key design decision we made is to base xShare's access control at the file level. The rationale behind this design decision is for generality. First, all major mobile operating systems, including Symbian, Linux, Windows Mobile, iPhone OS, Blackberry, and Palm, support files as an abstraction for both data and applications, implement file systems, and provide APIs for file operations. Second, most applications on major mobile OSes leverage files as a level of data abstraction, and our user studies show that accessing personal data is one of the popular reasons for phone sharing (Fig. 6). Therefore, a file-based access control allows an application-independent solution. While finer access controls are possible and may be desirable, e.g., access control to individual entries in the SMS history, they will require application-specific implementation.

Many popular smartphone OSes, including Symbian and Windows Mobile, do not support file access control. Therefore, xShare has to provide file access control, which may be platform-dependent. In Section 4.1, we will present the case for Windows Mobile. Linux and iPhone OS provide Unix-style file access control with which read, write, and execution rights can be granted to a user. Even with such file-level access control, there are multiple practical challenges to realizing access control for xShare, as will be addressed in Section 3.3.

3.2 User Interfaces

xShare provides multiple UIs for the owner. It also customizes the shell environment for the borrower.

3.2.1 Policy Specification

xShare provides a UI for owners to rapidly specify a sharing policy. A sharing policy consists of three components: shared files selection, shared applications selection, and resource allowance specification (Fig. 11). We utilize a hierarchical list-based selection that lists all possible choices. For file sharing, we employ a hierarchical list similar to the built-in file browser to enable the owner to explore and select individual folders and files. Each item on the list has an indicator that indicates the sharing status of the file, folder, or application. Clicking on the indicator changes the sharing status. Below are several design considerations for quick policy specification.

- Because the OS associates a default application to most file types, xShare asks the owner to select the shared files before applications and automatically selects appropriate applications for the selected files.
- All items start as not shared, except those applications that are automatically selected based on the shared data file types.
- The owner specifies whether a file or application is shared or not. Only shared applications can be run, and xShare gives them read and write access to shared files, as well as permission to create new files.

- xShare employs profiles to enable the owner to save frequently used sharing policies.
- xShare provides a quick launch method, called Quick Share. With Quick Share, xShare enters Shared Mode only allowing access to the current front application and the files currently opened by it; thus, the owner doesn't need to specify the sharing policy. This is motivated by our user studies; 76 percent of our Sharing Type interviewees reported that they rarely or never shared multiple applications in one sharing session.

3.2.2 Shell Customization in Shared Mode

The shell UI of a phone often shows some data items such as calendar events and contains icons or links to launch applications. In Shared Mode, these items are hidden from the borrower by concealing the nonshared data items and removing the icons and links of the nonshared applications. The borrower can still access shared data applications from the shell. Therefore, a consistent user experience is maintained in Shared Mode.

3.2.3 Borrower Data Reconciliation

When exiting from Shared Mode, xShare prompts the owner to manage changes made to files and settings in Shared Mode. It displays the changes and their location in a UI similar to the policy specification UI. The owner can reject or accept the changes. The default choice for modified items is *reject*; the default for new files is *accept*. This is based on our user study findings that show that most file sharing is intended for read only access, e.g., music and pictures. In contrast, many shared applications are intended for the borrower to create files, e.g., the camera.

3.3 Access Control

The key component of xShare is impromptu configurable access control to applications and data. Once the owner specifies the sharing policy, xShare must rapidly identify the files it should grant access in order to block access to nonshared applications and data. In runtime, xShare checks the list of the files to determine whether a given file is in the list and if access to it should be granted. We next present our design decisions for two important practical challenges.

3.3.1 In-Memory Services and Applications

Mobile OSes can keep some key services and recently used applications in memory to expedite their launch. Such services and applications create a challenge to access control when shared, due to the private data they may have loaded. For example, on Windows Mobile, all SMSes are stored in the file "cemail.vol," which is kept open once the OS is booted. As a result, if the owner allows the borrower to use the SMS application, they may allow the access to the SMS history. Most sharable applications and services can be terminated or restarted without any impact on the rest of the system. Therefore, xShare simply terminates their corresponding processes before entering Shared Mode. Further system and application cooperation may eliminate the need of terminating and restarting specific applications. Yet it requires modifying existing applications.

Certain applications and system services are so tightly integrated with the system that they cannot be terminated properly. Addressing this issue is highly platform and application-dependent. We will describe our solutions when we present our Windows-Mobile-based implementation in Section 4.

3.3.2 Identifying Files for Application Sharing

Many applications require more than their executable files to run, e.g., configuration files and DLLs. xShare must locate an adequate set of files for the application to run properly. It is generally difficult to distinguish between files necessary for an application to run and private data files opened by the user in that application. As most mobile OSes, including Windows Mobile, Symbian, and iPhone OS, store application files, and data files in different folders, xShare simply allows access to all the files in the same folder as the corresponding executable file. Some applications may access nonstorage peripheral devices, such as the camera and the microphone. These nonstorage peripherals do not contain any private information; therefore, xShare grants access to them in Shared Mode.

3.4 Virtual Environment

Given the sharing policy, xShare creates a virtual environment for Shared Mode. The virtual environment confines access to the shared data and applications through namespace virtualization based on file-level access control. In the virtual environment, xShare tracks all changes made by permitted applications and allows the owner to manage them when exiting Shared Mode.

3.4.1 Namespace Virtualization

Namespace virtualization is the natural choice for sandboxing the shared applications and data to implement access control and a virtual environment for Shared Mode. Namespace virtualization controls resource access by renaming those resources. By concealing unshared files, redirecting write access, and maintaining consistency, namespace virtualization provides a view of the system resources that is consistent with the sharing policy.

3.4.2 Change Separation

xShare separates changes made to files and settings in Shared Mode and ensures that those changes cannot affect the system in Normal Mode. Recall that xShare only asks the owner to decide which files or directories to share. Once a file is shared, xShare allows both read and write access to it in Shared Mode. To protect the shared files from unwanted modification, xShare creates a private folder to hold all the modified and created files in Shared Mode. Instead of changing the original files, xShare employs *copy-on-write* to redirect all the changes to the private folder. We will provide a detailed explanation in Section 4.2.

3.4.3 Hiding Nonshared Files

Through namespace virtualization, xShare hides nonshared resources from shared applications in Shared Mode. For example, if we share two of the five files in a folder, when an application lists the folder, it will only see the two shared files.

3.4.4 Resource Accounting

As indicated by our user studies, phone owners are concerned with the overusage of exhaustible system

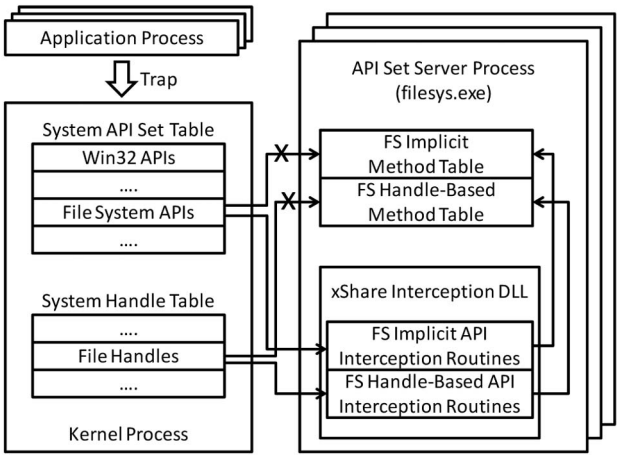


Fig. 10. System API interception on Windows Mobile.

resources, such as battery and storage, and network chargeable features, such as phone minutes, SMS, and data counts. Leveraging APIs provided by existing mobile operating systems, xShare enables the owner to set restrictions on the amount of resources used in Shared Mode, e.g., 5 MB storage or stopping sharing when the battery drops to as 20 percent.

4 Implementation

Windows Mobile provides rich support for development, especially for system-level programming, but presents two unique challenges for xShare implementation. First, it provides no file-level access control, unlike its desktop counterpart, the iPhone OS, and Linux. Second, Windows Mobile tightly integrates system services and critical applications; as a result, changes to one are likely to impact others. We next present our solutions in addressing these challenges. While we focus on the Windows Mobile implementation, some components, such as namespace virtualization and the UI, are common for both Windows Mobile and other mobile OSes, such as the iPhone OS.

4.1 API Interception for File Access Control

Similar to most mobile OSes, Windows Mobile provides no native support for strict file-level access control. Any application can use *CreateFile*() to open any file. We implement file-level access control for Windows Mobile by intercepting system APIs at the kernel level. API interception is a natural choice for file-level access control since all applications use system APIs to access files. While access control can be implemented at either the user level or the kernel level, user-level implementation requires changing the phone's ROM image. This is because Windows Mobile extensively employs eXecution-In-Place (XIP), i.e., executing native programs and DLLs directly from ROM without copying them into RAM, and therefore, we cannot change the import address tables of programs or the export address tables of DLLs [5] without rebuilding the ROM image.

4.1.1 Implicit System APIs

Windows Mobile uses a client/server model for APIs and implements all system APIs in separated server processes. Its system APIs are either implicit or handle-based. Implicit APIs are globally registered and dispatched through the system API table, e.g., *CreateFile*(). Handle-based APIs are attached to a kernel object such as a file or an event and called through a handle to the kernel object, e.g., *WriteFile*(). As shown in Fig. 10, when an application calls a system API, it causes a trap and jumps into the kernel. The kernel then searches the system API set table for the address of the method implementing the API and calls the method.

xShare intercepts implicit system APIs by locating the system API set table and manipulating its entries. We implement our interception routines in the xShare interception DLL, load it into the address space of an API set server process, and direct the corresponding entry in the system API set table to an interception routine. Consequently, the corresponding system API calls will be directed to xShare interception routines for access control. The interception DLL also holds the pointer to the original table in the server process so that the interception routines can call the original system API methods if the access control checking is passed.

4.1.2 Handle-Based System APIs

As all handles are created in implicit APIs, we track their creation in corresponding implicit interception routines. Before returning those handles to applications, we attach our own handle-based interception routines to them. As a

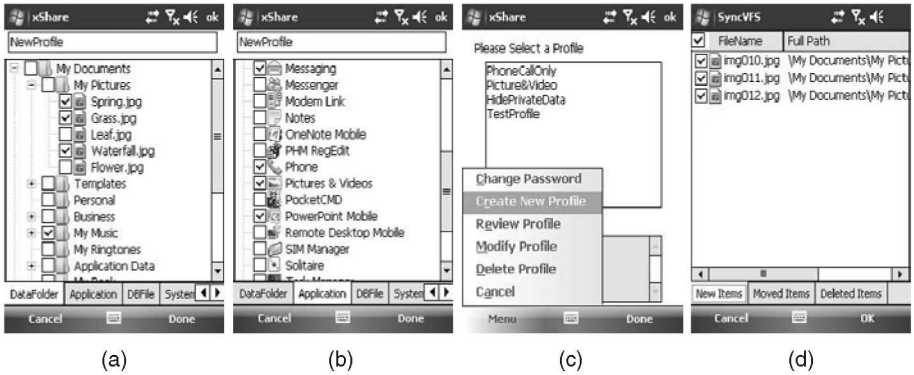


Fig. 11. User interface for owner. (a) Select files or folders to share, (b) select applications to share, (c) manage profiles, and (d) review the files changed in Shared Mode.

result, all calls to the handled-based APIs are directed to the xShare interception DLL. Fig. 10 illustrates how we intercept both implicit and handle-based API sets, using file system APIs as an example.

4.1.3 Access Control Implementation

xShare enforces the user-specified sharing policy in an interception routine for *CreateFile*(). The routine takes the path parameter passed to *CreateFile*() and uses a binary search to look it up in the list of the shared files to determine whether access to the file should be granted. If so, it calls the original *CreateFile*() to open the file and returns the file handle. Otherwise, it denies the access by returning an error. Because *CreateFile*() is called to open the executable when an application is launched, the interception routine ensures access control for both files and applications.

4.2 Namespace Virtualization

We implement xShare namespace virtualization on Windows Mobile by system API interception. We intercept 28 APIs in total.

4.2.1 File System Virtualization

According to the design described in Section 3.3, in Shared Mode, we provide a virtual file system to track and separate changes made in Shared Mode, hide nonshared files, and provide a consistent appearance to the borrower.

Change separation through path mapping. We employ a *virtual-intermediate-physical* path mapping to separate changes made in Shared Mode. We call the path used by an application the *virtual path* and translate it into the *intermediate path* by prefixing it with "\xShare\Root." We call the path to store the file in Normal Mode the *physical path*. For existing files that are shared from Normal Mode, their physical path is initially the same as their virtual path. For files created in Shared Mode, their physical path depends on how they are created. We employ a *virtual link* to maintain the mapping relationship between an intermediate path and its physical path. For each intermediate path, we create its virtual link as a file in the intermediate path by suffixing it with ".vlink." The virtual link file contains the physical path of the corresponding intermediate path. If a file shared from Normal Mode is opened for write in Shared Mode, xShare copies it to its intermediate path and generates its virtual link file. If a file is copied or moved to another path, xShare generates a virtual link file under the intermediate path of the destination. If a file is deleted, xShare treats it as moving the file to a *virtual recycle bin*. For preexisting shared files that have not been changed in Shared Mode, their virtual path is the same as their physical path and there is no need for a virtual link. By updating the corresponding virtual links, we record all the file change operations under "\xShare\Root\" without directly modifying any file shared from Normal Mode. This enables xShare to allow the owner to later manage changes made in Shared Mode.

Hiding nonshared files. To hide nonshared files, the interception routine for *CreateFile*() returns *ERROR_FILE_NOT_FOUND* when an application tries to open a nonshared file. We also intercept *FindFirstFile*() and *FindNextFile*() so that when an application searches a directory, nonshared files are invisible. To ensure a consistent view of the virtual file system in Shared Mode, xShare sets and obtains the attributes of the requested file using its intermediate path for each call to *SetFileAttributes*() and *GetFileAttributes*().

4.2.2 Registry Virtualization

We implement namespace virtualization for the registry as well, to protect it from unwanted changes. The registry plays a key role on Windows Mobile; it stores data and settings of the system and information about applications and drivers. As a temporary user may change registry settings in Shared Mode, xShare virtualizes registry access to track the changes and separate them from Normal Mode. The technique used for registry virtualization is the same to the one for file system virtualization.

4.3 Managing Unstoppable Services: CEMAPI

In Section 3.3, we described the challenge that a mobile OS may load private data into memory through services that are started at boot time. As an example, we next present our solution to an important service of this kind on Windows Mobile, CEMAPI, which provides COM style APIs for applications to access SMSes and e-mails. CEMAPI keeps SMS history in memory. Unfortunately, the shell process, shell32.exe, loads CEMAPI at boot time and CEMAPI cannot be terminated as it is tightly integrated with the OS. Therefore, it will retain the SMS history until reboot, further complicating SMS history protection when the SMS application is shared.

Our solution is to leverage the CEMAPI APIs as necessary upon entering Shared Mode to read all existing SMSes, save them into a nonshared file, and delete them from CEMAPI internal memory. As a result, the borrower will see none of the owner's SMSes. When switching back to Normal Mode, we restore all backed up SMSes to CEMAPI internal memory.

4.4 User Interfaces

We have implemented the UI design described in Section 3. We next provide details specific to Windows Mobile.

4.4.1 Owner UI

When xShare is launched for the first time, it prompts the owner to set a password. To switch back to Normal Mode, the owner must input the password correctly. We have implemented a hierarchical list-based UI design for policy specification and borrower data reconciliation, as illustrated in Fig. 11. In addition, through the xShare UI, the owner can configure other xShare properties, such as whether installing new applications or connecting to a PC is permitted. The owner can also limit the resources used in Shared Mode, specifically the usable storage capacity, battery percentage, and network traffic volume. These resources can be tracked using APIs already available on Windows Mobile. A button is devoted to Quick Launch, which places the phone into Shared Mode, with the application in the front and all of its open files being shared.

4.4.2 Borrower Shell Customization

The standard shell interface of Windows Mobile consists of the following main components: the start menu, title bar,

today screen, tray bar, and menu bar. The user can launch programs from any of the aforementioned components, with the exception of the title bar. In addition, most Windows Mobile phones have several physical buttons to launch predefined programs, such as placing phone calls or starting the camera. To enable a clean and simplified shell listing only shared applications and data, xShare customizes the shell for the borrower so that only shared applications and data are shown.

5 Evaluation of xShare

We evaluate our xShare implementation by measurement, usability study, and field evaluation. For measurement and usability study, we use the HTC Wizard phone running Windows Mobile 6.1 Professional with CE OS 5.2.

5.1 Measurement

We measure the memory footprint of xShare, its latency for switching from Normal Mode to Shared Mode, and the execution overhead in Shared Mode in terms of performance and energy. Our measurements demonstrate that our implementation achieves the design objectives in efficiency and mode switching latency.

5.1.1 Memory Footprint

xShare has two components: the interception DLL and the UI program. The interception DLL has 4,841 lines of C++ code and takes about 90 KB memory. The UI program has 5,023 lines of C# code and takes up to 1.3 MB memory. As the UI program is only used for mode switching, xShare requires only 90 KB memory in Shared Mode.

5.1.2 Mode Switching Latency

As our user studies in Section 2 indicate, a key design requirement for xShare is that the owner must be able to switch to Shared Mode with minimum latency. We break down the latency according to the main tasks involved in switching, as below.

- Terminate the running application processes (T1).
- Create the container file based on sharing policy (T2).
- Backup and delete SMSes (T3) if SMS is shared.
- Inject the interception DLL (T4).
- Customize the shell (T5).
- Other minor tasks (T6).

Measurement procedure. To measure the latency of switching to Shared Mode, we first create a representative system context on the HTC Wizard by loading 50 SMSes and launching Contacts, Messaging, Solitaire, ActiveSync, Phone, and Notes applications in Normal Mode. We then specify the sharing policy to allow access to Pictures and Videos, Solitaire, Phone, and Messaging applications along with 10 photos. We repeat the experiment 10 times.

Results. Fig. 12 shows the average time costs to perform these tasks sequentially. The sum of all the time costs is less than 5.8 seconds. We can see that it takes more than a second to terminate the existing processes. The reason is that xShare has to wait for the processes to release their resources and exit. Fortunately, this latency can be easily concealed from the owner: as it will take the owner a few seconds to interact with xShare to specify a policy, process termination can take place simultaneously in the background without introducing any noticeable latency.

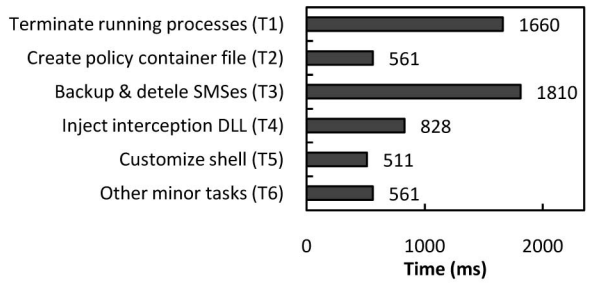


Fig. 12. Mode switching latency.

Another potentially lengthy task is to backup and delete the existing SMSes (~1.8 s). The main reason is that deleting SMSes using CEMAPI APIs is quite slow. However, xShare backs up SMSes only if the owner shares the SMS application. Moreover, similar to process termination (T1), this step can also take place in parallel as (T4-T6) after the owner specifies that the SMS application is to be shared.

Interception DLL injection creates the virtual environment for Shared Mode. The time cost is less than 1 second (0.83 s). It shows that xShare is able to create the virtual environment very quickly, even on our phone with just a 195 MHz CPU.

We also measure the latency of switching back to Normal Mode from Shared Mode. It takes about 3 seconds to unload the xShare interception DLL, terminate the applications launched in the Shared Mode, restart the autostart programs, restore the shell and system settings, and other necessary tasks. However, it can take place simultaneously with borrower data reconciliation. The time cost of borrower data conciliation depends on how much time the owner spends to review the data.

5.1.3 Execution Overhead

Because xShare is based on file-level access control, it introduces an overhead to most file accesses when enforcing the sharing policy. Our measurements show that xShare access control has little effect on the overall system operation in terms of performance and energy consumption. Moreover, as xShare does not intercept *ReadFile*() and *WriteFile*(), there is no overhead for reading and writing opened files. We next present measurements for a set of carefully designed benchmarks that involve file access in very different ways. Each measurement is repeated 10 times and we present the averaged results.

Benchmarks. We evaluate the performance impact of xShare on applications by preparing four benchmarks, as described below, and comparing their performance in Normal Mode and Shared Mode. First, since *CreateFile*() is called for every file access, we compare the time costs of calling *CreateFile*() on 10, 20, 50, and 100 empty files. We measure three cases: open existing files for read, open existing files for write, and create new files. Second, since xShare conceals nonshared files when listing the files in a folder, we use a test folder containing 100 files to measure the time costs when different numbers of files are shared. Third, to stress xShare with realistic file-intensive applications, we

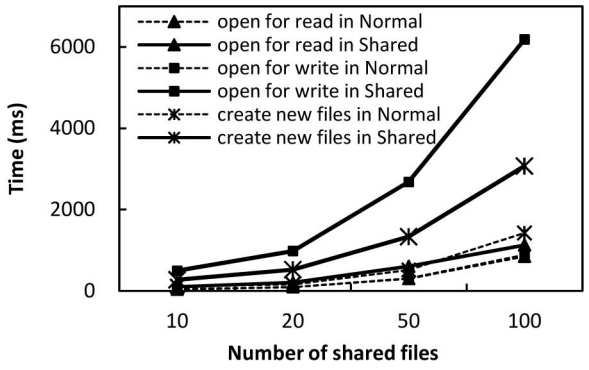


Fig. 13. Time costs for file operations.

write a *GZip* program that compresses and decompresses 100 files from three folders. The file sizes vary from 2 KB to 1 MB and the total data size is 7 MB. Fourth, to evaluate the overhead in application launching, we write a C# GUI program of 9 KB which simply opens a WinForm window and uses another program to launch the windows UI program and measure the launch latency.

Performance overhead. From Figs. 13 and 14, we can see that creating or opening a file for reading or writing and file listing operations have a significant execution overhead in Shared Mode. The reason is that xShare has to access the corresponding.vlink file for each file access, thus increasing the file access cost, especially when opening a file to write, as xShare uses copy-on-write. For file listing, xShare must traverse both the physical folder and the intermediate folder. However, while the relative overhead of file operations is large, the absolute time cost is small. For example, it takes only about 11 ms to open a file with read access, 62 ms to open a file with write access, and 31 ms to create a new file in Shared Mode. As most programs only spend a small fraction of its execution time opening, creating, or listing files, the overall overhead of a program can be small. For example, the extremely file-intensive *GZip* benchmark takes only 5.9 percent extra time to compress and decompress the 100 files in Shared Mode. Launching our simple GUI application requires more than 50 API calls on *CreateFile()* and *GetFileAttributes()* for the OS and.Net Compact Framework to load the execution file and other system binaries and resources. Yet it only takes an extra 270 ms (10 percent) to launch in Shared Mode.

TABLE 1
Policies in Evaluation

Policy	Apps.	Data
A	Phone call/SMS	none
B	Camera	5 photos

Energy overhead. We have used a Data Translation DT9802 data acquisition module to measure and calculate the energy overhead of xShare in Shared Mode. We measure and compare the total energy consumption in three benchmark tests in Shared Mode and Normal Mode: compressing and decompressing 100 files using *GZip*, playing a 1 minute MP3 file (192 kbps) in Media Player, and playing a 1 minute WMV video clip (1 mbps, 30 fps, 640×480) in Media Player. Our measurements indicate the file-intensive *GZip* benchmark costs only 4.7 percent extra energy in Shared Mode. We observed no measurable difference in energy consumption for the audio/video playback benchmarks. We attribute this to the fact that their main file operations are reading files which have no execution overhead in xShare.

5.2 Usability Evaluation

5.2.1 Evaluation by Owners

We evaluated the design of xShare by 12 mobile users to extremely positive feedback. The participants were aged between 18 and 39 years old and were recruited through the authors' social networks, e.g., by advertising the study through friends and coworkers. They all used a PC daily and five of them were existing Windows Mobile users. We designed and conducted a user study in which participants were given instructions on how to operate the phone if necessary, and then, given an introduction to various xShare functionalities. Each participant played around with xShare afterward for least 5 minutes, before being asked to specify policies *A* and *B* from Table 1 three times in succession, starting from the phone's home screen. To measure the participants' performance, we timed them on each run using a stopwatch. Fig. 15 shows the average time required by the participants separately, for prior Windows Mobile users and nonusers. Our results indicate that xShare has a very quick learning curve; our participants required only 20 seconds, on average, to specify the last policy, and non-Windows Mobile users required only 17 percent more time compared to prior Windows Mobile users.

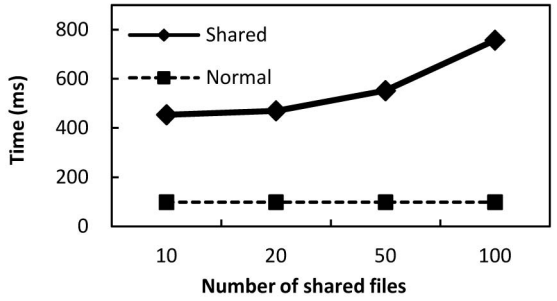


Fig. 14. Time costs for listing a folder.

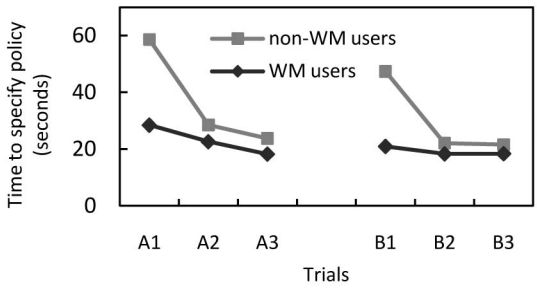


Fig. 15. Average time to start xShare and specify a policy.

After the timed experiments, we interviewed the participants with a structured survey about their subjective opinions on xShare. All participants agreed that xShare is easy to learn, and all but one agreed that it is useful. Only two participants responded that using xShare "takes too much time" and only one told us that it is not "easy to use." In contrast, nine of the participants responded their phones password protection "takes too much time." We attribute this to two factors. First, some phones may require entering the password at every access. Second, the built-in password protection takes too long compared to the little value it provides.

We received extremely positive feedback from our participants about xShare Quick Launch mode and borrower data conciliation. All but two responded that Quick Launch is useful. All the participants agreed that xShare is quick and easy to use. Similarly, all but one responded that xShare borrower data conciliation is useful. Only one participant responded that it takes too much time and another responded that it is not easy to use.

Overall, all participants responded that xShare satisfies their need for maintaining privacy when sharing their phone; in contrast, only one had the same opinion about the built-in password function of their phone. Furthermore, all but two responded that they would feel more comfortable sharing their own phones if they had xShare.

5.2.2 Evaluation by Borrowers

We evaluated the subjective performance, usability, and security of xShare for borrowers by eight expert Windows Mobile users to extremely positive feedback. The participants were aged between 18 and 39 years old and were recruited through the authors' social networks. They all had significant Windows Mobile experience, either as a programmer or as a power user who had at least changed their OS (ROM image) and phone software. In the user study, the participants were given a brief introduction on xShare and presented with two identical Windows Mobile phones, one without xShare and the other in xShare Shared Mode with the policies specified in Table 1.

To assess the security of xShare, the participants were offered a $30 prize for breaking xShare protection in either of the two policies. Breaking xShare protection was defined as accessing any of the functionality or files not intended for sharing, i.e., protected by xShare. Participants were allowed to do anything with the phone, including connecting it to a PC, with the exception of physical access to the phone internals and reflashing/reformatting the phone. Each participant had 30 minutes to break each profile, and we extended the time for any participant who requested it. Still, none of the participants were able to break xShare protection.

To assess the performance and usability of xShare, we interviewed the participants with a structured survey regarding their subjective opinion on the shared applications and phone. We focused on two items: 1) user friendliness and 2) performance of xShare. We asked the participants to perform several identical tasks with the non-xShare phone and the phone in Shared Mode. They were encouraged to perform each action on both phones as many times as necessary for an accurate assessment. The items we compared were the general phone UI, making a phone call, sending and receiving SMSes, and taking and viewing photos.

All participants responded that the UI of the xShare phone in sharing mode is consistent with the non-xShare phone. Furthermore, all but one participant told us that the two phones have the same user friendliness and speed for all the inquired functions. The one exception told us that the phone in Sharing Mode is actually faster to use in some cases since he has to go through less unwanted items. This confirms the fact that xShare does not induce a noticeable latency for most operations.

5.3 Field Evaluation

We evaluated xShare with two one-month field user studies by 12 Windows Mobile users in total, to very positive feedback, described below.

5.3.1 Participants and Data Collection Methods

We recruited 12 Windows Mobile phone owners in total. They were aged between 21 and 35 years old. Six participants were initially recruited through the authors' social networks, e.g., through e-mail solicitation among friends and coworkers. Another six participants were later recruited from the general public who didn't have any relationship with the authors. All but one were male. The participants had different models of phones: two Sony Ericsson Xperia X1 phones and 10 HTC phones, ranging from an older HTC Wizard to the latest HTC Diamond 2. All phones were running Windows Mobile Professional 6.0 or 6.1, and we successfully installed xShare on them. One participant upgraded his phone to another Windows Mobile device during the study, and we installed xShare on his new phone. We provided training to all participants to ensure that they were comfortable with xShare. While participants were free to create and remove xShare profiles, we provided and configured three default profiles: "phone call only" which only allows making a phone call, "games" which shares the default Windows Mobile games, and "all apps" which allows all applications but not private data like SMSes, e-mails, contacts, and calendar.

In order to record the participants experience with xShare over the one-month duration of the study, we supplied them with diaries to take note of their phone sharing events. For each sharing event, the diary asks multiple choice questions regarding when, why, where, with whom, and for what applications did they share their phone, and whether xShare was utilized. We used xShare logs to determine the length of the sharing session, the time spent on configuring xShare, and what specific items were shared. After the one-month study, we conducted an interview with each participant to gather their phone sharing stories and ask about their user satisfaction and opinion on xShare, through questions such as "xShare made me feel more comfortable to let other people use my phone" and "xShare can satisfy my need for maintaining privacy when sharing my mobile phone." We used a five-level Likert scale [18] for the responses: completely disagree, generally disagree, neutral, generally agree, and completely agree. We did not observe any significant difference between the responses of the two groups we studied; therefore, we present their data collectively.

5.3.2 Findings

We collected 63 phone sharing events in total, on average, 5.25 events per participant. Each participant shared phone at least once and the most active participant shared phone for 13 times. xShare was used for 53 of the 63 sharing events. The average sharing period is 9.2 minutes (min: 38 seconds, max: 2 hours and 38 minutes). Only three of the sharing events lasted longer than 15 minutes. Without counting those three sharing events, the average sharing period was 5 minutes. The field evaluation provided several new findings and confirmed others.

The phones were shared as suggested by the international interviews. The observed characteristics of phone sharing were consistent to what we found in our international user study. It happened with different people (coworkers, family members, friends, or even strangers), in various locations (mainly schools and workplaces but also other places like restaurants, parties, and conferences), and for a wide range of applications (making a phone call, games, viewing or taking pictures, etc.).

Trying new or different phones was popular. Twenty-three of the 63 sharing events were because the borrower asked for trying the lender's phone which was new or different. For the participant who upgraded his phone from HTC Touch HD to HTC Diamond 2, he shared his phone for eight times in five following days and three of the eight sharing events occurred in the first day.

A new popular way to share phones happened between parents and kids (seven of the 63 sharing events). One participant had a two-year old kid who always asked to play with the participant's phone whenever seeing it. The kid was able to play some simple games, but most of the time the kid just played with the phone as a toy, e.g., randomly pressing the buttons and the touch screen. This finding was confirmed by a recent report [19] of "parents using smartphones to entertain bored kids." This is actually a new threat for phone sharing: the kid may unconsciously change or delete important data such as contacts or e-mails, or make an unwanted phone call. This threat has nothing with privacy but requires protecting important data and preventing unsafe operations. xShare perfectly addresses this threat by applying access control to the related data and applications. Without xShare, the participant gave the phone to the kid only if the kid strong requested and had to keep a watchful eye on what the kid was doing with the phone. With xShare, the participant felt much more comfortable in sharing phone with the kid because that xShare can be used to restrict the usage of the phone, e.g., only allowing playing games.

Phone sharing for making a phone call was popular. Nineteen of the 63 sharing events were for making a phone call when the borrower's phone was not in hand or out of battery. Making a phone call was the number one application in all the sharing events, which is consistent to what we found in our international user study. xShare helped the participants in sharing their phone. During the final interview, one participant shared his story that without xShare, he rejected the request from a stranger for making a phone call, but with xShare, he felt much more comfortable in sharing his phone.

Predefined profiles were heavily used. For 42 of the 53 sharing events using xShare, predefined profiles were utilized. Profiles are very useful for the phone sharing events with fixed pattern like making a phone call and playing games. For those 42 sharing events, the average configuration time was 5 seconds. For the other 11 sharing events not using predefined profiles, the average configuration time was 25 seconds. Note that even for the sharing events using a predefined profile, the participants might review a profile before selecting it, which increased the configuration time. Directly selecting a predefined profile without review takes only 1 or 2 seconds and we observed that the minimal configuration time was just 1 second. These results demonstrated that xShare achieved its design goal for quick policy specification by using profiles.

Most of the phone sharing events were initiated by a borrower. Only nine of the sharing events were initiated by a lender while 54 sharing events were initiated by a borrower, which shows that phone sharing was mostly driven by the needs of the borrowers. Typically, a lender initiates phone sharing only when the lender has new or interesting data (e.g., photos from a recent trip) or applications to share with the borrower. Unfortunately, no participant had a trip during our field evaluation. For the nine sharing events initiated by a lender, six of them were for the lender to show certain features of applications of the phone to the borrower, two of them were for the lender to show certain data in the phone to the borrower, and the last one was that the lender wanted to show some pictures on the Internet to the borrower.

xShare was able to meet the needs of the participants in sharing their mobile phones. The active usage of xShare in the field evaluation demonstrated that xShare was very useful in real life of the participants and achieved its goals. We got very positive feedback. Seven participants responded "completely agree" and four participants responded "generally agree" on "xShare made me feel more comfortable to let other people use my phone." Only one participant responded "neutral" for the above statement. During the final interview, that participant told that he didn't have any private data in his phone so he didn't care how the borrower used his phone. For "Overall, I like xShare," six participants responded "completely agree" and three participants responded "generally agree." Only two participants responded "neutral" for the statement and the reasons were due to the limitations of xShare as described below in Section 5.3.3. All participants agreed on "xShare can satisfy my need for maintaining privacy when sharing my mobile phone" (five participants responded "completely agree" and seven participants responded "generally agree"). It was reported that xShare was most helpful when the phone owners had private or important data in their phone or they didn't fully trust the borrower. xShare also promoted phone sharing. Five participants reported cases where they shared their phone but wouldn't have done so without xShare.

5.3.3 Limitations of xShare Realization

One open question we asked the participants was about "the biggest shortcoming of xShare," from which we got some good feedbacks on how to further improve xShare.

The borrower will see the use of xShare and know that the lender is hiding something. Currently, to use xShare,

the user must first launch xShare, do some configuration, or select a profile, and then, switch into Shared Mode. Likely, the borrower will see that procedure and know that the lender is hiding something. Five participants pointed out that this somehow made both the borrower and the lender uncomfortable as the borrower might think that the lender did not trust the borrower. xShare was reportedly not used in two phone sharing events due to this reason: "I didn't want the borrower to figure out that I was hiding anything." We agree on that this is a valid problem general for all the phone sharing tools. One possible solution is to further integrate xShare with the system and make it invisible. One participant suggested a solution for the problem: use a special passcode to unlock phone for sharing. Before handing the phone to the borrower, the lender can input a special passcode to unlock the phone and tell xShare to directly switch into Shared Mode with a predefined profile. Thus, the borrower will only see that the lender unlocked the phone as normal but doesn't know that xShare is used. However, the user must remember a special passcode for each profile/sharing event.

xShare is still slow. Although all the participants agreed that xShare was much faster than preparing a phone for sharing by moving or deleting private information, five participants reported that xShare was still slow to initialize, and this sometimes prevented the participants from using xShare. One must explicitly select a profile or configure the access policy and it takes xShare for several seconds to switch into Shared Mode. If a lender is busy or the borrower asks to use the phone immediately, the lender may not have time to use xShare. For the other 8 of the 10 sharing events where xShare was not used, the participants didn't have time to use xShare due to various reasons, e.g., the borrower wanted the phone immediately, the participant was driving or was busy on something else, or the participant just did not want to spend time to use xShare. For those events, the participants also reported that they trusted the borrower. We agree that configuring access control policy always takes time and suggest using predefined profiles whenever possible. The main mode switching latency is from terminating running processes. Advances in future systems and applications may avoid process termination.

6 Related Work

The underpinning motivation of xShare is that mobile users are interested in sharing their devices. In recent years, researchers have investigated information technology sharing [1], [2], [4], [8], [12], [13], [16]. In particular, the authors in [13] pointed out that "face-to-face media sharing" is desirable but not well supported by the existing technologies. The authors in [2] studied the culture factors behind phone sharing in rural India and highlighted the necessity of phone sharing in underserved communities. The authors in [8] studied the concerns when sharing mobile phones and confirmed that phone sharing is also a common usage scenario to middle-class Americans. In [19], it was reported "parents using smartphones to entertain bored kids." Without providing a solution, these works are all motivational to xShare. Our user studies in Section 2 are complementary with the ones of [2] and [8] in terms of demographics. The authors in [8] also had findings similar to ours: the "all-or-nothing" binary security mode of most mobile phones cannot meet the security and privacy needs of users when sharing phones, and a flexible and lightweight security model is desired.

The system implementation of xShare is related to existing work in OS and application-level virtualization [6], [7], [10], [11], [14], [15], [17]. Companies like VMware recently announced VM solutions for mobile platforms [21]. Yet xShare has a very different design goal. Virtualization solutions aim at isolating multiple VMs from each other and preventing them from altering each other's data. They may not necessarily prevent VMs from reading each others' data, or system data, e.g., [17]. In contrast, xShare aims at preventing a single VM from accessing nonshared data and applications. Therefore, it can be significantly lighter in employing a different approach. As mobile devices are expected to remain processor and energy constrained compared to their PC counterparts, we expect that the additional overhead of VM solutions in terms of processing power and battery life would remain significant.

xShare is related to role-based access control [9], but switching roles must be fast and discrete in xShare. Conventional multiuser support, as present in desktop OSes, are designed for a computer that will be actively used by multiple but usually known users. It creates a complete system profile for each user and provides different privilege levels to each of them. In contrast, we aim at supporting a mobile device that is owned and actively used by a single user, and occasionally lent to other users. Therefore, there is no need to create a system profile for each possible user. While a *Guest* account may be used for all temporary users, it does not allow the owner to grant different temporary users with different accesses in an impromptu manner. For example, one may be willing to lend their phone to a colleague to make a quick call but only be willing to share some photos captured at a recent party with close friends.

Windows CE supports a Kiosk mode that can boot directly into an application without access to a shell, control panel, or any other method of launching other applications [3]. Similarly, some third parties provide kiosk-like modes with multiple applications, e.g., SPB Kiosk Engine [20]. However, such kiosk modes are fatally limited for phone sharing. First, a customized Windows CE image is needed and a lengthy reboot is necessary, making impromptu sharing impossible. Moreover, they do not provide protection over data files.

7 Discussion

xShare is intended to provide usable, lightweight protection against unauthorized access by borrowers, instead of making the system more resilient against attacks to existing security flaws. Even with xShare, existing security flaws in the phone may be used to compromise the phone and xShare. Even so, xShare improves overall system security in Shared Mode by providing file access control. For example, without xShare, the borrower can use the Internet Browser to download pre-prepared malicious software. With xShare, the malicious software cannot be simply downloaded and run, as xShare prevents launching an application that has

not been explicitly shared. Malicious borrowers can still overpower xShare by exploiting inherent system security flaws, including ActiveX in the Internet Explorer, and OS bypassing. Further, if someone hacked the system and posted the method online, other people would be able to leverage it. Addressing such unauthorized access based on system and application security shortcomings is out of the scope of this work.

xShare is limited in that it is built atop of the OS, assuming that the ROM image may not be modified. As seen in our Windows-Mobile-based implementation, many challenges are indeed posed by the OS itself, e.g., regarding CEMAPI. Our design and implementation, however, provide insight into how the OS can be modified to better support sharing atop of the OS. Moreover, our work serves as a blueprint for possible OS-based designs.

Implementing xShare heavily depends on the underlying OS, and different approaches may be used on systems other than Windows Mobile. For example, on the iPhone OS, one may leverage the native multiuser support in the kernel to simplify the access control required by xShare. Even so, implementing Shared Mode on the iPhone still requires system-level changes (e.g., system API interception) and nontrivial effort to create a virtual runtime environment and to virtualize resource access and separate changes made in Shared Mode.

8 Conclusion

In this paper, we present a complete research and development cycle of xShare for friendly, efficient, and secure sharing of mobile phones. We find that phone sharing is very popular and involves a wide range of applications, reasons, social settings, and relationships. Yet privacy remains a major concern that prevents users from sharing their phone, and existing systems provide inadequate privacy protection for sharing. Our user studies highlight the need for an efficient and secure solution for impromptu sharing of mobile phones.

Based on these findings, we present xShare, a software solution to support friendly, efficient, and secure phone sharing on existing systems. xShare allows phone owners to easily create impromptu sharing policies to decide which files and applications to share and control how much resource a borrower can use. Using the sharing policies, xShare creates a virtual environment, called Shared Mode, where only the explicitly shared files and applications are visible. Further, xShare enables the owner to manage data files and settings created or modified by the borrower.

We show that xShare can be realized for Windows Mobile without any ROM image changes or extra support from the operating system.

We evaluate our Windows-Mobile-based implementation through a set of carefully designed benchmarks and user studies. The measured results and positive user feedback demonstrate that our implementation has negligible overhead, provides robust protection on sensitive data and applications, and is able to achieve its design goals.

Acknowledgments

The authors thank their colleagues for fruitful discussions and valuable suggestions. The Rice team was supported in part by US National Science Foundation awards IIS/HCC 0803556, CNS/NeTS 0721894, and CRI/IAD 0751153. They also thank their reviewers and their shepherd, Professor Eyal de Lara, for the conference paper of this work. The conference comments and shepherding also helped improve this journal paper. Last but not least, they thank the participants of their user studies for their participation and valuable comments.

References

[1] A. Brush and K. Inkpen, "Yours, Mine and Ours? Sharing and Use of Technology in Domestic Environments," *Proc. Int'l Conf. Ubiquitous Computing*, pp. 109-126, 2007.

[2] A.L. Chavan and D. Gorney, "The Dilemma of the Shared Mobile Phone—Culture Strain and Product Design in Emerging Economies," *ACM Interactions*, vol. 15, pp. 34-39, 2008.

[3] M. Hall, "Create a Windows CE Image that Boots to Kiosk Mode," http://msdn.microsoft.com/en-us/libraryaa446914.aspx, 2009.

[4] R. Hull, B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas, "Enabling Context-Aware and Privacy-Conscious User Data Sharing," *Proc. IEEE Int'l Conf. Mobile Data Management*, 2004.

[5] G.C. Hunt and D. Brubacher, "Detours: Binary Interception of Win32 Functions," *Proc. Conf. USENIX Windows NT Symp.*, 1999.

[6] S. Jain, F. Shafique, V. Djeric, and A. Goel, "Application-Level Isolation and Recovery with Solitude," *Proc. Third ACM SIGOPS/EuroSys European Conf. Computer Systems*, 2008.

[7] P.H. Kamp and R.N.M. Watson, "Jails: Confining the Omnipotent Root," *Proc. Second Int'l SANE Conf.*, 2000.

[8] A.K. Karlson, A.J.B. Brush, and S. Schechter, "Can I Borrow Your Phone?: Understanding Concerns When Sharing Mobile Phones," *Proc. SIGCHI*, 2009.

[9] B. Lampson, "Computer Security in the Real World," *Proc. Ann. Computer Security Applications Conf.*, 2000.

[10] Z. Liang, V.N. Venkatakrishnan, and R. Sekar, "Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs," *Proc. 19th Ann. Computer Security Applications Conf.*, 2003.

[11] B. des Ligneris, "Virtualization of Linux Based Computers: The Linux-VServer Project," *Proc. 19th Int'l Symp. High Performance Computing Systems and Applications*, pp. 340-346, 2005.

[12] J.S. Olson, J. Grudin, and EricHorvitz, "A Study of Preferences for Sharing and Privacy," *Proc. Extended Abstracts on Human Factors in Computing Systems*, 2005.

[13] T. Pering, D.H. Nguyen, J. Light, and R. Want, "Face-to-Face Media Sharing Using Wireless Mobile Devices," *Proc. IEEE Int'l Symp. Multimedia*, 2005.

[14] D. Price and A. Tucker, "Solaris Zones: Operating System Support for Consolidating Commercial Workloads," *Proc. 18th USENIX Conf. System Administration*, 2004.

[15] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," *ACM SIGOPS Operating Systems Rev.*, vol. 41, pp. 275-287, 2007.

[16] A. Voida, R.E. Grinter, N. Ducheneaut, W.K. Edwards, and M.W. Newman, "Listening In: Practices Surrounding iTunes Music Sharing," *Proc. SIGCHI*, 2005.

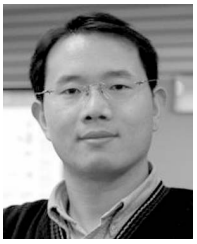
[17] Y. Yu, F. Guo, S. Nanda, L.-c. Lam, and T.-c. Chiueh, "A Feather-Weight Virtual Machine for Windows Applications," *Proc. Second Int'l Conf. Virtual Execution Environments*, 2006.

[18] Likert Scale, http://en.wikipedia.org/wiki/Likert_scale, 2010.

[19] Parents Using Smartphones to Entertain Bored Kids, CNN Living with Technology, http://www.cnn.com/2010/TECH/04/26/smartphones.kids/index.html?hpt= Mid, 2010.

[20] SPB Software House, "SPB Kiosk Engine," http://www.spbsoftwarehouse.com/products/kioskengine, 2010.

[21] VMware Corporation, "VMware Mobile Virtualization Platform," http://www.vmware.com/technology/mobile, 2010.



Yunxin Liu received the BS degree from the University of Science and Technology of China in 1998, and the MS degree from Tsinghua University in 2001. He is currently working toward the PhD degree at Shanghai Jiao Tong University. In 2001, he joined Microsoft Research Asia, where he is currently a researcher. His research interests include computer networking, system design, and mobile computing. He is a member of the IEEE.

Ahmad Rahmati received the BS degree in computer engineering in 2004 from the Sharif University of Technology, Tehran, Iran, and the MS degree in electrical and computer engineering in 2008 from Rice University, Houston, Texas, where he is currently working toward the PhD degree. He was a research intern at AT&T Labs-Research, Motorola Labs, and the Deutsche Telekom R&D Lab, in Summer 2006, Summer 2008, and Spring 2010, respectively. He received the ACM MobileHCI Best Paper Award in 2007. His research interests include the design and applications of efficient, context-aware mobile systems, system power analysis and optimization, and human-computer interaction. He is a student member of the IEEE.

Hyukjae Jang received the BS degree from Yonsei University in 2003. He is currently working toward the PhD degree at KAIST. He was a research intern at Microsoft Research Asia from September 2008 to March 2009.

Yuanhe Huang received the BS degree in 2009 from Tsinghua University, where he is currently working toward the MS degree. He was a research intern at Microsoft Research Asia from June 2008 to March 2009.

Lin Zhong received the BS and MS degrees from Tsinghua University in 1998 and 2000, respectively, and the PhD degree from Princeton University in September 2005. He was with NEC Labs, America, for the Summer of 2003, and with Microsoft Research for the Summers of 2004 and 2005. He joined the Department of Electrical and Computer Engineering, Rice University, as an assistant professor in September 2005. He received the AT&T Asian-Pacific Leadership Award in 2001 and the Harold W. Dodds Princeton University Honorific Fellowship for 2004-2005. He coauthored a paper that was selected as one of the 30 most influential papers in the first 10 years of the Design, Automation, and Test in Europe conferences. He and his students received Best Paper Awards from ACM MobileHCI 2007 and IEEE PerCom 2009. His research interests include mobile and embedded system design, human-computer interaction, and nanoelectronics. He is a member of the IEEE.

Yongguang Zhang received the PhD degree in computer science from Purdue University in 1994. He is currently with Microsoft Research Asia, where he is a senior researcher and the research manager for the Wireless and Networking research group. From 1994 to 2006, he was with HRL Labs in Southern California, leading various research efforts in internetworking techniques, system developments, and security mechanisms for satellite networks, ad hoc networks, and 3G wireless systems. At HRL, he was a co-PI in a US Defense Advanced Research Projects Agency (DARPA) Next Generation Internet project and a technical lead in five other DARPA-funded wireless network research projects. From 2001 to 2003, he was also an adjunct assistant professor of computer science at the University of Texas at Austin. His current interests include mobile systems and wireless networking. He has published one book and more than 50 technical papers in top conferences and journals in his field (like Sigcomm, MobiCom, Mobisys, and ToN). He recently won the Best Paper Award at NSDI 2009 and four Best Demo Awards in a roll: at MobiSys 2007, at SenSys 2007, again at MobiSys 2008, and at NSDI 2009. He is an associate editor for the *IEEE Transactions on Mobile Computing*, was a guest editor for the *ACM MONET Journal*, and has organized and chaired/cochaired several international conferences, workshops, and an IETF working group. He was a general cochair for ACM MobiCom 2009. He is a member of the IEEE.

Shensheng Zhang received the PhD degree from Stanford University in 1988. He is currently a professor at Shanghai Jiao Tong University.