Porscha: Policy Oriented Secure Content Handling in Android

Machigar Ongtang
Faculty of Information Technology
Dhurakijpundit University
Bangkok 10210, Thailand
machigar.ong@dpu.ac.th

Kevin Butler
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403 USA
butler@cs.uoregon.edu

Patrick McDaniel
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
mcdaniel@cse.psu.edu

ABSTRACT

The penetration of cellular networks worldwide and emergence of smart phones has led to a revolution in mobile content. Users consume diverse content when, for example, exchanging photos, playing games, browsing websites, and viewing multimedia. Current phone platforms provide protections for user privacy, the cellular radio, and the integrity of the OS itself. However, few offer protections to protect the content once it enters the phone. For example, MP3-based MMS or photo content placed on Android smart phones can be extracted and shared with impunity. In this paper, we explore the requirements and enforcement of digital rights management (DRM) policy on smart phones. An analysis of the Android market shows that DRM services should ensure: *a*) protected content is accessible only by authorized phones *b*) content is only accessible by provider-endorsed applications, and *c*) access is regulated by contextual constraints, e.g., used for a limited time, a maximum number of viewings, etc. The *Porscha* system developed in this work places content proxies and reference monitors within the Android middleware to enforce DRM policies embedded in received content. A pilot study controlling content obtained over SMS, MMS, and email illustrates the expressibility and enforcement of Porscha policies. Our experiments demonstrate that Porscha is expressive enough to articulate needed DRM policies and that their enforcement has limited impact on performance.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: General
; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls, Authentication*

General Terms

Security

Keywords

DRM, Mobile Phone Security, Android, Security Policy

1. INTRODUCTION

Mobile phones are used extensively by nearly 5 billion people worldwide [28] and form a vital information conduit for business and personal information access. The use of mobile phones has long transcended strictly voice calling, e.g., SMS exceeded 5 trillion messages worldwide in 2009 [43]. As users perform ever more data-centric activities on their phones, they are increasingly purchasing *smartphones* with the ability to run wide varieties of applications. Over 170 million smartphones were purchased globally in 2009 [18]. The data and applications used on these phones are equally diverse. For example, the Apple App Store, which contains over 130,000 applications, recorded 280 million application downloads in December 2009 [21].

Open platforms such as Android provide few direct protections for the content placed on the phone. Access controls restrict access to application interfaces (e.g., by placing permissions on application components in Android), rather than placing explicit access controls on data they handle. Therefore, what limited content protections exist are largely a by-product of the way interfaces are designed and permissions (often capriciously) assigned. Thus, a malicious application with the appropriate permissions can exfiltrate even the most sensitive of data from the phone. Malware has recently begun to exploit such limitations [3, 4, 5]. Moreover–even in the absence of malicious applications–commercial interests such as media providers wish to provide content without exposing themselves to content piracy.

To combat these issues, a consortium of mobile phone manufacturers including Nokia, LG, Motorola, Samsung, and Ericsson have recently developed standards for content protection on mobile devices. Codified within the Open Mobile Alliance and focusing primarily on pay-per-use content such as ringtones and multi-media, the OMA DRM v1.0 [37] and v2.0 [38] standards define an API and infrastructure for authorizing *devices* to process content. To simplify, OMA DRM devices obtain rights objects (use licenses and cryptographic keys) from providers that allow them to access downloaded content. The licenses can regulate how the content may be used in simple ways such as by discrete lifetime and maximum number of uses. The granularity of the OMA DRM specifications, however, is coarse. The licensing unit is the phone; as a result, the specifications say nothing about content management when it is on the phone. Specifically, there are no considerations of which applications may access content. This was reasonable when the specification was written in 2004, as there were no application markets at the time and phone manufacturers provided their own software for the phone. Since that time, though, the smartphone revolution has mandated a need for protections at the application layer, now that many applications can be purchased or downloaded to access on-phone content. Otherwise, content is subject to improper use by untrusted applications such as rogue media players.

In this paper, we introduce the Policy ORiented Secure Content Handling for Android (*Porscha*) system. Porscha enforces fine-grained content policies[1] over content delivered to the phone. A study of application markets (see next section) illuminates the needs of providers: from financial transactions and airline tickets, to personal applications such as diaries and journals, the diversity and sensitivity of content processed by smartphones is immense. This study prompts three classes of fine-grained content policies studied throughout: a) content should only be accessible by explicitly authorized phones, b) content should only be accessed by provider endorsed applications, and c) content should be subject to contextual constraints, e.g., used for a limited time, a maximum number of viewings, etc. In supporting these policies, we extend the OMA DRM policy schema [37] to embrace finer-grained controls.

Porscha policies are enforced in two phases: the protection of content as it is delivered to the phone (in transit, see Section 4.2), and the regulation of content use on the phone (on platform, see Section 4.3). For the former, Porscha binds policy and ensures content confidentiality "on the wire" using constructions and infrastructure built on Identity-Based Encryption [11]. For the latter, Porscha enforces policies by proxying content channels (e.g., POP3, IMAP, Active Sync) and placing reference monitor hooks within Android's *Binder* IPC framework. We implement and test Porscha on a T-Mobile G1 smartphone and perform experiments using the three most popular content types: SMS messages, MMS messages, and email. Our experiments with Porscha show that delivery delay for MMS is slightly over 1 second, while latency from processing emails is only about 1 second or less. A security analysis is given and we conclude with a discussion of alternate designs and policy and infrastructure extensions. We begin in the next section by considering whether content policy is necessary for smartphones.

2. DO SMART PHONES NEED DRM?

In looking at DRM in smartphones, we must ask the obvious question of what must the service actually do. To answer this question, we surveyed applications and usage seen in current cell phones to attempt to ascertain what kinds of documents are commonly exchanged and what the reasonable requirements are that the providers may place on them. We evaluated the top 50 free Android applications in each of the 16 application categories present Android Market in April 2010. For the purposes of this initial study, we focused on applications that delivered content via SMS, MMS, or email. Table 1 shows the number of applications requesting permissions to receive SMS and MMS and to read from or write to SMS, MMS, and email attachments. We briefly summarize our findings on content use below:

Personal and Business Documents:.

Applications in the Communication category (e.g. third-party email/SMS/MMS clients), Tools category (e.g. anti-virus, backup tools, and Office Viewer), and Travel category (e.g. language translator) in Table 1 frequently manage personal or business SMS, MMS, and emails.

Documents in this category include sensitive emails between business partners and others encompass security capabilities, e.g., SMS is used for authorization in access control systems such as Grey [9]. In these cases, unintended exposure can leak business secrets or compromise the access control system. Thus, providers need to ensure that (a.) only targeted phones (i.e. authorized users) receive the documents, (b.) only trusted client applications can handle them, and that (c.) these documents can never be modified.

Service-specific data:.

A number of applications use SMS to send commands to on-phone clients. Note that in almost all cases, there should only be one legitimate client consumer for the content type– the one provided by the service itself. For instance, one spy camera application used SMS to command the phone capture pictures and record videos. Similarly, Mydroid[2] is a tool for finding the phone by turning off the silent mode and turning up the volume when it receives a command via SMS, and Mobile Defense[3] allows remote connection to the phones after receiving an authenticated SMS message.

Commands in these applications are sent or received via provided websites or other interfaces. Unauthorized exposure of the "command" documents could reveal the application behavior, and indirectly the user's intent. The applications may misbehave if the commands are tampered with. In response, the senders must let (a.) only the phones under control receive the commands, (b.) only the applications to execute the commands to process them, and (c.) the commands read only and may be read only once, and (d.) ensure only legitimate content is consumed by the client.

Financial Information.

Emails and SMS have become key media for financial institutions to communicate with their clients. For instance, banks and credit card companies offer SMS banking, SMS account alerts, and e-statements. Payment service providers such as PayPal and Amazon Payments mainly contact their customers via email and also offer SMS-based payment service. Similar to personal and business documents, the sending institutions aim to inform the users. As a result, the documents must be (a.) sent only to the phones of such particular customers. They must be (b.) accessed only by trusted messaging clients. Moreover, some documents such as payment requests may be designed to work with a group of payment applications trusted by the institutions which can also be identified by their hashes or signatures. In most cases, the senders should also ensure that (c.) these documents are read-only. They should be deleted only through trusted client applications.

[1] Throughout we use the terms DRM and content policies interchangeably. While the latter term is arguably more general, any distinctions are outside the scope of the definition and enforcement of policy studied here.

[2] http://code.google.com/p/mydroid/

[3] https://www.mobiledefense.com/

Application Category	Receive SMS	Receive MMS	Read SMS	Write SMS	Read Attachment
Communication	7	2	10	6	1
Tools	5	2	6	3	0
Finance	1	0	1	0	0
Travel	1	0	1	0	0
Others	3	1	2	3	1

Table 1: Number of sample applications that access SMS, MMS, and email.

2.1 DRM Policy Requirements

The surprising result of the application analysis was the incredible consistency of the content policy requirements. With few exceptions, the requirements fell into three categories:

- **Binding content to the phone** – most of the applications required that the content be targeted to a single identified user or phone. Failure to implement this policy could have catastrophic consequences (in financial applications), or undermine the entire service (in media access applications).

- **Binding content to endorsed applications** – often observed in desktop environments but largely ignored in the mobile device industry, it is important to control which particular applications can process protected content. The consequences of the failure to enforce this policy are similar to those above—a malicious application on an otherwise legitimate phone could corrupt, exfiltrate, or otherwise misuse delivered content.

- **Constraining continuing use of the content** – it is essential that the provider be able to control not just access, but how that access evolves or expires over time. Frequency, count, or temporal constraints were common. Failure to provide these polices would marginalize the license structures upon which many services are now built.

To illustrate, Table 2 gives example policies falling into these categories (using a self-explanatory policy schema). Policy 1 states that a particular document (for example, an authorization SMS in our access control system example) can be read only if the application is signed with a particular key and the phone is within 50 meters from the place under control (e.g., as used in the Grey system). Policy 2 explicitly identifies a legitimate application by its fingerprint (i.e., a hash of the application image `.apk` file). We revisit these policies in subsequent sections.

3. BACKGROUND AND ASSUMPTIONS

Having considered DRM policy requirements, we now examine how content is currently delivered to mobile phones. We begin this section by briefly describing Android, then discuss content delivery through the network and its handling on the phone itself.

3.1 Android Background

Android is a middleware platform for mobile phones, built on a Linux kernel that isolates applications by having them run in their own process spaces with their own virtual machine instances. Operating system details are hidden from application developers, who access and provide functionality through *components*. These components are assembled to provide applications, which perform all inter-process communication (IPC) through Android's *Binder* mechanism. Components interact primarily through the use of *Intent* messages, which can either explicitly address components by name or through implicit *action strings*, resolved to the appropriate receivers by the Android middleware. Components set up *Intent filters* and specify action strings in order to subscribe to these specific Intents.

There are four types of components in Android. *Activity* components normally provide user interfaces via the touch screen and keypad. *Service* components perform background processing. *Broadcast Receiver* components act as listeners which enable asynchronous event notifications. They usually receive Intents addressed by action strings. Standard action strings include "boot completed" and "SMS received". Lastly, *Content Provider* components act as a persistent data store that implement an SQL interface. If implemented by the providing application, other applications can query, update, and delete the data in the Content Providers. Figure 1 describes how these components interact with each other. In addition, applications have the ability to directly call APIs from other applications.

Applications in Android can be classified into two groups. *System* applications, including the phone, dialer, and messaging applications, are bundled with the phone when it is provisioned to a subscriber and are stored in a read-only system partition. *User* applications are obtained from a variety of sources, including the Android Market, and are installed by users: these programs can compete with or complement system applications, or extend the platform's functionality in very different ways.

Android's security framework is based on *permission labels*, which are unique text strings defined either by applications or the middleware. Application developers specify a list of permission labels in an application's *manifest* file, which presents information about resources an application is allowed to access. For example, receiving and reading an SMS message requires an application to hold the `RECEIVE_SMS` and `READ_SMS` permissions, respectively. In Android, all content is treated equally, meaning that *any application with permissions to access a particular document type can access all documents of that type.* Importantly, elevated protection levels such as allowing "dangerous" functionality for certain permissions rely on the user to confirm all permissions associated with an application at install time; however, users who may not fully understand what they are allowing and consequently may make bad security decisions.

3.2 Documents in Transit

Figure 2(i.) provides an overview of how content is delivered to a phone from outside sources. For clarity, we refer to these external providers of content as *content sources*. Doc-

(1) allow-read{(sig=934db3d4 . . .) and (location=40.304107,-75.585938,50)}
Only the access control application signed by developer key 934db3d4 . . .. *can read the document, and only when the phone is within a 50-meter radius of location (40.304107,-75.585938).*
(2) allow-read{(hash=6ab843a)} allow-modify{none} allow-delete{(hash=6ab843a)}
Only the application identified by its binary hash 6ab843a *can read and delete the document.*

Table 2: Examples of security policies for content protection.

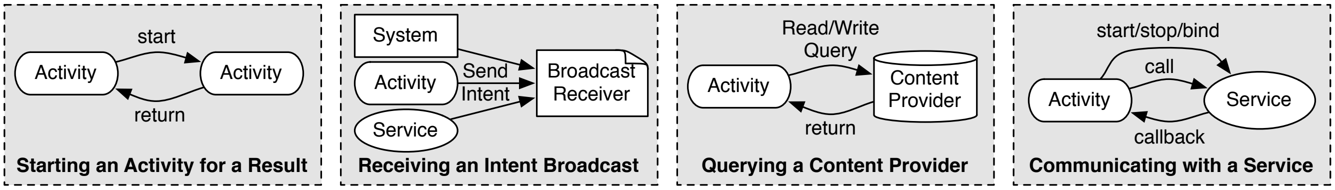


Figure 1: Typical methods of component interaction in Android.

uments sent through the cellular (SS7) network, including SMS and MMS messages, are received at the phone's Radio Interface Layer (RIL), processed by the baseband processor, and made accessible to the phone application. Applications whose content source originates from the Internet, however, connect directly to them in order to receive these documents, such as email. There are no intermediaries on the phone to process this content prior to its handling by the application.

Lack of end-to-end security is a major problem in SMS, MMS, and email transport. SSL/TLS for email delivery only secures connection between the phones and the mail servers. SMS and MMS documents delivered through SS7 make use of security mechanisms found within the cellular network. The heart of the SMS system is the Short Message Service Center (SMSC) which receives short messages from mobile devices inside the cellular network or from external short message entities (e.g. web-based SMS portals). The messages are processed, stored in the SMSC queue, and delivered to the destination devices through a control channel. Messages in transit are encrypted by the network providers.

The MMS system centers on the Multimedia Message Service Center (MMSC). However, phones do not communicate with the MMSC directly but through a WAP gateway push proxy. Upon the arrival of MMS messages, the MMSC notifies the receiving clients with WAP push notifications over SMS. In response, the clients create a TCP/IP connection to retrieve the messages from the MMSC through the WAP gateway push proxy. Clearly, the security of the WAP push notification delivery is based on SMS security. The accompanying MMS message retrieval can be secured using SSL/TLS. More information about the structure of the cellular network and its security is available from Traynor et al. [50].

While SMS and MMS notifications are encrypted, there are still several security issues. GSM encryption is provided only over the radio interface since it is assumed that the SS7 network is inaccessible to external entities. The network providers do not always encrypt SMS messages [20]. Even if they did, though, the employed A5-family encryption algorithms have been compromised: a full rainbow table for the A5/1 cipher has been published [6], while an attack against the KASUMI cipher that is the basis for the new A5/3 cryptosystem can be performed in less than two hours on a PC [14]. Most importantly, GSM encryption does not give end-to-end security because the encryption key is shared between the mobile devices and the network providers, not directly between the content source and receiving device.

3.3 On-Platform Document Access

Access to documents that arrive on the phone is contingent on their method of delivery. Figure 2(ii.) demonstrates how various document types are handled as they arrive at the platform. There are three cases that we consider:

1. **Initial Document Recipients:** These applications either receive documents directly from the platform or from system applications. Their access will be dependent on permission labels set within their manifest files.

2. **Documents at Rest:** Some documents such as SMS, MMS, and the attachments of the emails received will be stored by the phone platform. In Android, these documents will be stored in Content Provider components, which act as databases for this content. Access to these Content Providers to either read or write data is also contingent on permissions in the application's manifest file.

3. **Document Sharing:** *Indirect receivers* are applications receiving documents from other applications. In Android, the APIs allowing interaction and data sharing between applications are mediated by the *Binder* IPC mechanism. Permissions are placed on application components to limit the data that can be sent and received between applications. This acts as a weak method of enforcing information flow enforcement, but is not secure as there is no concept of partial ordering with permission labels such that a lattice may be derived [13].

3.4 Threat and Trust Model

We assume that *the network is untrusted:* an adversary is capable of subverting any communications received from the network interface, regardless of whether the cellular network or the Internet were transited. In addition, any user applications on the phones are assumed to be untrustworthy unless otherwise identified by a sender. Our trusted computing base (TCB) comprises the underlying Linux operating system and the Android middleware itself, as well as system applications.

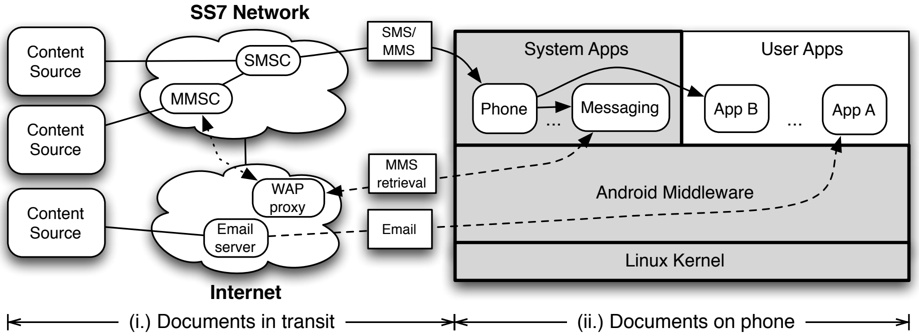


Figure 2: External content providers deliver their documents to the application(s) on the phone. The shaded area represents our TCB.

4. ARCHITECTURE

Porscha enforces security policy in two phases: initially as the content is transmitted over telephony networks and the Internet (as content in transit), and thereafter as it is processed and stored on the phone (on platform). This section describes how Porscha defines and enforces policy in each of these phases.

4.1 Supported Policy

Porscha extends the XML OMA policy [37, 39] to embrace the new DRM requirements identified in our application study, discussed in Section 2.1. All policies are mandatory. That is, any OMA client receiving a Porscha policy must implement the extensions or deny access. Detailed below, policy is encoded in XML as either a section in the text part of MMS or email attachment (see Section 4.3). Porscha supports[4]:

Constraints on Devices – OMA DRM 1.0 binds the content to devices that have acquired the proper license (i.e., right object). OMA DRM 2.0 also supports binding to specific devices identified by the users' International Mobile Subscriber Identity (IMSI) or WAP Identify Module (WIM). Described below, we extend identity to the phone number of the device itself (as regulated by the cellular provider).

Constraints on Applications – Porscha extends OMA DRM to constrain applications consuming content. Here the senders can specify access be restricted to applications with a given code fingerprint (hash of the application image), that are signed with a given developer key, or require that the application be configured with a given set of permissions (this last policy is similar to those found in the Saint system [36]).

Constraints on Use – Common policies codified in OMA DRM, such as validity period and number of uses, are supported. We extend these to support not only the regulation of simple accesses, but also differentiation of simple access from read, modify and delete rights.

4.2 Content-in-Transit

Porscha must secure content delivery over the untrusted networks–namely the *confidentiality*, *integrity*, and *authenticity* of SMS, MMS, and email must be preserved. Porscha uses identity-based encryption (IBE) [11] to ensure these properties. IBE enables the senders to construct the public keys of the recipients from known identities (phone numbers or email addresses), thus eliminating the need for *a priori* key distribution.

We briefly review the structure and use of IBE. Identity-based Encryption systems form a subclass [11] of public key cryptosystems. As with all public key systems, participants (users or other entities) are assigned a public key (which is widely distributed) and a private key (which is kept secret by the participant). What differentiates IBE from other kids of public key systems is the public key itself. An IBE public key is an arbitrary string such as an email address, name, social security number or any other value that is desirable for the target environments. Serving a similar role to a CA in traditional PKI systems, each IBE system contains a trusted private key generator (PKG). The PKG generates private keys using system wide secrets and provides them to the participants through a registration process. The PKG advertises public cryptosystem parameters for the IBE instance.[5] Encryption using the public key is performed by inputting the message (data), public key string, and cryptosystem parameters into the IBE encryption algorithm. Decryption is performed by inputting the ciphertext and private key to the decryption algorithm. Again, as in normal public key systems, it is also possible to encrypt using the private key and decrypt using public key (e.g., as used in creating digital signatures).

We use the following notation below. We denote the private key generator PKG, sender (content source) S, and receiver (phone) R. The identity for participant a is denoted I_a, the public/private key pair for a is K_a^+ and K_a^- respectively, content is m, and a policy for m is p_m. Encryption and signature operations are denoted $E(d,k)$ and $Sign(d,k)$, where d is the input data and k is the key. We denote the one-time time ephemeral key used in the delivery of email as k_e.

SMS/MMS – In SMS/MMS delivery, the recipient's telephone number (MSISDN) is used as their public key identity. Each cellular provider runs a Private Key Generator (PKG) that publishes the IBE public parameters via publicly ver-

[4] For brevity, we omit our XML structure and example polices beyond that identified in Table 1. Note that we are able to encode all policies uncovered in the application study using OMA and the Porscha extensions.

[5] The participant registration process, the secure acquisition process, and other key management services (e.g., revocation) have been discussed at length in other works [19, 31, 8]. Such issues are outside the scope of this work.

ifiable medium, e.g., in a Verisign certificate. The phone's private key and IBE parameters are loaded at subscription time in the phone SIM (see Section 6).

To send SMS/MMS, the sender encrypts its ID, the content, and policy using the receiver's public key. The sender then signs the resulting ciphertext his/her own private key K^-_{sender}. More precisely:

$$S \rightarrow R : E(\{I_S||m||p_m\}, K^+_R)|| \\ Sign(E(\{I_S||m||p_m\}, K^+_R), K^-_S)$$

If the sender uses different PKG than the receiver (e.g., subscribes to a different provider), it contacts the receiver's PKG directly using Internet connection or through multi-domain key management service supported by mobile systems [48, 22, 51]. Note that the addition of the policy, padding, and signature can increase the size of an SMS message beyond that supported by current networks (160 characters). Thus, as described in the following section, we convert SMS messages into MMS messages.

Email – We use a recipient's email address as the target identity. PKGs run in support of email domains publish the public key parameters through extended attributes on DNSsec [27] MX records. Emails can be vary in size and be arbitrarily large. As performing IBE encryption on large content potentially incurs high overhead, the sender encrypts the body of the email using the one-time 128-bit AES symmetric key k_e, which is obtained through Diffie-Hellman key exchange protocol. The symmetric key k_e is in turn encrypted with the receiver's IBE public key. The ciphertext is then signed, and the entirety is sent to the receiver as a MIME encoded email, as follows:

$$S \rightarrow R : E(\{I_S||m||p_m\}, k_e)||E(\{k_e\}, K^+_R)|| \\ Sign(\{E(\{I_S||m||p_m\}, k_e)||E(\{k_e\}, K^+_R)\}, K^-_S)$$

The following section details how these documents are process once they are received by the phone.

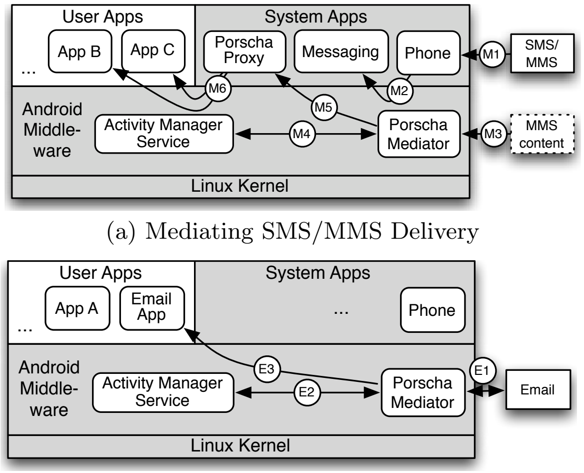
4.3 On-platform Policy Enforcement

When a document arrives at the designated phone, Porscha enforces the content security policy specified by the content source. Detailed in this section, the *Porscha mediator* enforces policy through a set of protocol proxies and authorization hooks within the Android middleware.

4.3.1 Policy Enforcement on Initial Recipients

SMS/MMS – Depicted in Figure 3(a), SMS and MMS PDUs (the base structure for data messaging services in cellular networks) initially arrive at the Phone application (M1 in Figure 3(a)). Under normal (non-Porscha) circumstances, the Android SMS Dispatcher delivers these messages to all applications that have registered for `WAP_PUSH_RECEIVED` Intents (which requires the `RECEIVE_MMS` permission). The Intent identifies the URI to the MMS content. Messaging application then downloads the MMS content and stores it in the MMS Provider inside of the middleware. The MMS Provider broadcasts another Intent when the content is stored.

To enforce policy, Porscha delays the initial `WAP_PUSH_RECIEVED` broadcast, but automatically triggers the Messaging application to download the content (M2). Once the content download completes, a second mediator hook parses the PDU (M3), extracts the policy from the content, and determines, in conjunction with the Activity Manager Service (M4), which applications are allowed to receive the document. Applications compliant with the policy receive the subsequent notifications, while those not in compliance will not.



(a) Mediating SMS/MMS Delivery

(b) Mediating Email Delivery

Figure 3: The Porscha mediator implemented inside Android intercepts the document delivery and enforces the policy on initial recipients.

Note that the Intent notifying the arrival of SMS/MMS cannot be issued from within the Android middleware: the broadcasting entity must be an application that possesses `BROADCAST_SMS` permission for SMS and `BROADCAST_WAP_PUSH` for MMS. Additionally, applications must be signed by the platform key for these permissions to be granted. To address this problem, we implement the *Porscha Proxy* as a system application signed and built with the platform. It receives from the Porscha mediator the list of the applications allowed to receive the document, the document content, the document metadata, and whether the document should be dispatched as SMS or MMS (M5). If the document should be dispatched as SMS, the Porscha Proxy constructs an SMS PDU and broadcasts an `SMS_RECEIVED` Intent to authorized applications (M6). If the document is dispatched as MMS, the proxy broadcasts a `WAP_PUSH_RECEIVED` Intent, containing the URI to the MMS content which is accessible through the MMS Content Provider, to authorized applications.

Email – Email traffic is opaque to Android: email client applications use application level protocols such as POP [25], IMAP [26], or Active Sync [33] to communicate with remote mail servers. For this reason, Porscha must "shim" email traffic by creating transparent proxies. The email enforcement intercepts email traffic at the network level through an SSL socket (E1). This mechanism operates at the middleware level inside of our TCB which is inaccessible to the applications. Messages are intercepted and interpreted within each proxy and policy enforced. We use the Apache Mime4j library [7] to parse the e-mail message streams in plain RFC-882 and MIME formats. For each email, the XML attachments are examined. If the attachment is recognized as content policy, the Porscha mediator coordinates with Activity Manager Service to enforce the policy (E2). The content may only be retrieved by an email client if it satisfies the policy (E3).

For usability, rather that filter email from applications that fail policy, we chose to mask its content. In such cases, Porscha removes all information from the email's header and body, and replaces these fields with the string *Hidden*. We will extend policy schema in the future to allow the sender to provide "alternate text" that would instruct the user where to go to obtain an appropriate application or license for the received content in the event the accessing phone/application does not satisfy the policy.

Note that while controlling access to DRM-protected documents, Porscha allows unprotected documents to be received without restriction. These documents are not IBE encrypted by Porscha. Thus, they can be accessed by the receiving applications.

4.3.2 Policy Enforcement on Documents at Rest

By default, Android stores the SMS, MMS, and email attachments with the system applications using Content Provider components. Applications with permissions to access (read or write) these Content Providers can access this content even if they are not the initial recipients. To allow external senders to control access to the documents delivered from them, we add an extra *policy* field to the structure of each Content Provider record. The Porscha mediator inserts the policy (if available) into this field, and when the Content Provider record is accessed the corresponding policy is checked, and access allowed or denied based on the compliance of the caller application with the policy.

4.3.3 Enforcement on Indirect Receivers

Porscha mediates passing of data between applications as shown in Figure 4.3.3 and as follows:

Intent – The Porscha mediator acts as a reference monitor for Intents that pass protected content. The sending application binds the policy with the Intent that encompasses the content. The mediator prevents applications not satisfying the policy from receiving the Intent.

Content Sharing – Inclusion of a *policy* field into Content Provider records, as described above, allows the Porscha mediator to ensure that every access to stored content satisfies the attached policy. Access is mediated through the Content Resolver mediation hook.

Inter-application API calls – When an application API is called by another application (e.g. Service call), we bind a policy to the input parameter or return value containing content delivered in an Android's *parcel* object. The mediator interprets the parcel, and enforces the policy. If the policy fails (on either the call or return), a security exception is thrown.

5. EVALUATION

This section briefly evaluates the costs of policy enforcement in Porscha. All experiments were executed on a HTC G1 Dream smartphone over T-Mobile 3G services. Porscha was built on the Cyanogen [1] Android 2.1 firmware build and installed on the phone, as was the Stanford IBE library V.0.7.2 (a C implementation of Boneh-Franklin IBE [11]). The IBE module was crossed compiled for the ARM processor as a native executable. Each experiment was repeated 10 times and the average reported (with negligible observed sample variance).

Highlighted in Table 3, an initial set of experiments sought to measure the overheads associated with SMS processing. Here we measured the time between the arrival of the PDU and the time it is dispatched to consuming applications. The experiments showed that SMS processing time is less than 0.1 seconds in unmodified Android. SMS documents are delivered as MMS introducing an additional 4 or greater second overhead. Microbenchmarking of SMS processing revealed three central underlying costs: MMS push notification handling ($\approx$ 1.03 s), MMS content retrieval ($\approx$ 1.44 s), and other connection management processing ($\approx$ 1.04 s). The lower costs associated with SMS-over-MMS vs. MMS with media content were associated with the reduced size of the objects being downloaded (SMS policy objects were on the order of 100s of bytes versus 18kb `.jpg` objects in MMS experiments). The maximum observed overhead for IBE was about 480 msec.

For MMS, we measure the latency from the arrival of the PDU to the time to MMS content is completely downloaded and applications notified. Without IBE, Porscha incurs about a 4% overhead ($\approx$ 20 msec). IBE adds about 1 s. to the overhead–significantly more than in SMS. Here again the cause is the size of the encrypted media: the `.jpg` object. Note that recent advances in IBE offer run-time improvements that can reduce these overheads by as much as 20-35% [23], and techniques such as the use of one time symmetric keys (as detailed in section 4.2) may substantially reduce these costs.

The overhead of processing email ranges from 0.7 seconds to just over 1 second, depending on the email access protocols. These costs are largely due to the proxying of the access protocols, SSL, buffering, email message reconstruction, and policy extraction and evaluation (if policy is available). Note that we have not yet implemented IBE for email, but the experiments above suggest that overheads will be manageable.

6. DISCUSSION

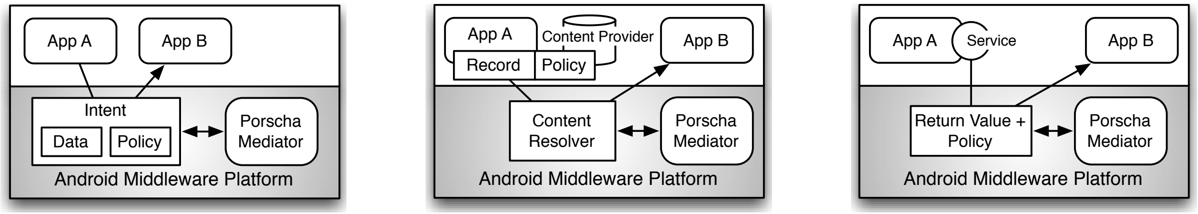
This section examines the security guarantees provided by Porscha and potential threats to these guarantees.

6.1 Protecting the Private Key

Porscha's model for key distribution involves the client phone receiving a private key from the phone provider. Recall from Section 4.2 that a private key generator (PKG) is trusted to create IBE keys and that a phone's MSISDN can be used as a public key identity. The cellular provider will operate the PKG and provide the private key at subscription time on the client's SIM card. However, the SIM is merely a memory card, and is susceptible to being stolen or lost. Therefore, we use a shared secret between the provider and client phone to encrypt the stored private key, with knowledge of the secret required to unlock the key package. One way of communicating this to the user would be with on a slip of paper or a similar out-of-band method when the SIM is purchased or reprogrammed. This method is already used for the SIM authentication key, stored by the provider in their authentication center (AUC).

6.2 Recipients Without Porscha

External senders do not have prior knowledge about whether Porscha is available on client phones. In the absence of



(a) Mediate Intent Passing

(b) Mediate Content Provider Access

(c) Mediate Service Call

Figure 4: Porscha Mediator intercepts cross-application content passing

	Original Latency	Overhead from Porscha without IBE		Overhead from Porscha with IBE	
		policy passed	policy failed	policy passed	policy failed
SMS	0.083	4.07	4.12	4.57	4.56
MMS	5.16	0.22	0.21	1.52	1.53
POP3	6.34	0.68	0.91	2.51	2.61
IMAP	3.79	1.02	1.02	2.94	2.85
Active Sync	3.38	0.8	0.85	2.18	2.19

Table 3: Overheads in processing SMS, MMS, and Email (in seconds).

Porscha, the clients would not be able to access protected documents. This is reasonable, as any content delivered to and intended for the phone should remain opaque to the user. As a result, whether it be phones that do not have a Porscha framework installed or another means of accessing content, e.g., retrieving emails on a computer, content protected by Porscha should and will be inaccessible by these entities.

Note that emails accessed by the IMAP protocol are ultimately managed by an IMAP server; thus, any modifications made to an email by Porscha may be reflected on other clients. To resolve this issue, we store all modifications, such as decrypted emails and those with information removed, locally on the phone, and only reflect back to the IMAP server the original email that was sent to the phone - thus, an original copy of the email is always maintained.

6.3 Application and Platform Trust

With Porscha, we are making assumptions of trust in the Android middleware and associated system applications. There are a number of reasons why this level of trust can be considered appropriate. First, Android applications are signed with a certificate whose private key is held by the system developer; this provides a means of ensuring that the application's integrity is intact and that the origin of the code is as expected. Tools such as Kirin [16] allow install-time certification of applications against potentially dangerous functionality.

Android is middleware that runs on a Linux kernel. Several methods of ensuring kernel integrity have been considered, and this is an area of ongoing research. These include run-time monitors such as the Linux kernel integrity monitor (LKIM) [32]. Ensuring that the phone platform itself is booted into a trustworthy state has also been an area of considerable focus. One promising solution is to include trusted platform modules (TPMs) [49] inside mobile phones; specifically, the *Mobile Trusted Module* (MTM) initiative [15] has considered a TPM-like device that adds functionality for secure boot, which enforces integrity protection of the underlying firmware and system state, and allows for continual assurance of boot-time guarantees through use of the Linux Integrity Measurement Architecture (IMA) [45].

6.4 Alternative Application Enforcement Infrastructures

The extent to which Porscha protects a document largely depends on the attached policies. The senders can indicate the target applications by different degrees of specification from unique application package hashes to loosely defined application properties.

An even stronger security model can be implemented as a performance trade-off. For example, Porscha can be used along with Saint [36] which regulates application interactions (but does not examine the content being passed). As a result, we would gain a more comprehensive view of application behavior and could ensure that all applications are monitored for sharing violations.

Adoption of more heavyweight mechanisms offering continual content monitoring, such as dynamic taint analysis [12, 52], is also possible. However, these systems are not designed for information with semantically rich policy attached. The policies for incoming documents are mostly unique. Managing large and highly dynamic set of taint markings (e.g. in taint analysis) can thus be burdensome. Porscha's content enforcement mechanism is comparatively quite lightweight.

6.5 Digital Rights Management

DRM has been contentiously discussed for nearly 15 years. Such controversy stems from the primary application of DRM: to restrict the use of digital content and prevent piracy, ostensibly to preserve artistic integrity and protect revenue streams for content creators. For example, three competing DRM technologies for mobile or portable devices are Apple's FairPlay [29], Microsoft PlayReady [34]. Along with OMA DRM, all aim to limit media usage for commercial purposes. In Android, OMA DRM 1.0 is supported to manage ringtones, MMS, and pictures, preventing users for forwarding these documents. An external generic framework for DRM implementation is also available but is not used by the official platform. An external framework containing the Open-

Core Content Policy Manager (CPM) [40] is also available. CPM does not implement any DRM algorithms or protocols, but acts as an aggregator with interfaces for authentication, authorization, and access control. Plugins are available for multiple DRM agents, such as WMDRM [42] and DivX [41].

There has been significant opposition to DRM [17, 2] with detractors viewing it as a means for limiting consumer rights and eliminating "fair use" provisions. However, DRM is by definition a generic term for access control technology that secures content and limits its distribution [10, 24]. We consider DRM's role in Porscha strictly as a means to providing content-based access control without comment on business, legal, or philosophical issues. We differentiate from existing DRM schemes, however, and provide a superset of functionality by preserving confidentiality, integrity, and availability, not merely employing encryption and licensing as with typical DRM implementations. In addition, Porscha is lightweight and designed with mobile solutions in mind; by contrast, many advanced DRM protocols are heavyweight and not transparent to applications.

7. RELATED WORK

Mobile phone security often involves regulating the behavior of individual applications installed on the phone to protect the platform. As permissions requested by Android applications reflect their capabilities, Kirin [16] prevents installation of malware by identifying potentially dangerous applications based on these permissions. By contrast, Saint [36] modifies the Android middleware to enforce application policies which regulate how application permissions are granted and how applications interact with each other. While Saint concentrates on securing communication endpoints, Porscha concentrates on the actual content passed.

Enhancing mobile phone security through mandatory access control (MAC) and trusted hardware, specifically Security-Enhanced Linux (SELinux) [47] and TPMs, is a means of protecting application and platform integrity. SELinux security policy has been applied to ensure the integrity of the Openmoko phone platform and trusted applications [35]. Additionally, Rao and Jaeger [44] developed an SELinux-based MAC system that considers input from multiple stakeholders to develop policies for controlling application permissions. Recently, Shabtai et al. [46] have ported SELinux to Android and enabled security policy for enhancing the protection of system processes. Unlike Porscha, which enforces MAC policies to secure documents arriving at the phones, these approaches focus strictly on platform security; while they are orthogonal to our concerns, platform trustworthiness will increase the security of all overlying layers above, including Porscha.

Several IBE solutions have been proposed for use with mobile phones. Mobile phone numbers are commonly used as client identities because they can be effortlessly authenticated by the network. Communication among different network providers running their own PKGs is a major challenge for IBE implementation; proposed solutions have included the use of hierarchical IBE [22, 51] and cross-domain key extensions [48] .

8. CONCLUSION

This paper has proposed Porscha, a content protection framework for Android that enables content sources to express security policies to ensure that documents are sent to targeted phones, processed by endorsed applications, and handled in intended ways. Through a study of real-world applications, we formed an initial scope of appropriate content policies, and we demonstrated how these may be used in Porscha to protect SMS, MMS, and email documents. Porscha secures content delivery using identity-based encryption and mediates on-platform content handling to ensure conformance with content policy. Future work will examine additional types of content that may be protected by Porscha and the policy implications of managing this content.

9. REFERENCES

[1] Android Community ROM. `http://www.cyanogenmod.com/`, March 2010.

[2] I hate DRM: A site dedicated to reclaiming consumer digital rights. `http://ihatedrm.com`, June 2010.

[3] Mobile Watchdog. `http://www.mymobilewatchdog.com/`, January 2010.

[4] SMS Trap. `http://www.smstrap.com/`, January 2010.

[5] Stealth SMS. `http://stealthsms.trusters.com/s_features.htm`, January 2010.

[6] A5/1 Security Project. Creating A5/1 Rainbow Tables. `http://reflextor.com/trac/a51`, 2009.

[7] Apache Software Foundation. Apache James Mime4j. `http://james.apache.org/mime4j/`, March 2010.

[8] G. Appenzeller, L. Martin, and M. Schertler. Identity-Based Encryption Architecture and Supporting Data Structures, Jan. 2009. IETF RFC 5408.

[9] L. Bauer, S. Garriss, J. M. Mccune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the grey system. In *Proceedings of the 8th Information Security Conference (ISC'05)*, pages 431–445, 2005.

[10] E. Becker, W. Buhse, D. Günnewig, and N. Rump, editors. *Digital Rights Management Technological, Economic, Legal and Political Aspects*. Springer, 1 edition, 2003.

[11] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Proceedings of CRYPTO*, 2001.

[12] J. Clause, W. Li, and A. Orso. Dytan: A Generic Dynamic Taint Analysis Framework. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA)*, pages 196–206, 2007.

[13] D. E. Denning. A Lattice Model of Secure Information Flow. *Commun. ACM*, 19(5):236–243, May 1976.

[14] O. Dunkelman, N. Keller, and A. Shamir. A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony. In *Proceedings of the 30th Annual Cryptology Conference (CRYPTO 2010)*, 2010.

[15] J.-E. Ekberg and M. Kyläanpää. Mobile Trusted Module (MTM) - An Introduction. Technical Report NRC-TR-2007-015, Nokia Research Center, Helsinki, Finland, Nov. 2007.

[16] W. Enck, M. Ongtang, and P. McDaniel. On Lightweight Mobile Phone Application Certification. In *Proceedings of ACM CCS*, November 2009.

[17] Free Software Foundation, Inc. The Campaign to Eliminate DRM. http://www.defectivebydesign.org/, June 2010.

[18] Gartner. Gartner Says Worldwide Mobile Phone Sales to End Users Grew 8 Per Cent in Fourth Quarter 2009; Market Remained Flat in 2009. http://www.gartner.com/it/page.jsp?id=1306513, Feb. 2010.

[19] C. Gentry. Certificate-Based Encryption and the Certificate-Revocation Problem. *Advances in Cryptology*, 2656, January 2003.

[20] M. Gholami, S. M. Hashemi, and M. Teshnelab. A Framework for Secure Message Transmission Using SMS-Based VPN. *Research and Practical Issues of Enterprise Information Systems II*, 1:503–511, 2008.

[21] GigaOm. The Apple App Store Economy. http://gigaom.com/2010/01/12/the-apple-app-store-economy, Jan. 2010.

[22] J. Horwitz and B. Lynn. Toward Hierarchical Identity-Based Encryption. In *Proceedings of EUROCRYPT '02*, pages 466–481, London, UK, 2002. Springer-Verlag.

[23] J.-S. Hwu, R.-J. Chen, and Y.-B. Lin. An Efficient Identity-Based Cryptosystem for End-to-End Mobile Security. *IEEE Trans. Wireless Comm.*, 5(9):2586–2593, September 2006.

[24] R. Iannella. Digital Rights Management (DRM) Architectures. *D-Lib Magazine*, 7(6), 2001.

[25] IETF Network Working Group. Post Office Protocol - Version 3. http://www.ietf.org/rfc/rfc1939.txt, May 1996.

[26] IETF Network Working Group. Internet Message Access Protocol - Version 4, rev1. http://www.ietf.org/rfc/rfc1939.txt, March 2003.

[27] IETF Network Working Group. DNS Security Introduction and Requirements. http://www.ietf.org/rfc/rfc4033.txt, March 2005.

[28] ITU. Measuring the Information Society. http://www.itu.int/ITU-D/ict/publications/idi/2010/index.html, 2010.

[29] S. Jobs. Thoughts on Music. http://www.apple.com/hotnews/thoughtsonmusic/, February 2007.

[30] M. Kirkpatrick and E. Bertino. Enforcing Spatial Constraints for Mobile RBAC Systems. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, 2010.

[31] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Secure Key Issuing in ID-based Cryptography. In *Proceedings of the ACSW Frontiers Workshop*, 2004.

[32] P. A. Loscocco, P. W. Wilson, J. A. Pendergrass, and C. D. McDonell. Linux Kernel Integrity Measurement Using Contextual Inspection. In *Proceedings of ACM STC*, 2007.

[33] Microsoft Corporation. ActiveSync HTTP Protocol Specification, version 6.0. http://msdn.microsoft.com/en-us/library/dd299446(EXCHG.80).aspx, May 2010.

[34] Microsoft Corporation. Microsoft PlayReady. http://www.microsoft.com/playready/default.mspx, June 2010.

[35] D. Muthukumaran, A. Sawani, J. Schiffman, B. M. Jung, and T. Jaeger. Measuring Integrity on Mobile Phone Systems. In *Proceedings of ACM SACMAT*, June 2008.

[36] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application-Centric Security in Android. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, December 2009.

[37] Open Mobile Alliance Ltd. Rights Expression Language Version 1.0. Technical Report OMA-Download-DRMREL-V1_0-20040615-A, Open Mobile Alliance, June 2004.

[38] Open Mobile Alliance Ltd. DRM Architecture 2.0.1. Technical Report OMA-AD-DRM-V2_0_1-20080226-A, Open Mobile Alliance, February 2008.

[39] Open Mobile Alliance Ltd. DRM Rights Expression Language Version 2.0.2. Technical Report OMA-TS-DRM_REL-V2_0_2-20080723-A, Open Mobile Alliance, July 2008.

[40] PacketVideo Corporation. Content Policy Manager Developer's Guide OHA 1.0 r.1. November 2008.

[41] PacketVideo Corporation. PV Android DivX Premium Package. July 2009.

[42] PacketVideo Corporation. PV Android Windows Media Package. November 2009.

[43] Portio Research. Mobile Messaging Futures 2010-2014: Analysis and Growth Forecsts for Mobile Messaging Markets Worldwide, 2010.

[44] V. Rao and T. Jaeger. Dynamic Mandatory Access Control for Multiple Stakeholders. In *Proceedings of ACM SACMAT*, June 2009.

[45] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, Aug. 2004.

[46] A. Shabtai, Y. Fledel, and Y. Elovici. Securing Android-Powered Mobile Devices Using SELinux. *IEEE Security and Privacy*, 8:36–44, 2010.

[47] S. Smalley, C. Vance, and W. Salamon. Implementing SELinux as a Linux Security Module. Technical Report 01-043, NAI Labs, 2001.

[48] M. Smith, C. Schridde, B. Agel, and B. Freisleben. Securing Mobile Phone Calls with Identity-Based Cryptography. *LNCS: Advances in Information Security and Assurance*, 5576:210–222, June 2009.

[49] TCG. *TPM Main: Part 1 - Design Principles.* Specification Version 1.2, Level 2 Revision 103. 2007.

[50] P. Traynor, P. McDaniel, and T. La Porta. *Security for Telecommunications Networks.* Advances in Information Security. Springer, July 2008.

[51] Z. Wan, K. Ren, and B. Preneel. A Secure Privacy-Preserving Roaming Protocol Based on Hierarchical Identity-Based Encryption for Mobile Networks. In *Proceedings of ACM WiSec*, 2008.

[52] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing System-Wide Information Flow for Malware Detection and Analysis. In *Proceedings of ACM CCS*, 2007.