A Personal Mobile DRM Manager for Smart-Phones

Siddharth Bhatt
Stony Brook Network Security and Applied Cryptography Lab
Computer Science, Stony Brook University
Stony Brook, NY 11794
sid@sidbb.com

Radu Sion[1]
Stony Brook Network Security and Applied Cryptography Lab
Computer Science, Stony Brook University
Stony Brook, NY 11794
sion@cs.stonybrook.edu
phone: (631) 731 - 1695
fax: (631) 632 - 1690

Bogdan Carbunar
Applied Research and Technology Center
Motorola Labs, 1295 E. Algonquin Rd. IL05
Schaumburg, IL 60195
carbunar@motorola.com

Abstract

In this paper we report on our experience in building the experimental Personal Digital Rights Manager for Motorola smart phones, an industry first.

Digital Rights Management allows producers or owners of digital content to control the manner in which the content is consumed. This may range from simply preventing duplication to finer access policies such as restricting who can use the content, on what devices, and for how long. In most commercial DRM systems, the average end user plays the role of content consumer, using DRM protected content made available by a service. Here we present a personal digital rights system for mobile devices where the end user has the ability to place DRM protection and controls on his or her own personal content.

We designed the personal DRM system to allow users of a mobile device to transparently define controls and generate licenses on custom content and securely transfer them to other mobile devices. A user is able to define and restrict the intended audience and ensure expiration of the content as desired. Compatible devices automatically detect each other and exchange credentials. The personal DRM system on each device safely enforces the content

[1]Supported by Motorola, NSF (IIS-0803197, CNS-0627554,0716608,0708025), IBM, Xerox, CEWIT.

usage rules and also handles moving licenses between devices while preventing leakage of content. We implemented a prototype of our system on Motorola E680i smartphones.

Key words: Digital Rights Management, Mobile Devices, Cellular, Smartphones

1. Introduction

Digital Rights Management (DRM) refers to a collection of technologies used to control access to digital data. It is generally used by copyright owners or publishers of digital content to specify who can access the data and in what manner. This is accomplished by specifying the rights that the user of the content has, and the restrictions on the consumption of the content.

For instance, an online movie rental service that lets users download video files to a personal computer for a certain period uses DRM protection to enforce the terms of the rental license. Thus, even though the downloaded file is on the user's computer and may be seen as being under the "user's control", DRM is used to lock the file and essentially make it difficult to play after the expiration of the rental period.

DRM is also used in mobile devices where, driven by huge advances in network infrastructure support as well as a boom in the number of interconnected personal devices, the modern mobile customer experience has become increasingly compelling. A large set of novel communication and content streaming services have become accessible, ranging from simple SMS messaging to live TV broadcasts. This trend is likely to accelerate with the eventual introduction of fully packet switched 4G networks featuring increased bandwidth and global mobility.

Mobile devices are increasingly being used as personal media centers. Traditional boundaries between the roles of information "producer" and "consumer" are blurring as device users produce and distribute content such as personal pictures and videos on their mobile devices. In such a scenario, it makes sense to have technology that lets the individual user retain control over the dissemination of such personal content. It is essential to enable user-level DRM controls for content access, data integrity and rights management.

Currently, most DRM technologies are geared toward the end user being the content consumer, and the DRM protected content coming from a commercial service. The model can be viewed as analogous to a client-server system, with the end user being the client that consumes the content provided by the server, e.g. an online music store. In contrast, we see the need for a personal DRM system that can be viewed as analogous to a peer-to-peer model, with every end-user being able to act as content producer, and assert rights and retain control of his or her own content. Any given device can act as both sender (content producer) and recipient (content consumer).

In this paper we discuss a new personal digital rights management system for mobile devices in which a user is able to transparently define, generate, package and migrate content licenses between mobile devices on-demand. Our system lets a user select a certain piece of content on a mobile device and associate various controls with it and specify who can

consume the content. The controls may include restrictions such as how many times the content can be played etc, as well as whether the content may be exported to other devices.

We designed the personal DRM system and developed a prototype on the Motorola E680i smartphone. Each device automatically detects other personal DRM enabled devices in its proximity and exchanges credentials with them. It presents the user with an efficient graphical user interface that can be used to specify restrictions on any digital content on the phone, and lets the user select a target device to send the content to. Files are transferred between devices using a secure transfer protocol. Each device can play any content it is authorized to, and safely enforces the controls associated with the protected content.

2. Background

The core entities in a Digital Rights Management system are the users, the content and the rights. A DRM system design models these entities and the relationships between them, as shown in Figure 2. A DRM system can also be modeled by more complex relationships between finer-grained entities, as described in [14]. For instance, the "rights" entity can be broken down into usage rules which include the obligations of the rights holder, such as paying for use, and permissions and constraints. Permissions specify *what* the user can do with the content, and constraints specify *how*. For instance, permissions may specify that the user can play or print the content. The constraints on the play permission may specify that the content can be played between certain dates. The constraint on the print permission may specify how many times the content can be printed.

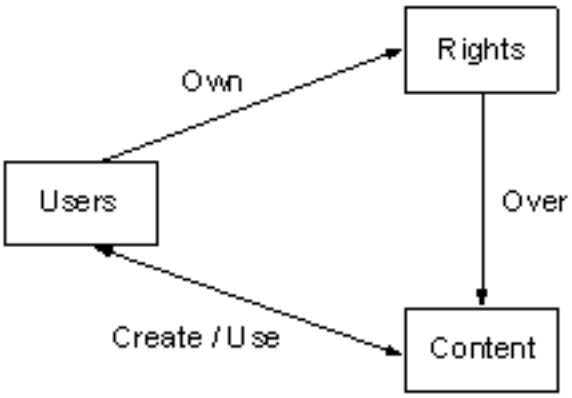


Figure 1: Core Entities in a DRM System

In terms of actual implementations, Park et al. [23] describe and compare various security architectures for use in dissemination, control and tracking of digital content, such as encapsulated or independently distributed access rights, online or offline access, virtual machine based architectures etc.

In most such traditional DRM models, all devices are equally trustworthy and have discretionary control over all protected content. Popescu et al. [24] suggest a deviation

from this paradigm and propose a multilevel security policy based model that differentiates between devices based on their types and tamper-resistance properties. For instance, a device that has high tamper resistance would be restricted from sending protected content to a device with low tamper resistance, or a device would not be allowed to send protected video content to an audio device.

Trusted computing [38] is increasingly being touted as an ideal platform for the implementation of digital rights management. In spirit, trusted computing suggests there should be mechanisms in place to enforce what a device can or cannot do. This is enforced by a hardware chip called the trusted platform module which provides tamper-resistant security functions. Several models of DRM on trusted platforms have been suggested [6, 9, 25]. Using tamper resistant trusted computing hardware would prevent attacks based on dismantling hardware and reading sensitive information from chips. The trusted computing base would be outside the operating system, and so it would be possible to use unmodified software and yet have strongly enforced DRM. Trusted hardware would also provide a component for providing a tamper resistant clock source which is crucial for DRM systems.

In this paper we propose a software based DRM for personal content generated on mobile devices, in particular a Motorola E680i phone. Our goals, of protecting and enforcing access control rules on content generated on mobile devices would certainly benefit from the existence of a trusted platform module. However, we believe the cost of such devices and the incentives attackers have to circumvent protection mechanisms on personal content are sufficient reasons for focusing solely on software based DRM techniques.

Implementing DRM on mobile devices can be particularly challenging because of the limited capabilities of the platform. Another drawback is that a user may have multiple personal mobile devices while most DRM schemes usually lock content to a specific device, thus restricting the user's ability to move the content between devices. Messerges and Dabish [21] describe the basic strategies for DRM on mobile phones, propose a framework for implementing DRM on embedded devices, and also propose a "family domain" approach allowing portability of content. The scheme lets a user specify all of his or her devices, and a trusted domain authority installs a private domain key on each device, allowing access to the content from all of these devices. Sheppad et al. [31] suggest a similar domain-based licensing scheme. Priyadarshini and Stamp [34] proposed the use of the resources of general purpose P2P networks (which also include mobile networks) to relieve the traffic and processing bottlenecks of centralized content distributors. Additionally, there currently exist a number of proprietary DRM schemes in mobile phones, developed by various device manufacturers and service providers. These are used to protect purchased content (e.g. ringtones) and streaming services, such as Verizon's V CAST [37] service. The Open Mobile Alliance, a standards body for the mobile phone industry, is attempting to replace proprietary schemes with an interoperable standardized digital rights management scheme [33].

Even outside the mobile industry, there have been several attempts to standardize certain key components of DRM systems, if not the entire system itself, as described in [27]. Safavi-Naini et al. [28] propose an import/export functionality by which content protected by one DRM scheme can be made available on a device that supports another DRM scheme

without compromising the original protection imposed on the content. They rely on an intermediary system to perform translation between schemes. Kravitz and Messerges [16] suggest a similar idea consisting of a "rights issuer module" situated in the network between two devices, performing translation between DRM schemes.

In practice, the two main aspects of most DRM schemes are "locking" the content to prevent it from being accessed directly, and specifying the rights and usage rules on the content. These rules specify the circumstances in which the content is allowed to be unlocked and rendered. The locking of the content is generally accomplished by encrypting the actual content file(s). The usage rules are specified in a file referred to as the license. The DRM software on a device is responsible for checking the license and enforcing the controls before decrypting the content [19].

The license generally specifies the user or device to whom rights are granted, the permissions granted (e.g. play, print, export) as well as constraints on those permissions, but the specifics of the license may vary based on the application and the service model. Gunter et al. [11] have developed a mathematical model and formal language to capture the meaning of a wide variety of licenses with precise semantics. In practice, licenses are generally a simple text-based file. One of the most popular formats is XrML [12], an XML-based format for digital rights licenses. There have also been attempts to create a more expressive and flexible license format, such as LicenseScript [4], a logic-based language for digital rights. It is based on multiset rewriting, uses Prolog programs for licenses and makes it possible to express a multitude of sophisticated usage patters clearly and precisely.

Among commercial applications, one of the early examples of DRM is the Content Scrambling System [5] used on commercial DVDs. It employs simple encryption to protect the content which could be decrypted only on licensed DVD players until the scheme was compromised. DRM has since been employed on various media such as audio CDs [13], online music stores [17] and e-books and has featured prominently in public discussion, especially in the context of the U.S. Digital Millennium Copyright Act which criminalizes the circumvention of DRM controls [29]. A unique DRM scheme for geospatial data has also been suggested [20].

On an average personal computer, the most common example of DRM technology is Windows Media DRM which is embedded in Microsoft's Windows Media Player and its associated components. It is used in protected content available from several online services. WMDRM uses elliptic curve cryptography for key exchange and a combination of a block and stream cipher to protect the content [18] and features an automatic license acquisition protocol. A version of Windows Media DRM also exists for portable audio devices. Similarly, Apple has its own DRM scheme, FairPlay [3], used in its iTunes music store and the iPod. In FairPlay, each content file is encrypted with a master key, which itself is stored encrypted along with the content file. The master key is encrypted with a unique user key for each user who buys the content, and a record of the user key and the user's unique machine identifier are stored on Apple's servers. The user keys are transferred to the user's computer along with the purchased content, and to any portable devices to which the content is copied. For more details on FairPlay, we recommend the work of Stamp and Venkataramu [35].

Finally, Microsoft's Zune portable audio player allows wireless sharing of files between devices, with a form of personal DRM added to shared files. However, Zune allows only one type of control on the content: a shared file stays valid for "3 days or 3 plays", whichever comes first [39] and cannot be forwarded to other devices. In contrast, our personal DRM system allows flexible user-defined controls and also allows licenses to be moved to other devices.

It is interesting to note that all major industry DRM schemes, including the ones described above have been showed to be vulnerable to attacks. This includes the Content Scrambling System [36], iTunes [2, 1], Kindle [32] and the multiple times hacked WMDRM [26, 10]. In fact, any pure software based approach, used by many existing DRM mechanisms, is inherently vulnerable to reverse engineering attacks.

Moreover, any DRM scheme is vulnerable to the "analog hole", where content is captured in analog format while being rendered. While outside the scope of this paper, we note that the quality of the analog copy will likely be below that of the original content.

3. Design

In broad overview, the personal DRM system lets a user lock a content file, associate certain controls with it and send it to another device. The recipient device allows the file to be unlocked and played only after enforcing the controls. Therefore, the design of the personal DRM system involves designing the following central aspects of the system.

- Securing i.e. encrypting the content
- Specifying the controls on the content i.e. the license
- Securely transferring the files between devices
- Checking and enforcing the license before rendering the content

Before detailing the design of these aspects for our system, we take a look at some of the important security requirements and assumptions.

3.1. Security Requirements

A number of security requirements and issues arise in the personal DRM system. These are discussed below.

3.1.1. Protected storage

The system requires that secret keys, state information about active licenses, temporarily decrypted files and other sensitive information be stored on disk. However, it is crucial to the security of the system that the device user be unable to access and modify or even read this protected content. Such content must be kept in a protected area of storage where only the DRM software running with elevated privileges has access. In other words, we need a file system with access control. The system must be implemented on a platform that provides such a feature. Naturally, it should also be ensured that the protected storage lies on an internal memory chip and not a removable memory card.

3.1.2. Memory separation

Apart from information on disk, there may also be sensitive information in memory while the DRM agent performs its tasks. It is crucial that another user process not be able to access the memory area of the DRM agent while it runs. Therefore, the system must be implemented on a platform that provides such memory separation and process isolation.

3.1.3. Trusted rendering software

Protected content which is usually encrypted is temporarily decrypted while it plays. The rendering software, e.g. a media player has access to this decrypted content. A malicious media player could be siphoning off the decrypted data and saving it elsewhere while it plays. We therefore need trusted rendering software that we can be confident of not performing unauthorized operations. One way to enforce this is to have the manufacturer of the mobile device digitally sign the rendering software. The DRM software on the device would decrypt the content and invoke the player only if it finds a valid signature for the player.

3.2. The DRM Agent

The DRM agent is the main process responsible for overseeing all the personal DRM related operations. It acts as an intermediary between the user-level applications and the device's operating system, as shown in Figure 3.2.

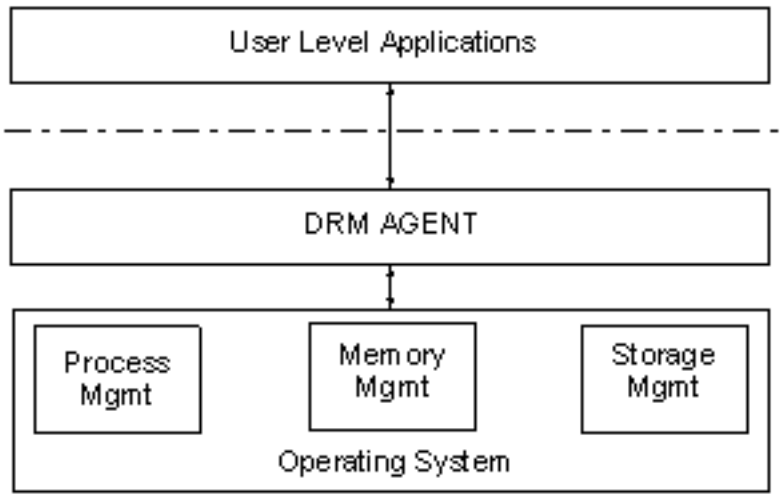


Figure 2: Operating System augmented with DRM Agent

The DRM agent runs with elevated privileges, like the operating system, and is a trusted system responsible for ensuring the security of the personal DRM system. All the tasks described in the following sections are performed by the DRM agent. One key requirement for our system is that only the operating system and the DRM agent have elevated privileges. All user-level applications including user-installed software run at lower privilege levels.

3.3. Securing the content

As seen in the previous section, the most common way to secure a content file in DRM schemes is to encrypt it. In our system, we encrypt the content file using a symmetric key algorithm.

However, using such a scheme requires that both the sender and the recipient share the encryption key, which is often impractical, since any given device will need to maintain a separate shared key for every other device it communicates with. Additionally, in our personal DRM scheme, we also have the requirement that the sender be able to specify who has access to the locked content. That is, if device A sends some protected content to device B, it must be ensured that the content will be playable on B only and no other device. These two factors motivate us to use a basic public key cryptosystem.

3.3.1. The public key cryptosystem

In the personal DRM system, every compatible device has a public/private key pair associated with it. The private key is a secret embedded in the device, inaccessible to the user of the device. The key is stored in a file on the file system, with permissions set to allow read-only access only to the DRM agent. The public key is present in the form of an X.509 [15] format public key certificate on the device. We use the public and private keys for encryption and decryption using the RSA public-key algorithm.

Let E and D denote encryption and decryption respectively with a symmetric key algorithm (we use RC4 in our implementation), and Pb and Pr denote the RSA operations of encrypting and decrypting with the public and private keys respectively. The securing of the content file now takes place as follows.

1. The sender generates a random symmetric key K. The content file is first encrypted as $E_K(File)$.
2. The random symmetric key itself is encrypted using the recipient's public key as $Pb_{Recipient}(K)$.
3. Both the encrypted content file and the encrypted key are sent to the recipient.
4. At the other end, the recipient device first decrypts the key using its private key.
$$Pr_{Recipient}(Pb_{Recipient}(K)) = K$$
5. Finally, having obtained the symmetric key K, the recipient then decrypts the content file.
$$D_K(E_K(File) = File$$

In this case, there is no need to maintain shared keys on any device. They can be randomly generated each time a content file needs to be encrypted. The encrypted key, i.e. $Pb_{Recipient}(K)$ can be embedded in the license which is sent to the recipient along with the protected content. We call this entire process the "packaging" of content. The process is shown in Figure 3.3.1.

Note that encrypting the content is a necessary but not sufficient condition for providing DRM protection. The use of encryption can thwart external attacks (e.g., from attackers sniffing other devices' transmissions) and insider attacks originating from casual users, lacking the means to recover private, locally stored keys.

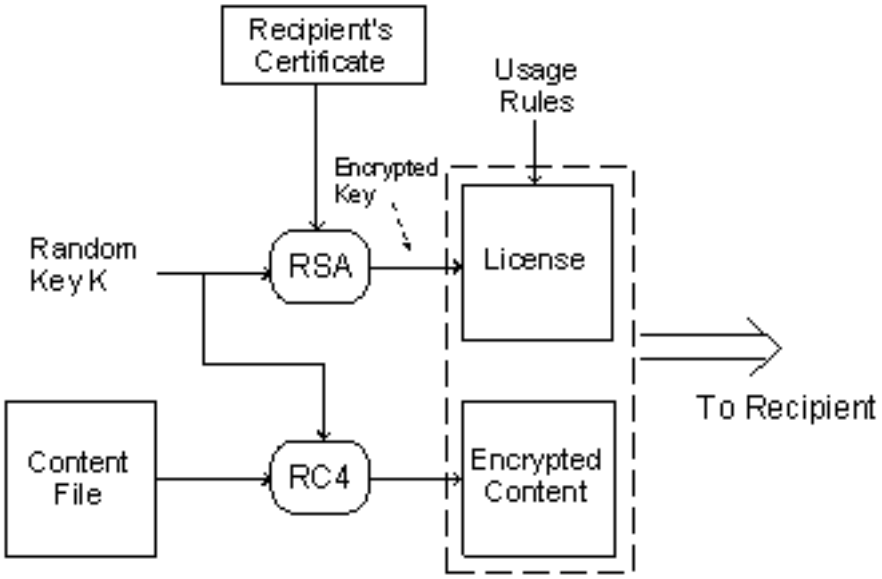


Figure 3: Content packaging

3.3.2. Obtaining certificates

Each device must obtain a signed certificate from a certificate authority. The authority would usually be the device manufacturer or the service provider. Our system design is not involved in this process and assumes that each device has already obtained its certificate. However, when a personal DRM enabled device A chooses to send a file to another device B, A must already have B's public key certificate before it attempts to encrypt the content and transfer it.

For this purpose, we design our system such that each device periodically scans its surroundings for other compatible devices. Each device must also have a "listener" constantly running, to receive certificate requests and reply by sending its own certificate to the requesting device. We have implemented three different mechanisms for acquiring certificates. In the first mechanism, during the periodic scan, the device not only detects compatible devices but also requests their certificates. This mechanism is transparent to the user although we allow the user to set the frequency with which this scan should be performed. In the second mechanism we let the user manually issue a scan request, which allows the device to retrieve the certificates of all compatible devices in its vicinity. In the third mechanism, the device requests the certificate of specific device, right before transferring DRM protected content to that device. However, each device still needs to perform a periodic scan, consisting of solely building a list of neighboring compatible devices.

The first mechanism is proactive, in that devices periodically acquire the certificates of all compatible neighbors. The second and the third mechanisms are reactive, triggered by user actions. Note that unlike the second mechanisms, the third mechanism allows a device to acquire only the certificate of a specific device.

The scan and transfer itself can be done using any available technology, e.g. 802.11 or Bluetooth.

3.4. The license file

The license primarily specifies the controls on the protected content. It is a separate file that is sent to the recipient along with the encrypted content file.

We choose to use a plaintext XML file to specify the license. Our license file format is loosely based on the OMA DRM 2.0 Rights Expression Language specification [22]. Table 1 shows the primary tags used in our license file and their associated semantics.

Tag	**Semantics**
`<rights>`	Root element of the license, contains the `asset` element and the `permission` element.
`<asset>`	Specifies a particular asset i.e. content file. Contains the `digest` and `KeyInfo` elements.
`<digest>`	Specifies a hash digest of the content. Contains the element `DigestMethod` which specifies the hash algorithm used, and the `DigestValue` element which contains the hash value as a hexadecimal string.
`<KeyInfo>`	Contains information about the key used to encrypt the content. Contains the `EncryptedKey` element.
`<EncryptedKey>`	Specifies that the key included is an encrypted key. Contains the `EncryptionMethod` element which specifies the encryption method used to wrap the key, and the `CipherData` element which contains the actual encrypted key as a hexadecimal string.
`<permission>`	Root element for the permissions section of the license. These specify the rights and restrictions applied on the content.
`<constraint>`	Used within the `permission` element. Specifies a usage rule.
`<count>`	Used within `constraint`. Its value specifies how many times the content is allowed to be rendered.
`<datetime>`	Used within `constraint`. Specifies a set of dates (`start` and `end`) between which the content is allowed to be rendered. Dates are in the format `YYYY-MM-DDTHH:MM:SSZ`.
`<interval>`	Used within `constraint`. Specifies the interval in hours, minutes or days for which the content is allowed to be rendered starting from the first attempted access.
`<export>`	Used within `permission`. If specified, this element indicates that the recipient can choose to move the content and the license to another device.

Table 1: List of license file tags.

3.4.1. Digest and KeyInfo

The license contains a SHA-1 hash value of the encrypted content file. This serves to tie the license file to the corresponding content file. When an attempt is made to invoke the content, the hash value of the encrypted file is computed and compared against the value in the license file. If they do not match, the content is not allowed to be invoked.

The license also contains a `KeyInfo` element which holds a hexadecimal string value of $Pb_{Recipient}(K)$ as described in Section 3.3.1. The recipient decrypts this to obtain the content encryption key K.

3.4.2. Usage rules

The usage rules are specified in the `permission` element of the license file. They dictate what the recipient can and cannot do with the content file. For our system, we have the following three types of constraints that can be specified.

1. **Count based:** This rule specifies a value N indicating the maximum number of times the content may be rendered. For instance, if the count is set to 5, the user can invoke and play the content on five separate occasions at the most. Re-playing content e.g. a music file in a loop counts as a single instance. The count thus applies toward separate decryptions and invocations of the content.
2. **Date based:** This rule specifies a start and end date between which the content can be rendered as many times as desired.
3. **Interval based:** This rule is a more flexible version of the date based rule. It specifies a time interval in minutes, hours or days for which the content is allowed to be rendered. The interval begins when the very first attempt is made to invoke the content. E.g. a content file and license may be transferred to a recipient on April 1, with the usage rule specifying a 48-hour interval. The recipient makes no attempt to play the content until 5:45 PM on April 17. The user then has free access to the file and can play it as many times as desired until 5:45 PM on April 19.

A license file can have more than one usage rule specified. In such a case, the content is allowed to be rendered only when all the rules are satisfied. E.g. a license file specifying start and end dates of April 1 and April 15 respectively, and also a count of 10. In this case, the license will expire after the 10th invocation, or on April 15, whichever comes first.

Keeping state. The recipient device needs to maintain state information regarding all active licenses on the device. This primarily includes information on how many times the content has been rendered if the license has a count based constraint, and a timestamp of when the content was first invoked if the license has an interval based constraint. The state information is maintained separately from the actual license file, in a file with set read-only permissions solely for the DRM agent.

3.4.3. License export

In addition to the constraints specified in the previous section, the license file also has a rule allowing the export of a license. The presence of an `export` tag in the license indicates this.

Ordinarily, the protected content is bound to a specific recipient device because of the use of the intended target device's public key, as described in Section 3.3. However, the recipient may want to move the content and license to another device. The sender can sanction this by including export rights in the original license.

If the recipient chooses to export the content, the personal DRM system on this recipient device first ensures that exporting is allowed. If so, the device then generates a new license derived from the old license. Let $R1$ be the device on which the content currently resides, and the user wants to export the content to device $R2$. The user invokes an export utility on $R1$ which performs the following steps.

1. $R1$ creates a new license which is a copy of the old license, but with changes to the `KeyInfo`, and if needed, the `permission` sections.
2. $R1$ decrypts the key $Pb_{R1}(K)$ in the license to obtain K. It then re-encrypts K with the public key of the other device, i.e. $Pb_{R2}(K)$ is inserted into the new license file.
3. $R1$ updates the permissions in the new license file to match the current state of the license. For count based constraints, the originally allowed value is decremented by the number of times the content has already been invoked on $R1$. For date based constraints, no change is necessary. For interval based constraints, no change is necessary if the interval has not already begun on the current device. If it has, the constraint is altered to a date based constraint, with the start date specifying when the interval began on the current device, and the end date specifying the end of the interval. The end date is computed as $interval\_start\_date + original\_interval$.

Figure 3.4.3 shows a sample license file with a combination of count- and interval-based constraints.

3.4.4. License integrity and replays

Since the license is simply a plaintext file, a user could attempt to copy an old expired license and present it as a new license. The DRM Agent must be able to detect or prevent such replays of old licenses.

One alternative is to maintain a history of all previously seen licenses. Each time a license is presented, it is checked against all previously seen licenses to detect a replay. This can be impractical due to the storage requirement of keeping expired licenses on disk permanently and the overheard of checking each new license against a growing list of old licenses.

Instead, we prevent the possibility of license replays by using a secure transfer protocol for sending and receiving licenses, described in the next section. A genuine sender engages in an authentication protocol with the recipient before sending licenses. In effect, the DRM agent on the recipient device accepts only those licenses that are received through such a protocol. Since the protocol involves the sending device authenticating itself using its private key, the recipient DRM agent can be confident that the license comes from a genuine DRM agent on the sending device.

Also, the license is stored with read-only attributes for the device user, so that the user cannot edit the license to modify the usage rules.

```
 <rights id = "/diska/MyAudio/MySample.mp3">
<agreement>
<asset>
<digest>
<DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>hex_value_of_hash</DigestValue>
</digest>
<KeyInfo>
<EncryptedKey>
<EncryptionMethod Algorithm = "http://www.w3.org/2001/04/xmlenc#rsa" />
<CipherData>
<CipherValue>hex_value_of_encrypted_key</CipherValue>
</CipherData>
</EncryptedKey>
</KeyInfo>
</asset>
<permission>
<play>
<constraint><count>10</count></constraint>
<constraint><interval>48H</interval></constraint>
</play>
<export />
</permission>
</rights>
```

Figure 4: A sample license file

3.5. File transfer

The sending device engages in a secure authenticated transfer protocol with the recipient when sending the license file. The content file itself can be sent separate from this protocol, since it is already encrypted. The use of the authenticated transfer protocol prevents license replays, as described in the previous section, since the recipient can be confident that the license comes from a valid DRM agent on the other device.

The wireless transfer itself can take place over any available technology, such as 802.11 or Bluetooth. We use Diffie-Hellman key exchange [7] with added authentication [8]. Note that at the start of the protocol, the sender and recipient have already acquired and verified each other's certificates through the automatic certificate exchange described previously.

The sender selects the two public parameters, p and g. The parameter p is a prime number, and g is an integer less than p with the property that for every number n between 1 and $p-1$ inclusive, there is a power k of g such that $n = g^k \bmod p$, i.e. g is a generator.

The protocol proceeds as follows.

1. The sender generates a privates value x and calculates $A = g^x \bmod p$. The sender sends g, p and A to the recipient.
2. The recipient generates a private value y and calculates $B = g^y \bmod p$. The recipient also calculates $K = A^y \bmod p$ which is equivalent to $(g^x)^y \bmod p$. This forms the session key K.
3. The recipient signs the values A and B with its private key, and encrypts this signature with the symmetric session key K. The recipient sends the following back to the sender:

$$B,\ E_K(Sign_{Recpt}(A, B))$$

4. The sender takes the value B and calculates $K = B^y \bmod p$ which is equivalent to $(g^y)^x \bmod p$ which again gives the session key K. Both the sender and the recipient have thus independently derived the common session key.
5. The sender decrypts the other part of the recipient's message with the key, and verifies the recipient's signature.
6. The sender then signs the values of A and B with its private key, encrypts this with the session key and sends it back to the recipient.

$$E_K(Sign_{Sender}(A, B))$$

7. The recipient decrypts this received message with K and verifies the sender's signature.
8. The sender then encrypts the license file with the symmetric key K, sends it to the recipient and terminates the connection.

The security of the key exchange lies in the difficulty of the discrete logarithm problem. Given g^x or g^y an eavesdropper must derive g and x or y to break the protocol, which is considered difficult. In addition, signing these values ensures authentication. Both the sender and the recipient are thus confident of what device they are engaging in the transfer with.

3.6. Rendering the content

When the user tries to invoke a protected content file, the DRM agent intervenes and locates its license file. It checks the license for validity, which involves verifying the hash of the content file and enforcing the usage rules. If the rules are valid, the DRM agent decrypts the content file into protected storage where the device's user has no access. The DRM agent then invokes the appropriate software to render the content. The rendering software must also run with elevated privileges in order to access the decrypted content. When the rendering software exits, the DRM agent deletes the decrypted content file.

It is possible that the user may try to turn off the device, for instance by removing the battery, while the rendering software runs. This would mean that the DRM agent fails to delete the decrypted content. However, since the user has no read access to the decrypted file, copying is not possible. Additionally, every time the device boots, the DRM agent must scan its protected storage and clean up any remnants of such temporary files.

3.7. Adversarial model

Our design relies on the operating system to enforce some of the key security requirements. The rest, such as preventing license replays and preventing leakage of content are ensured by the DRM agent, as described in preceding sections.

We do not have a trusted CPU or tamper resistant hardware. Our system can prevent the user from trying to steal decrypted content through simple means (e.g. trying to copy the file while it plays, or turning off the phone during playback to prevent deletion) but a more determined user would be able to compromise the system. For instance, downloaded applications that are designed to exploit vulnerabilities in the operating system cannot be prevented. Similarly, a user who can simply take apart the hardware and try to read the storage chip directly from its pins cannot be stopped. Such attacks, including also the reverse engineering and "analog hole" attacks described in Section 2, can be (and some have been) successfully applied to existing commercial DRM schemes [36, 2, 1, 32, 26, 10].

Our security model is however aimed at only the casual user. Of particular concern is also the following common approach in the "cat-and-mouse" competition between DRM producers and attackers. A single skilled attacker usually exploits a vulnerability in the DRM software, then builds and distributes automated tools (e.g., scripts) allowing any content consumer to circumvent the DRM protection layer. Note however that our personal DRM scheme is intended to be deployed on cellular phones. Performing even "simple" operations like compiling, signing, downloading and installing new programs require a non-trivial effort on the part of the user and cannot be automated easily.

Thus, if the personal DRM system is being used for protecting personal content that is not extremely sensitive, we believe the current security model is adequate. For stronger protection, we would need to use system-wide security policies and/or trusted computing, as described in Section 5.1.

4. Implementation

We implemented a prototype of our design on Motorola E680i smartphones. The E680i is a multi-feature palm-size embedded linux-based cell phone with direct MPEG4 video capture and playback, a real-time 3D sound engine and 3D stereo speakers, an integrated MP3 player, a large capacity internal memory of up to 2GB, a 240 x 320 color screen, and an integrated VGA camera with 8x zoom. It features an Intel XScale 300 MHz processor with 50MB shared memory.

The full-featured Linux kernel on the phone satisfies our security requirements of file system access control and process isolation.

4.1. Implementation basics

The personal DRM system was written primarily in C++ with a graphical user interface using the Qt toolkit. We used Bluetooth technology for the wireless transfers using the open source BlueZ library, which provides a Linux implementation of the Bluetooth specifications, for this purpose. For cryptographic functions, we ported a version of the OpenSSL library to the phone.

We use SHA-1 as the hash function, RC4 for symmetric key encryption, and RSA with 1024 bit keys for public key encryption.

4.2. Certificate transfer

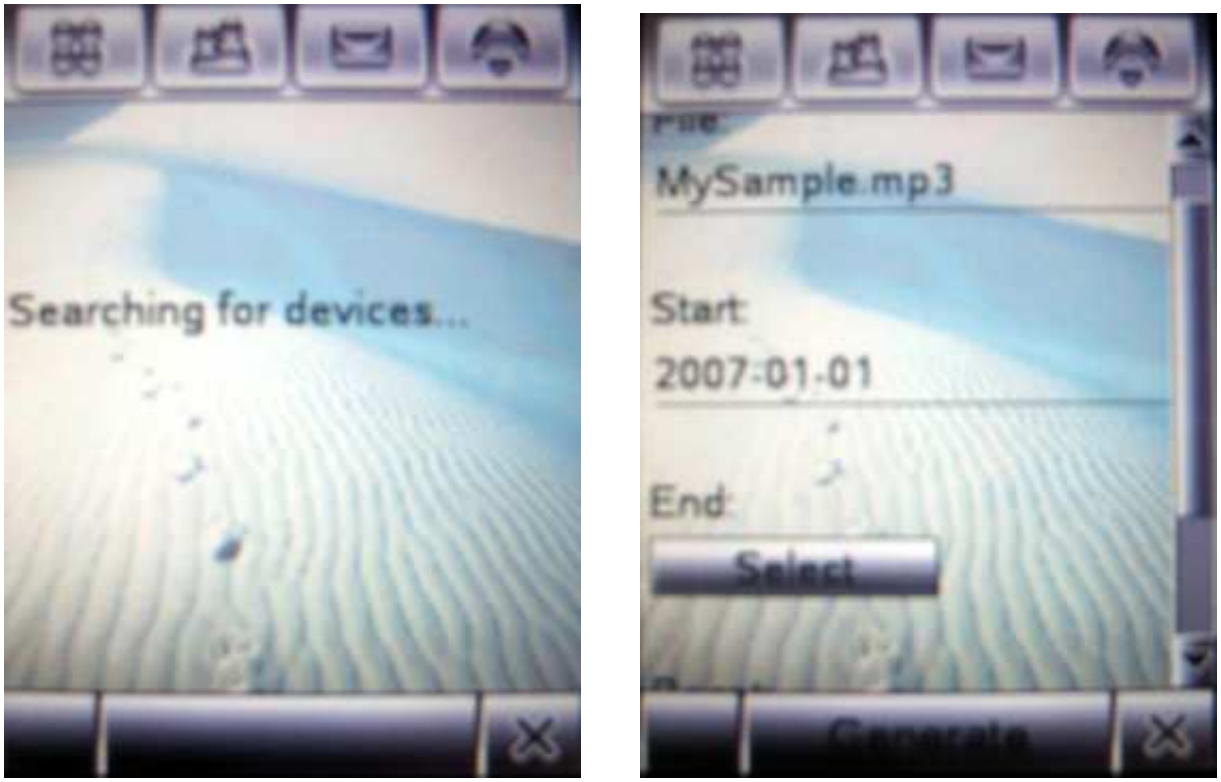
Each device has its own certificate located in a user-accessible folder. A listener daemon waits for incoming certificate requests on a specific Bluetooth port. For any incoming connection, the daemon responds by sending the device's certificate to the requesting device.

Each device also has a certificate request daemon that periodically tries to get certificates from devices around it. However, this can be impractical since it causes significant battery drain as shown in Section 4.5.2. Therefore, we also include a GUI program that disables the daemon and instead relies on the user to manually invoke a scan when desired. Figure 5(a) shows a screenshot of the manual certificate scan utility.

The certificate scan utility first issues a regular Bluetooth scan to find all devices around it. For each device found, it attempts to connect to the specific port for certificate transfers and if successful, receives the certificate data from the device. The program creates a file `<dev-name>.pem` in a `certificates` folder on the phone, where `<dev-name>` is the Bluetooth name of the device whose certificate was fetched.

4.3. Content packaging and sending

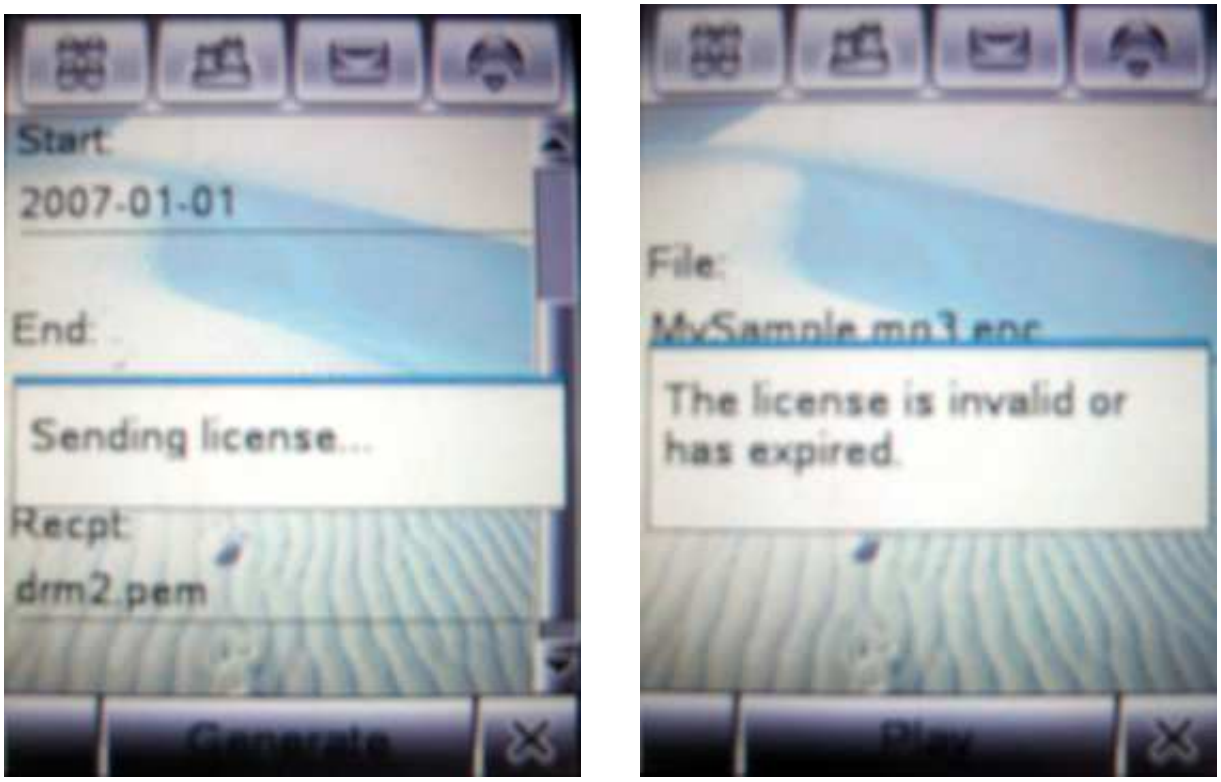
The content packaging GUI is the main user-level application for packaging content, generating licenses and sending them to recipients. The program allows a user to select a content file and define usage rules for it. The user is also prompted to select a recipient. To do this, the contents of the `certificates` folder are displayed and the user is prompted to select the certificate corresponding to the recipient's device. Recall that the certificates are named after the Bluetooth display name of the device, and so the `certificates` folder serves as an "address book" of recipients. Once the user has specified these parameters, the



(a) **(b)**

Figure 5: E680i Personal DRM implementation screenshots (a) Core Entities in a DRM System. (b) Content packaging utility

content packaging utility encrypts the content file, generates a license file and attempts to connect to the target device. Figure 5(b) shows a screenshot of the content packaging utility.



(a) **(b)**

Figure 6: E680i Personal DRM implementation screenshots (a) File transfer (b) Content packaging utility

At this point, the target device must run a Receive utility which is also a GUI program that lets the recipient receive a DRM file transfer and specify where to save the received content. The sender and recipient perform the secure protocol handshake and securely transfer the license file. The content file is transferred in the clear. Figure 6(a) shows a screenshot of the file transfer. On the recipient device, all protected content files are saved with the extension `.enc` and license files are saved with a `.lic` extension.

4.4. Playing the content

When the user attempts to invoke protected content, the DRM agent locates its license file and performs a series of checks. This includes verifying that the hash of the encrypted content matches the one specified in the license, and checking the usage rules to see if they are valid at the current point in time. If the checks are successful, the DRM agent decrypts the file and invokes the relevant program to render the content e.g. a media player. When the media player quits, the DRM agent deletes the decrypted file. Figure 6(b) shows a

screenshot of an attempt to play expired content. In such a case, the DRM agent does not decrypt the content file.

4.5. Performance evaluation

4.5.1. Benchmarking cryptographic operations

We benchmarked the performance of OpenSSL on the E680i to get an idea of the speed of cryptographic operations on the phone. Table 2 shows the number of Kbytes per second the E680i can process when performing a SHA-1 hash or symmetric-key encryption/decryption using RC4. For the sake of comparison, we also include corresponding figures from a personal computer with a 2.16 Ghz Intel processor. The most relevant cryptographic operation in our implementation is RC4. Note that RC4 is much faster than SHA-1.

Algorithm	**Block Size**	**E680i (KB/s)**	**PC (KB/s)**
SHA-1	16 bytes	1006.37	17868.43
SHA-1	64 bytes	2923.05	57162.12
RC4	16 bytes	14475.59	214763.07
RC4	64 bytes	17760.43	282504.58

Table 2: Performance of SHA-1 and RC4 in KBytes/second.

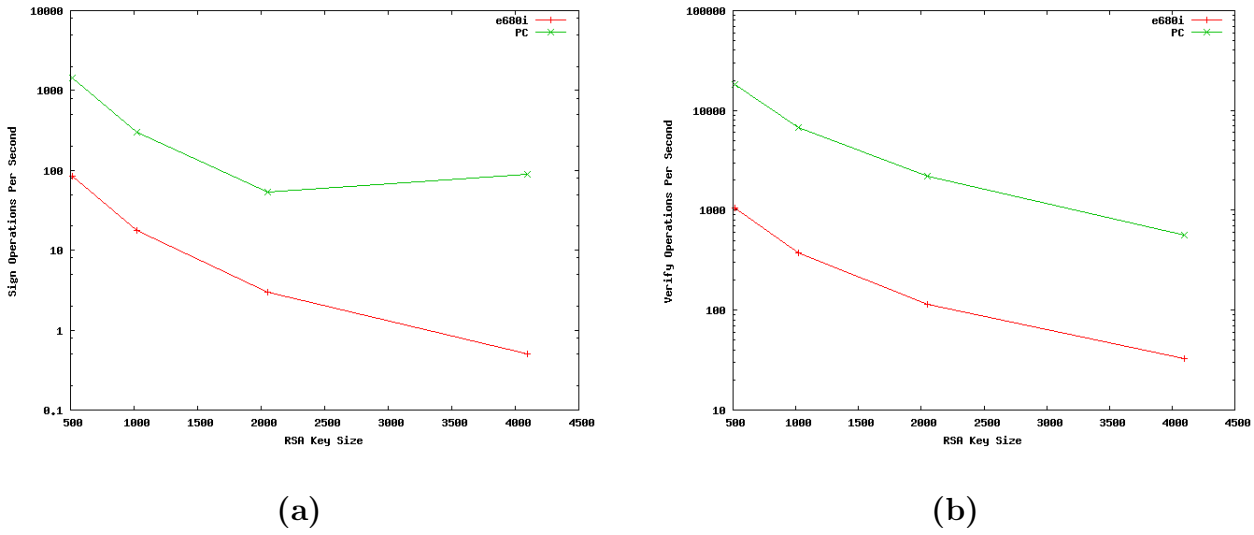
Table 3 shows the number of RSA sign and verify operations per second that the E680i can perform for various key sizes. Once again, corresponding figures for a personal computer are included.

Operation	**E680i (ops/s)**	**PC (ops/s)**
RSA 512-bit sign	84.5	1454.3
RSA 512-bit verify	1063.4	18312.1
RSA 1024-bit sign	17.6	298.4
RSA 1024-bit verify	372.7	6693.5
RSA 2048-bit sign	3.0	54.1
RSA 2048-bit verify	114.5	2193.9

Table 3: Performance of RSA in operations/second.

Figure 7(a) shows the decline in the number of RSA signing operations per second for increased key size for the E680i and a PC. Though the overall trend is the same, the PC is significantly faster than the mobile device. Figure 7(b) shows the same trends for RSA verify operations.

Naturally, the mobile device is significantly slower than the personal computer, but the overall performance is acceptable for our purposes.



(a) **(b)**

Figure 7: E680i RSA Benchmarks. (a) Number of RSA sign operations/s vs. key size. (b) Number of RSA verify operations/s vs. key size.

4.5.2. Battery drain

In our initial design, devices are set to periodically look for other devices and get their certificates automatically. In practice, that means issuing a Bluetooth scan and then attempting to connect to each device found and getting a certificate from it.

We evaluated the effect of periodically doing a Bluetooth scan on the device's battery. Table 4 shows the results of the total battery time when no scanning is performed, and when scans are performed at various intervals. Figure 8(a) shows the decline in the percentage loss of overall battery life as the scan interval increases.

Scan Setting	**Battery Lifetime**
No scanning (idle)	23.6 hours
Scans at 10-minute intervals	11.5 hours
Scans at 20-minute intervals	14.25 hours
Scans at 30-minute intervals	18.5 hours

Table 4: Battery lifetimes.

We observe that scanning at ten-minute intervals reduces the battery lifetime to almost half that of an idle device. Scans at longer intervals also have a significant effect on battery drain. The purpose of the automatic certificate scan is to have an "address book" of devices ready when the user wants to send a piece of content to someone else.

Having too large a scan interval defeats the purpose as the sender might not have discovered the intended recipient when a user initiates the sending program. But having a small interval is not practical, as we see from the results above. Therefore, in practice, the most efficient option is to have the user manually invoke a certificate discovery when needed. This

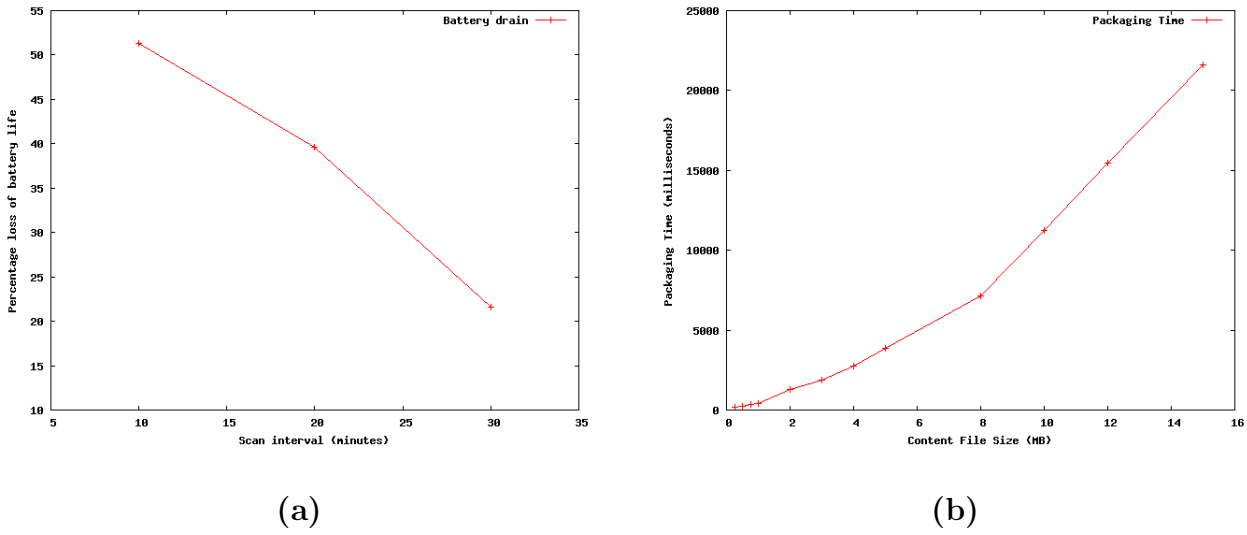


Figure 8: (a) Percentage loss of battery life vs. scan interval. (b) Content packaging time vs. content file size.

is what we have implemented in our prototype.

Benchmarking results pertaining to the effect of cryptographic operations on battery drain in a similar mobile device can be found in [30]. Specifically, the work in [30] presents extensive benchmark test results for cryptography techniques on two mobile devices. The benchmarks measure the battery consumption of each cryptographic tool, including symmetric encryption, hashing and public key crypto, on wireless devices in terms of number of computations per unit percent battery drain. The results, coupled with the strength of the algorithms, motivate our choice of RC4 for symmetric key encryption, SHA-1 for hashing and RSA for public key cryptography.

4.5.3. Content packaging and transfer

We evaluated the actual total time our content packaging utility takes to prepare a content file for transfer. This represents the latency that the user experiences when trying to generate a license for some content. This operation includes applying symmetric RC4 encryption to the content file, and then applying RSA encryption to the 128-bit symmetric key. Figure 8(b) shows the increase in the packaging latency as the size of the content file increases. Table 5 shows the actual time taken in seconds to package content files of certain sizes.

We observe that the packaging time does not grow perfectly linearly with file size. Larger file sizes take longer, presumably because of the limited computational abilities of the phone, and because of the processor having to multi-task with other real-time processes on the phone. However, smaller file sizes, such as 5MB or less, have a relatively small packaging time. Since we envision the personal DRM system being used primarily for personal content such as videos, sound clips or pictures, and the usual size for such files on a phone is generally small, we consider this a reasonably good result. An average personal video can be packaged

in under five seconds.

Content File Size	Time
500 KB	0.246 s
1 MB	0.438 s
2 MB	1.272 s
4 MB	2.778 s
8 MB	7.160 s
15 MB	21.625 s

Table 5: Latency of content packaging application.

We also evaluated the file transfer rate and found that the Bluetooth transfer provides an average rate of 73.6 KBps. This may seem unusually low, but is a limitation imposed by the particular implementation of Bluetooth that we used. Using a better Bluetooth library, or instead using 802.11 for wireless transfers may prove to be faster.

4.5.4. Certificate transfer

We evaluated the time it takes for a set of devices to get each others' certificates. If two personal DRM devices attempt a certificate scan at the same time, the total time taken for them to discover each other and fetch each other's certificates is 12 seconds. In this case, each device has to fetch only one certificate – the other device's. For three devices attempting to scan at the same time, each device has to fetch two certificates. The total time for the entire process to complete is 37 seconds. The increased time is due to collisions. We estimate that for a large number of devices in the same room, all attempting to fetch certificates from each other, the entire process may take several minutes to complete.

This may be impractical if a user in a crowded room is attempting to get the certificate of just one particular user. Thus, the solution we adopted is for any device to request a certificate only from the device of interest, before attempting to package and transfer DRM protected content. Note however that this approach is vulnerable to denial of service attacks, attempting to drain the battery of targeted victims. Specifically, a resource rich attacker (e.g., with access to a plugged-in computer) can issue many certificate requests, each with a unique IP/MAC address, to the same target device. Since the victim is unable to determine the fact that all the requests come from the same device, it will be forced to answer, thus draining its battery much faster. Our solution then is to make the certificate response daemon manual as well. Users can choose when to let their devices respond to certificate requests.

5. Conclusions

The main contributions of this work are twofold. First, it proposes a first personal DRM system that allows users to retain control over how the content they have produced is consumed, even when this takes place on devices other than their own. The second contribution

consists of the implementation of a prototype system on several Motorola E680i smartphones. Our observations resulting from extensive testing of the system are that the content packaging and license generation operations are quite efficient. However, the certificate exchange services causes significant battery drain if set to periodically scan for and respond to devices automatically. Our conclusion then is that the practical approach is to have devices retrieve certificates on demand, before sending DRM protected content.

5.1. Future Work

5.1.1. License expressivity

At present, the expressivity of our licenses is limited. We plan to implement more usage rules to strengthen the content producer's control over its consumption while introducing more flexibility in how the consumer can use the content, for instance, allowing the rendering of a down-sampled version of the content after the license expires.

One planned usage rule is dynamic proximity-based control. It would allow a user to send content to another device with the stipulation that the content stays active on the recipient's device as long as it stays within wireless range of the sender. One simple way to accomplish this would be to have the DRM agent on the recipient's device continually ping the sender to check if it is in range. However, we have seen that such a technique introduces a large overhead in terms of battery drain. It would therefore be an interesting challenge to design an efficient proximity based control.

Along similar lines, we also envision a location based control scheme which could be useful in enterprise applications. An organization could protect proprietary data on employees' portable devices by specifying that the data be accessible only when the device is physically located within the company's premises.

5.1.2. Security policy based framework

The enforcement of security in our design is inherently distributed. Low-level factors such as process isolation and memory management are handled by the operating system on the individual devices. The DRM agent on each device handles the enforcement of the license controls and deletion of decrypted content after playback finishes. As an alternative, we would like to design a centralized security policy based framework for the personal DRM system. The security policy can specify all aspects of the system, from attributes on protected storage areas to license enforcement and decryption of content. Each device would have a policy certificate, containing policy settings assigned by some authority, which could be the device manufacturer or the service provider.

The advantage of using a policy based framework is increased flexibility. Currently, rules about enforcement, access rights etc. have to be hard-coded into the DRM agent. The only configurable factor are the usage rules on the content, which the DRM agent interprets and enforces. Using policies is similar, except the enforcement occurs at a lower level. Not just the content usage rules, but all other policies in the system would be configurable, and would be derived from a security policy certificate, thus making them easier to change if desired.

The policy based framework would be especially useful in enterprise DRM situations, like the one mentioned in Section 5.1.1. A centralized policy authority for the organization

could specify the detailed policies for employees' mobile devices. For instance, an employee could share personal content with custom controls, as is the purpose of the personal DRM system, but the policy would ensure that the controls do not supersede the organization's business rules.

5.1.3. Trusted hardware

We would like to design our system for trusted computing hardware. The use of a trusted platform module would provide secure key generation and storage and would help strengthen the security of various aspects of our system, such as license integrity verification and secure transfer. Tamper resistant hardware would mean that even an extreme attack like opening up the device and trying to read directly from the storage chip would not work.

Additionally, trusted hardware can provide a trusted clock source which the DRM system can rely on to keep track of date-based usage rules. Similarly, trusted hardware can provide monotonic counters to keep track of count-based rules. The monotonic counters can only be incremented and not decremented, thus preventing tampering of state information about count-based rules.

The remote attestation service provided by the trusted computing model can be especially beneficial. Our current transfer protocol uses an SSL-like protocol to mutually authenticate devices and transfer licenses. This could be replaced by remote attestation which could be used to authenticate the other device and also verify the integrity of the device. The trusted platform module on the recipient device could attest to the validity of the DRM software running on that device so that the sender can be confident that the DRM agent, media rendering software etc. on the recipient have not been compromised.

5.1.4. Attack Analysis

An interesting direction that we plan to follow in the future is to explore various attack techniques, some described in more detail in Section 2, on our Personal DRM system. In particular, it would be interesting to determine if we can devise a simple, "one-click" attack that would allow any smartphone user to circumvent the DRM protection mechanisms implemented by our solution.

Authors.

Siddharth Bhatt is a graduate student in Computer Sciences at Stony Brook University. His research interests are in network and computer security.

Radu Sion is an assistant professor of Computer Sciences in Stony Brook University, heading the Network Security and Applied Cryptography Laboratory. His research focuses on data security and information assurance mechanisms. He has been applying practical cryptography and strong assurance mechanisms to achieve practical data privacy solutions, develop efficient regulatory compliant systems, cellular DRM solutions and conditional micro-payment schemes. Sion also directs the Stony Brook Trusted Hardware Laboratory, a central expertise and research knowledge repository on secure hardware. Collaborators and funding partners include Motorola Labs, Xerox/Parc, IBM Research, the IBM Cryptography Group, the Center of Excellence in Wireless and Information Technology CEWIT, the Stony Brook Office for the Vice-President for Research and the National Science Foundation. Sion serves on the organizing committee and steering boards of conferences such as CCS, NDSS, FC, USENIX Security, SIGMOD, ICDE, S&P, ICDCS, a.o.

Bogdan Carbunar is a senior staff member in the Applied Research and Technology Center of Motorola. His main work focus is on (i) devising secure applications for mobile devices and (ii) improving the scalability of video on demand services. His broader interests also span the areas of applied cryptography, data and network security and distributed algorithms, with particular applications in private information retrieval and electronic payment technologies.

References

[1] How to break drm (itunes, dvd, etc) for lawful purposes. http://www.antidrm.hpg.ig.com.br/.

[2] How to break itunes drm. http://www.iterasi.net/openviewer.aspx?sqrlitid=giuslah7t0uf6ltbe5lqea.

[3] Apple Fairplay DRM. http://www.apple.com/itunes.

[4] C. N. Chong, R. Corin, S. Etalle, P. Hartel, W. Jonker, and Y. W. Law. LicenseScript: A Novel Digital Rights Language and its Semantics. In *Third International Conference on WEB Delivering of Music*, page 122, 2003.

[5] Content Scrambling System, DVD Copy Control Association. Available at http://www.dvdcca.org/css/.

[6] A. Cooper and A. Martin. Towards an open, trusted digital rights management platform. In *DRM '06: Proceedings of the 6th ACM workshop on Digital Rights Management*, pages 79–88, 2006.

[7] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[8] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[9] J. S. Erickson. Fair use, DRM, and trusted computing. *Communications of the ACM*, 46(4):34–39, 2003.

[10] S. Gilbertson. Windows media drm hacked yet again. http://blog.wired.com/monkeybites/2007/07/windows-media-d.html, 2007.

[11] C. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *34th Annual Hawaii International Conference on System Sciences*, page 9076, 2001.

[12] H. Guo. Digital Rights Management (DRM) using XrML. *T-110.501 Seminar on Network Security 2001*, 2001.

[13] J. A. Halderman. Evaluating new copy-protection techniques for audio CDs. In *DRM '02: Proceedings of the 2nd ACM Workshop on Digital Rights Management*, 2002.

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

[15] International Telecommunication Union, Recommendation X.509, August 2005. Available at http://www.itu.int/rec/T-REC-X.509-200508-I.

[16] D. W. Kravitz and T. S. Messerges. Achieving media portability through local content translation and end-to-end rights management. In *DRM '05: Proceedings of the 5th ACM Workshop on Digital Rights Management*, pages 27–36, 2005.

[17] S. H. Kwok. Digital rights management for the online music business. *ACM SIGecom Exchanges*, 3(3):17–24, 2002.

[18] Y. Y. Leung, M. Peinado, and C. P. Strom. Binding digital content to a portable storage device or the like in a digital rights management (DRM) system. U.S. Patent 7010808, March 2006.

[19] Q. Liu, R. Safavi-Naini, and N. P. Sheppard. Digital rights management for content distribution. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 49–58, 2003.

[20] A. Matheus. Authorization for digital rights management in the geospatial domain. In *DRM '05: Proceedings of the 5th ACM Workshop on Digital Rights Management*, pages 55–64, 2005.

[21] T. S. Messerges and E. A. Dabbish. Digital rights management in a 3G mobile phone and beyond. In *DRM '03: Proceedings of the 3rd ACM Workshop on Digital Rights Management*, pages 27–38, 2003.

[22] Open Mobile Alliance, DRM v2.0 Specifications, March 2006. Available at `http://www.openmobilealliance.org/release\_program/drm\_v2\_0.html`.

[23] J. Park, R. Sandhu, and J. Schifalacqua. Security architectures for controlled digital information dissemination. In *Proceedings of the 16th Annual Computer Security Applications Conference*, page 224, 2000.

[24] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Support for multi-level security policies in DRM architectures. In *NSPW '04: Proceedings of the 2004 Workshop on New Security Paradigms*, pages 3–9, 2004.

[25] J. F. Reid and W. J. Caelli. DRM, trusted computing and operating system architecture. In *ACSW Frontiers '05: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 127–136, 2005.

[26] B. Rosenblatt. Windows media drm hacked. `http://www.drmwatch.com/drmtech/article.php/3629681`, 2006.

[27] N. Rump. Can digital rights management be standardized? *IEEE Signal Processing Magazine*, 21(2):63–70, 2004.

[28] R. Safavi-Naini, N. P. Sheppard, and T. Uehara. Import/export in digital rights management. In *DRM '04: Proceedings of the 4th ACM Workshop on Digital Rights Management*, pages 99–110, 2004.

[29] P. Samuelson. DRM {and, or, vs.} the law. *Communications of the ACM*, 46(4):41–45, 2003.

[30] D. Shah and S. Zong. Benchmarking security computations on wireless devices. Available at `http://www.cse.buffalo.edu/tech-reports/2006-19.pdf`.

[31] N. P. Sheppard and R. Safavi-Naini. Sharing digital rights with domain licensing. In *MCPS '06: Proceedings of the 4th ACM International Workshop on Contents Protection and Security*, pages 3–12, 2006.

[32] I. Skochinsky. Mobipocket books on kindle. `http://igorsk.blogspot.com/2007/12/mobipocket-books-on-kindle.html`, 2007.

[33] M. L. Smith. Digital rights management & protecting the digital media value chain. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 187–191, 2004.

[34] M. Stamp and P. Priyadarshini. Digital rights management for untrusted peer-to-peer networks. *IGI Global*, 2008.

[35] M. Stamp and R. Venkataramu. P2ptunes: A peer-to-peer digital rights management system. *IGI Global*, 2008.

[36] F. A. Stevenson. Cryptanalysis of contents scrambling system. `http://www.dvd-copy.com/news/cryptanalysis_of_contents_scrambling_system.htm`, 1999.

[37] Verizon V CAST. `http://getitnow.vzwshop.com`.

[38] A. Weiss. Trusted computing. *netWorker*, 10(3):18–25, 2006.

[39] Zune Sharing. `http://www.zune.net`.