

Analysis of Using Java Card™ for DRM Master Key Security

John Buford

*Panasonic Digital Networking Lab
2 Research Way, 3rd Floor
Princeton, NJ 08540, USA
buford@research.panasonic.com
+1 609 734 7342*

Rakesh Kumar

*Department of Electrical and Computer Engineering
Polytechnic University
Brooklyn, NY, USA
rkumar04@utopia.poly.edu
+1 347 224 6745*

Abstract

Recently Wei and Montemayor [1] have demonstrated a Java Card smart card application for authenticating playback for DRM-protected media. In this scheme the Java Card stores the encrypted master key and communicates during media playback with a signed content handler to securely provide the segment keys to the content handler for decryption and rendering of the media.. In this paper we summarize the Java Card-based design and analyze its vulnerabilities.

Keywords: DRM, Java Card, content handlers, trust

TRACK: DRM Workshop

Analysis of Using Java Card™ for DRM Master Key Security

John Buford
Panasonic Digital Networking Lab
Princeton, NJ, USA
buford@research.panasonic.com

Rakesh Kumar
Polytechnic University
Brooklyn, NY, USA
Rkumar04@utopia.poly.edu

Abstract

Recently Wei and Montemayor [1] have demonstrated a Java Card™ smart card application for authenticating playback for DRM-protected media. In this scheme the Java Card stores the encrypted master key and communicates during media playback with a signed content handler to securely provide the segment keys to the content handler for decryption and rendering of the media.. In this paper we summarize the Java Card-based design and analyze its vulnerabilities

1. Introduction

Recently we have witnessed a strong interest in digital rights management (DRM) systems which can be used to protect digital media content by enforcing pre-defined access and distribution rules. In general DRM refers to the protection, distribution, modification and enforcement of the rights associated with the use of digital content [2]. There have been a number of proprietary proposals for DRM architectures (e.g., Microsoft Media Player) but in essence all DRM mechanisms rely on cryptography (symmetric key ciphers, PKI etc.) to achieve secure delivery of content, usage rights and authentication.

Contemporary DRM architectures follow a client-server paradigm wherein the content issuer's media servers control access to the digital media and store and distribute it in packaged (encrypted) form to various clients. The key distribution servers independently distribute access rules and decrypting keys to an authenticated client after appropriate payment mechanisms. The authenticated client then accesses digital media according to access rules after decrypting it with the keys supplied by the key distribution servers.

As the number and functionalities available in CE (consumer electronics) devices proliferates a potentially huge market is being created for digital

media delivery to these devices. along with the possibilities of content delivery on CE devices, responsibilities for rights management through an appropriate DRM implementation on these devices will also arise. To this end, the Open Mobile Alliance (OMA), a standardization organization for service enablers in the mobile domain has completed work on version 2 of its open standard for DRM on mobile terminals [3].

Among the numerous approaches of enabling a DRM system on an end-user CE device, use of Java Card [4] to implement DRM has been recently proposed and demonstrated [1]. The advantage of such an approach is that not only vendor independence is achieved but Java Card already has a widely installed client base as in GSM cell phones with Java Card based SIM. Further, Java Card provides for a secure execution environment with extensive support for cryptographic algorithms and is optimized for PKI operations which lie at the core of achieving secure delivery of digital media in DRM architectures.

The goal of this paper is to analyze potential vulnerabilities in such an approach. We argue that this DRM implementation inherently relies on existence of a trusted content handler and the ability to create a secure binding between the content, the content handler, and the master key. We describe the possible ways a trusted content handler can be obtained and also discuss related security vulnerabilities.

The next section describes the Wei and Montemayor design and provides background on Java Card. Section 3 provides an overview analysis, and section 4 looks at specific cases by which the content handler can be trusted. Section 5 describes related work, and section 6 concludes the paper.

2. Java Card-based DRM

2.1 Java Card

Java Card [4][5] is a widely deployed smart card running a specifically designed version of the Java virtual machine (JVM) (Figure 1). Java Card technology enables smart cards and other devices to run applets. This technology has several benefits like interoperability between various card vendors as the same applet will run on any Java Card technology-enabled smart card. Also it provides for a dynamic solution wherein new applications can be installed securely after a card has been issued. This ensures that in case implementation flaws are discovered, software patches can remedy the situation.

Applets running on the JVM use special APIs to access the smart card environment or communicate with host-side processes. Each applet has an execution context that controls access to the objects assigned to it. The boundary between one applet execution context and another is called an applet firewall. Communication between the Java Card and the host is either through message passing over the APDU or using Java Card RMI (remote method invocation).

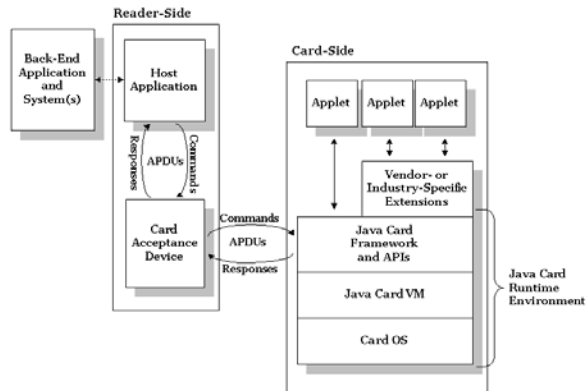


Figure 1 Java Card architecture

As a smart card, Java Card security features include: a tamper-resistant package, encrypted persistent memory systems, and resistance to differential power analysis. It has support for RSA asymmetric ciphers and allows for multi-factor authentication. It has in-card key generation capabilities and can store public key pairs for trusted CA entities which allows verification of certificates presented to the card.

Security in the Java Card is determined by various aspects. First of all the data is stored in the card and java card applets are executed in an isolated environment, separate from underlying OS and

hardware. Further Java Card firewall ensures that private data can not be accessed by untrusted applets. More importantly for DRM implementations, the Java Card provides isolation between the decryption keys and the actual content handler even though the information as to which content corresponds to which handler is stored locally on the Java Card.

Implementations of Java Card have received security certifications including US NIST FIPS 140-1 and Common Criteria EAL 5+ Certification.

2.2 Java Card-based DRM

In the Wei and Montemayor design (Figure 2), protected content is encrypted using conventional techniques, and the master key is encrypted with the public key of a licensor. This master key is used during playback to decrypt segment keys. Each segment's key is used by the content handler to decrypt the associated media segment so that the media segment can be decoded and rendered.

The role of the Java Card is to store the encrypted master key and to decrypt each segment key using the master key. The content handler including the media decryption function is on the host platform to which the Java Card is attached. Each decrypted segment key is passed to the content handler using Java Card messages. Additionally, the content handler software is signed to verify the identity of the content handler software.

In the absence of a content handler running on a secure OS [6] on a trusted computing base [7], the Java Card provides a partial secure environment in which the master key and the decryption of the segment keys is protected, but the content handler itself is vulnerable. We observe also that the Java Card could be used with a content handler running on a secure OS on a TCB.

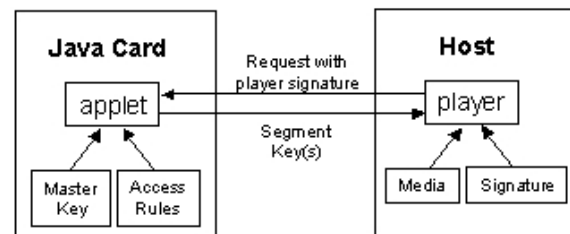


Figure 2 Playback of encrypted media using Java Card applet to decrypt segment keys [1]

3. Analysis

Analysis of the security vulnerabilities of this playback approach can include the individual

components (such as the Java Card or the content handler), the interaction of the components, key management, and specific encryption algorithms. For this paper we are primarily interested in the trust relationship between content handlers, content issuers, and the Java Card authentication and decryption applet. We are not analyzing Java Card security itself or specific encryption algorithms, which are already well-studied topics.

The content handler runs outside the Java Card and if installed on a conventional computing platform is then subject to well-known vulnerabilities such as access control granularity and OS attacks. In addition the content handler could contain back door or trojan horse components which compromise the playback security.

One critical assumption in this approach is the existence of a *trusted* content handler. The content handler receives the segment keys for the given content and can potentially expose them either intentionally or inadvertently. As we see in the next section different possibilities of the source of this content handler lead to different security implications.

Figure 3 shows the top level interactions between Java Card and the content handler. The Java Card applet acquires access rights including the master key from the rights issuer and stores these access rights securely on the card. Then when content playback is initiated, the content handler must be authenticated to the Java Card applet that handles segment key decryption. The applet decrypts the segment keys using the master key previously stored on the card, at the request of the content handler. The content handler receives the segment key and decrypts and renders the media segment.

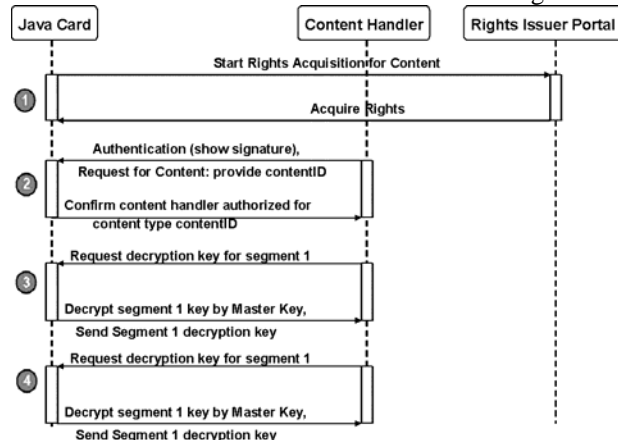


Figure 3 Top level interactions between Java Card applet and the host content handler

The question of the source of the content handler raises the question as to why the content issuer can

trust the content handler itself. As we see in the next section, there could be a variety of ways through which the content issuer agrees to trust the content handler on the user device.

There may be more than one content handler on the host, along with content from multiple content issuers available in the system. It is possible that different content issuers prefer to trust different content handlers for content playback in the system. Hence, the applet running on the Java Card needs a method of precise determination of which master key encoded for a particular content corresponds to which content handler for playback. That is, in the general case where there are many possible content handlers, there must be a verifiable and secure binding {master key, content, content handler}.

In the case of availability of more than one content handler which is trusted by the content issuer, the applet needs a policy to unambiguously select an appropriate content handler. We discuss answers to these and related issues in the next section.

4. Trusted Content Handler Scenarios

4.1 Trusted Content Handler

In the Wei and Montemayor design, the content handler provides its implementation signature to the Java Card applet at run time to identify itself as a trusted content handler.

First this is a slightly different use of software signatures than existing practice, in which the signature is used in two ways: 1) at software installation time on a platform, 2) when software is loaded in to a dynamic run-time environment such as a Java VM. In this case the signature of the content handler must be verified at run-time over the APDU by the Java Card applet. This means that the applet should be able to directly re-compute the hash function on the content handler binary and compare it with that value stored in the software's signature. However this may be time consuming because of data transfer between the Java Card and the host. An alternative approach would be to have an OS component on the host perform the runtime signature verification on behalf of the applet and communicate the result to the applet. But this leaves this OS component as a security vulnerability.

Further, in either approach, the communication to the applet should be under control of the host OS and must be secure, non-interceptable, and non-spoofable. Otherwise, a malicious component could spoof the content handler by using the real content handler's

executable and signature when communicating with the Java Card.

The signature on the content handler indicates that the software is from the given supplier. Alternatively if the software is signed by a content issuer, it could indicate that the content issuer authenticates the software for playback of its content. Or, if the software is signed by a third party validator, it could be that the software has been audited or validated by the third party. The intent then is to provide assurance to the content issuer and the licensee that the content handler does not have a backdoor, trojan horse, or other hole that would lead to the segment keys being exposed.

We now enumerate different cases in which the content handler is trusted by the content issuer and securely bound to the content master key so that the Java Card can verify this binding (Figure 4).

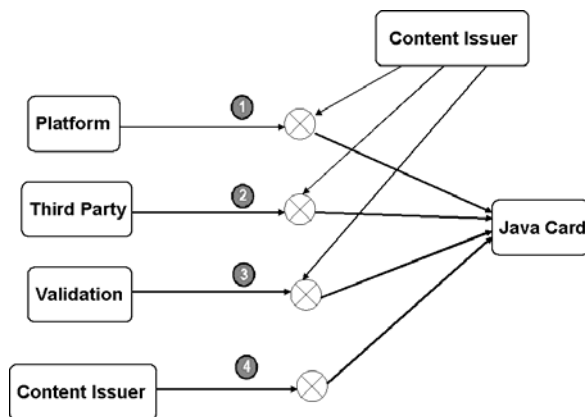


Figure 4 Different bindings between content handler signature, content issuer, and content

4.2 Platform Specific Content Handler

The first case is where there is only a single content handler which can be used on a given platform. This could be the situation if the content handler is provided by the platform vendor. In this case there is no issue of selection of an appropriate content handler by the content issuer as there is only one choice. However, the content issuer's choice to trust this content handler is related to the content issuer's trust of the platform. In this case a secure and verifiable binding is needed between the following entities:

- Content, Master Key, Content Issuer, Platform

We assume that the content handler is verifiably and securely bound to the platform.

4.3 Content Issuer Supplies Content Handler

Another possibility is that the content issuer offer its own content handler. In this case, the trust issue between the content issuer and the handler are completely taken care of. Also, the applet in this case can always by default choose the content issuer's handler for content playback. In this case a secure and verifiable binding is needed between the following entities:

- Content, Master Key, Content Issuer, Content Handler

4.4 Validated Content Handler from Any Third Party

On the other hand after due verification, the content issuer can also choose to trust a third party content handler. It can directly trust a given third party, or it can trust a content handler that has been validated by trusted validator. These two cases are covered by the following bindings:

- Content, Master Key, Content Issuer, Content Handler Vendor, Content Handler
- Content, Master Key, Content Issuer, Content Handler Validator, Content Handler

4.5 Summary

The trust bindings described above are summarized in Table 1. Each component of a binding must have a secure, unique, verifiable id. The binding must be encrypted, such as by using the consumer's public key when the license is obtained, and stored on the Java Card for use during verification of the content handler.

Table 1 Trust bindings for secure playback

	Master Key	Content Issuer	Content Handler Vendor	Content Handler Software
Platform Content Handler	Y	Y	Platform vendor	Y
3 rd Party Content Handler	Y	Y	C.H. Vendor	Y
3 rd Party Content Handler	Y	Y	Validator	Y
Content Issuer's Content Handler	Y	Y	Content Issuer	Y

In addition to the availability of the appropriate binding to the Java Card applet, if there are multiple content handlers to choose from, there must be an

efficient way for the Java Card to quickly match content, content handler and master key.

Further, the bindings might distinguish between specific components (e.g., content handler version 3.1.2) and content handlers from vendor X. One purpose is to permit a set of content handlers to be covered by the binding. Another purpose is to permit upgrades of trusted content handlers to be covered by a binding.

5. Related Work

A security architecture [8] allows DRM in home networks consisting of CE devices. The idea is to allow devices to establish dynamic groups where legally acquired copyrighted content can seamlessly move from device to device. Java Card approach is complementary with this.

A limitation of existing DRM architectures in reacting to possible device compromise is presented by Popescu et al [9]. The authors advocate a multi-level security policy that differentiates between devices based on their proven tamper-resistance properties.

Messerges et al [10] focuses on how copyright protection of digital items can be securely managed in a 3G mobile phone and other CE devices.

As an alternative to PKI which requires a two way handshake between the client and the media server, [11] advocates use of broadcast-encryption based content distribution system, which can work without requiring any secrets in the DRM client.

6. Summary

We reviewed a new approach to DRM based on Java Card for authenticating playback of DRM-protected media. We then analyzed the vulnerabilities of the scheme focusing on: (a) the source of the trusted content handler, (b) motivation of the content issuer to trust the content handler, and (c) the requirements of the Java Card applet to act as an authenticator the content handler when rendering a given content object. Depending on the source of content handler, we identified the information needed to bind the authenticated components for content issuer and Java Card applet to trust the content handler during handling of the segment keys.

7. References

- [1] J. Wei, O. Montemayor, "Writing Java Card™ Applications for Digital Rights Management and Security", *JavaOne* 2006.
- [2] A. M. Eskicioglu, J. Town and E. J. Delp, "Security of Digital Entertainment Content from Creation to

- Consumption", *Proceedings of SPIE Applications of Digital Image Processing XXIV*, Vol. 4472, San Diego, CA, July 31-August 3, 2001, pp. 187-211.
- [3] Open Mobile Alliance, "DRM Specification V2.0", Open Mobile Alliance Ltd, 2004, La Jolla (CA), USA.
- [4] Java Card Forum, <http://www.javacardforum.org>
- [5] Z. Chen, "Java Card™ Technology for Smart Cards: Architecture and Programmer's Guide", *Addison-Wesley* 2000.
- [6] C. Wright, C. Cowan, S. Smalley, J. Morris and Greg Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel", *11th USENIX Security Symposium*, San Francisco, CA, August, 2002
- [7] "TCG Specification Architecture Overview", *Spec. Rev. 1.2*, <http://www.trustedcomputinggroup.org>, 28 April 2004
- [8] B. C. Popescu , B. Crispo, A.S. Tanenbaum and F. Kampermann, "A DRM security Architecture for Home Networks", *Proceedings of the 4th ACM workshop on Digital rights management*, Washington DC, USA, 2004.
- [9] B. C. Popescu , B. Crispo and A.S. Tanenbaum, "Support for Multi-Level Security Policies in DRM Architectures", *Proceedings of the 2004 workshop on New security paradigms*, Nova Scotia, Canada.
- [10] T. S. Messerges and E. A. Dabbish, "Digital Rights Management in a 3G Phone and Beyond", *Proceedings of the 2003 ACM workshop on Digital rights management*, Washington, DC, USA
- [11] J. Lotspiech, S. Nusser and F. Pestoni, "Anonymous trust: digital rights management using broadcast encryption", *Proceedings of the IEEE* Volume 92, Issue 6, June 2004 Page(s):898 – 909.