Diplomarbeit

Christian Grewenig

Ein Framework für Digital Rights Management Systeme

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Christian Grewenig

Ein Framework für Digital Rights Management Systeme

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang technische Informatik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Martin Hübner
Zweitgutachter: Prof. Dr.rer.nat. Gunter Klemke

Abgegeben am 17. Februar 2005

Christian Grewenig

Thema der Diplomarbeit

Ein Framework für Digital Rights Management Systeme

Stichworte

Digital Rights Management, Verschlüsselung, Kopierschutz, Urheberrecht, Rechteschutz, Framework, API, Mulit-Media-Inhalte.

Zusammenfassung

Anbieter von digitalen Inhalten stehen immer häufiger der Anforderung gegenüber, den Download ihrer Güter (Klingeltöne, Hintergrundbilder, etc.) zu kontrollieren, bzw. diesen zu schützen. Mit Hilfe des Digital Rights Management ist es möglich diesen Prozess zu steuern und die Inhalte zu schützen. Damit ist es Anbietern möglich für ihre Inhalte verschiedene Rechte zu erstellen und diese ihren Produkten zuzuordnen. Ein Framework stellt dem Anwender eine einheitliche Schnittstelle zur Verfügung und definiert damit Rahmenbedingungen zur Programmierung. Zum Erstellen und zum Ausliefern von geschützten digitalen Inhalten wird ein DRM-Framework entworfen, welches die verschiedenen DRM-Systeme unter einem einheitlichen API zugänglich macht.

Christian Grewenig

Title of the paper

Framework for digital rights management systems

Keywords

Digital rights management, encryption, decryption, copy-protection, copyright, legal protection, framework, API, Multi-media-content.

Abstract

The protection of digital content (ringtones, wallpaper, etc.) and control of these downloads is a challenge service providers are confronted anytime. Digital rights management provides mechanisms to control these processes and to protect the content. It provides the creation of digital rights and associates them to the content. A framework provides a structure of interfaces to which a programmer is bounded to. A DRM-Framework is implemented to protect and to deliver the digital content. It also provides access to all underlying DRM-Systems through an API.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung

Die Firma arvato mobile ist Dienstleister und hält digitale Inhalte für mobile Endgeräte bereit, welche über Partner-Firmen Online vertrieben werden. Digitale Inhalte lassen sich nach ihrer Art gruppieren, so z.B.:

- Klingeltöne (monophone, polyphone, mp3-audio,...),
- Logos (Hintergrundbilder im Display),
- Java-Spiele, etc.

Dazu stellt arvato mobile seinen Partnern die Inhalte in einem Katalog bereit, aus welchem sich die Partner bedienen, um gewünschte Inhalte auf ihren Internetseiten anzubieten. Der Vertrieb der Inhalte durch die Partnerfirmen auf ihren eigenen Internetseiten wird von arvato mobile, individuell je Partner entsprechend der verkauften Inhalte, auf Provisionsbasis vergütet.

Die Freischaltung der Inhalte erfolgt durch arvato mobile und richtet sich nach den expliziten Wünschen der Partner. Dazu realisiert arvato mobile für die Partner die notwendige technische Infrastruktur, damit diese die Inhalte auf ihren Seiten darstellen können.

Die Inhalte müssen für die unterschiedlichen herstellerspezifischen Endgeräte (Mobiltelefone, Personal Digital Assistant [PDA's], etc.) in einem individuellen Format vorliegen. Die Auslieferungsplattform von arvato mobile ermöglicht die Erfassung jeder Transaktion, die auf einer Partnerseite ausgelöst wird, nach Art des Inhaltes und des Endgeräte-Herstellers. Dazu wird jede ausgelöste Transaktion in einer Datenbank bei arvato mobile registriert und gestattet so eine individuelle Abrechnung.

Die Integration der Inhalte auf den Partnerseiten wird über ein entsprechendes Application Programmers Interface (API) vorgenommen, welches von arvato mobile eingesetzt wird. Die Internetseiten werden dabei als JSP-Seiten (Java Server Pages) realisiert und bei arvato mobile gehostet. Die Partnerfirmen binden diese Seiten dann als externe Seiten in ihre Internetpräsenz ein.

Bei den digitalen Inhalten handelt es sich um Wirtschaftsgüter, welche dem Benutzer nach einmaliger Bezahlung direkt auf sein mobiles Endgerät übertragen werden. Ob der digitale Inhalt nachfolgend an einem Dritten weitergegeben wird, entzieht sich der Kontrolle der Anbieter dieser Inhalte.

Nur mit direkt in den Endgeräten integrierten Mechanismen kann das Kopieren bzw. Weitergeben der Inhalte durch die Benutzer kontrolliert werden. So verfügen moderne Handys heutzutage verstärkt über DRM-Mechanismen (Digital Rights Management), mit welchen das Kopieren bzw. Weitergeben der Inhalte verhindert oder auch bewusst kontrolliert werden kann. Dazu sind die Inhalte in einen bestimmten Format zu übertragen, damit das Handy diese als einen geschützten Inhalt registrieren und dadurch besonders behandeln kann.

Dazu werden mobile Endgeräte zunehmend mit einen DRM-Client ausgestattet, um die Darstellung bzw. Nutzung von geschützten Inhalten zu ermöglichen. Dieser Client garantiert die Nutzung der Inhalte im Rahmen der definierten Nutzungsbedingungen durch den Anbieter. Aufgrund der vielfältigen Hersteller für Endgeräte werden auch heterogene DRM-Lösungen in den Endgeräten implementiert. Der Ablauf bei der Erstellung und Übertragung von geschützten Inhalten ist bei vielen DRM-Lösungen jedoch funktionsgleich.

Aufgrund der vielfältigen DRM-Lösungen ist es somit für die Auslieferungsplattform der Firma arvato mobile sinnvoll, möglichst flexibel bei der Wahl einer DRM-Lösung zu sein. Die Wahl einer einzelnen DRM-Lösung würde das Angebot auf eine zu geringe Anzahl von Endgeräten beschränken.

So ermöglichen einige Endgeräte, wie z.B. PDA's, auch das abspielen von Videodateien, welche ebenfalls mit Nutzungsbedingungen auszustatten sind. Die DRM-Lösungen in Handys gestatten dies jedoch nicht.

Die Entscheidung welches DRM-System zum Einsatz kommen soll muss im Moment der Bestellung erfolgen, nachdem klar ist um welches Endgerät und um welchen Inhalt es sich handelt.

Dies erfordert eine dynamische Nutzung verschiedener DRM-Systeme.

1.2 Zielsetzung

Die Benutzung eines bestimmten DRM-Systems entscheidet sich erst zum Zeitpunkt der Bestellung. Erst zu diesem Zeitpunkt ist der Auslieferungsplattform bekannt um welche Art von Endgerät und Inhalt es sich handelt. Das API der Auslieferungsplattform muss um ein weiteres API ergänzt werden, welche eine dynamische Nennung eines DRM-Systems gestattet. Da der Ablauf für die Erstellung und die Auslieferung von geschützten Inhalten bei nahezu jeder DRM-Lösung gleich ist, macht es Sinn eine einheitliche Zugriffsschicht zu erschaffen, mit der die einzelnen DRM-Lösungen angesprochen werden können.

Bei der Auslieferungsplattform handelt es sich um ein API, welches in Form eines Java-Packages zu benutzen ist. Es soll ein Framework entwickelt werden, welches Schnittstellen zum Erstellen von geschützten Inhalten und Nutzungsbedingungen definiert.

Die Auslieferungsplattform soll sich anhand dieses Frameworks für ein bestimmtes DRM-System entscheiden und dieses dann über die definierten Schnittstellen benutzen können.

Das Framework stellt konkrete Implementierungen der jeweiligen DRM-Systeme bereit und ist somit in der Lage die entsprechenden Inhalte und Nutzungsbedingungen zu erzeugen.

1.3 Aufbau der Arbeit

In dieser Arbeit geht es um die Entwicklung eines Frameworks für DRM-Systeme. **Kapitel 1** beschreibt die Problemstellung des Themas. Es wird erläutert warum und in welchem Umfeld ein Framework für DRM-Systeme benötigt wird. Es werden die Ziele definiert, die mit dem DRM-Framework erreicht werden sollen, um die beschriebene Problemstellung zu lösen.

Kapitel 2 erläutert Grundlagen die zum Verständnis von Digital Rights Management Systemen notwendig sind. Die durch das Internet neu entstandenen Vertriebswege und die daraus entstehenden Probleme werden dargestellt. Die Grundlage jedes DRM-Systems bilden kryptographische Verschlüsselungstechniken, welche anschließend beschrieben werden. Nach der grundlegenden Betrachtung der Komponenten von DRM-Systemen und deren Architektur werden drei existierende DRM-Systeme vorgestellt.

Das **dritte Kapitel** befasst sich mit der Definition des Begriffs Framework und den Anforderungen, die an ein DRM-Framework gestellt werden.

Die Konzeption eines DRM-Frameworks wird im **vierten Kapitel** erläutert. Es beschreibt die DRM-Referenz Architektur, welche als Grundlage für ein DRM-Framework genommen wird. Davon ausgehend wird eine Zwischenlösung entworfen, die den Anforderungen eines Frameworks noch nicht gerecht wird. Der Übergang zum endgültigen Framework bildet den Schluss dieses Kapitels, in dem alle Anforderungen abgedeckt werden.

Das **fünfte Kapitel** beschreibt die Implementierungsdetails des DRM-Frameworks. Nach einer kurzen Einführung der verwendeten Sprache, werden alle Software-Artefakte des Frameworks erläutert.

Den Schluss der Arbeit bildet eine Zusammenfassung und ein Ausblick im **sechsten Kapitel**.

2 Grundlagen Digital Rights Management

Die Digitalisierung und das Internet dürften für viele Anbieter von digitalen Inhalten wie Musik oder Videodateien ein Fluch und ein Segen zugleich sein. Durch das Internet ist ein Transportmedium entstanden, welches die verlustfreie Übertragung von Inhalten ermöglicht. Den Internetanwendern stehen damit alle Möglichkeiten offen eine Vielzahl von Kopien zu erstellen, ohne Qualitätsverluste in Kauf nehmen zu müssen.

Durch das Internet haben sich somit neue Vertriebswege ergeben. Diese ermöglichen Anbietern von digitalen Inhalten neue Formen der Vermarktung. So lassen sich z.B. Vorschau, Verleih und ähnliche Mechanismen leicht realisieren.

Die Nutzung dieser neuen Vertriebsmöglichkeiten erfordert auch eine Anpassung bisheriger Preismodelle. So ist anzunehmen, dass ein Benutzer für ein einzelnes Musikstück, welches er zur Vorschau einmalig anhören darf, weniger zu zahlen bereit ist, wie bei unbeschränkter Nutzungsdauer. DRM bedeutet in diesem Zusammenhang nicht einfach nur den Zugang zu den Inhalten zu erschweren, sondern auch neue Angebote zu unterschiedlichen Preisen anzubieten. Ohne DRM gibt es nur die Möglichkeit „Alles oder Nichts“ zu konsumieren.

Die Kopie spielt in der heutigen Zeit eine immer größere Rolle. Benutzer die ihre Musik oder Videos erworben haben speichern diese auf ihren PC oder anderen Home-Entertainment Systemen. Genutzt werden diese Inhalte allerdings auf einer Menge von Endgeräten wie z.B. MP3-Player, PDA’s, etc. D.h. der Benutzer kopiert diese Dateien auf sein bevorzugtes Endgerät und hört sich diese dort an. So stehen auch Dritten diese Inhalte ohne Einschränkungen zur Verfügung. Es ist eine Leichtigkeit Dateien unter verschiedenen Endgeräten auszutauschen (z.B. über eine Infrarotschnittstelle).

Die neuen Vertriebswege, welche sich aus den Endgeräten und den Transportmedium (Internet) ergeben, haben den Anbietern von digitalen Inhalten eine neue Sichtweise auf diese gegeben. Ein Musikstück wurde vor einigen Jahren ausschließlich im Kontext einer CD oder Schallplatte gesehen. Doch mit der Einführung von MP3-Playern und ähnlichen Geräten existiert das Musikstück auch ohne einen physikalischen Träger wie z.B. eine CD. Die CD wurde ursprünglich nicht dafür konzipiert in immer neuen Arten von Endgeräten benutzt werden zu können, sondern ausschließlich in CD-Spielern.

Ohne irgendeine Art von Management-Technologie lässt sich eine legale Übertragung (Sicherheitskopie, oder private Nutzungskopie) nicht von einer illegalen Kopie (Weitergabe an Dritte) unterscheiden.

2.1 Die Vorläufer in der Musikindustrie

Das Verständnis für Digital Rights Management erlangt man, wenn man den herkömmlichen Vertriebsweg in der Musikindustrie betrachtet und dabei auf Schwachstellen stößt. Dieser Abschnitt beschreibt den herkömmlichen Vertriebsweg der Musikindustrie und zeigt Schwachstellen auf, die sich durch moderne digitale Inhalte ergeben haben.

An der Wertschöpfungskette in der Musikindustrie gibt es unterschiedliche Teilnehmer:

- Den Künstler/Interpret und seine Partner (Manager, Komponisten, etc.)
- Die Plattenfirma. Das Unternehmen welches die Songs produziert und herausgibt. Verantwortlich für die Produktion, Vertrieb und Werbung.
- Die Anbieter/Verkäufer, welche für den Einzelhandel verantwortlich sind. Sie zahlen einen prozentualen Anteil ihres Profits zurück an das Label.
- Der Konsument/Endverbraucher, welcher für die CD oder Kassette bezahlt.

Der Künstler produziert sein Musikstück, um es auf den Markt zu bringen. Das Unternehmen welches ihm dabei hilft ist die Plattenfirma (Die Musik-Produzenten). Sie produzieren mit dem Künstler zusammen eine CD, oder DVD. Damit existiert ein Medium welches in großen Stückzahlen hergestellt werden kann und eine gute Qualität aufweist. Die Plattenfirmen investieren in die Vermarktung (Werbung, Vertrieb, etc.) dieses Produkts. Hierdurch gelangen die CD’s in die Regale der Einzelhändler, wo die Endverbraucher diese erwerben können.

Doch wie verteilt sich der Profit entlang dieser Kette? Der Endverbraucher, welcher die CD, oder DVD beim Einzelhändler erwirbt, bezahlt diesen für das Produkt. Der Einzelhändler wiederum zahlt davon einen gewissen Prozentsatz zurück an die Plattenfirma. Die Plattenfirma bezahlt davon die Lieferanten, Hersteller und nicht zuletzt auch den Künstler, welcher von jedem verkauften Musikstück einen prozentualen Anteil bekommt (unabhängig vom Format).

Jetzt betrachten wir, wie diese Kette durch einen Dritten gestört werden kann und wie dieser die Erlöse innerhalb der Wertschöpfungskette beeinflusst. Als erste Schwachstelle wäre eine Kopie der Original-CD zu sehen, welche auf dem Weg vom Studio zur CD Produktion erstellt und somit in Umlauf gebracht werden könnte. Der Besitzer dieser Kopie wäre damit in der Lage das komplette Album im Internet zu veröffentlichen, oder es als weitere Kopie an Dritte weiterzugeben. Das bedeutet natürlich einen erheblichen Verlust für die Plattenfirma und den Künstler. Genauso könnten Aufnahmen auch auf dem Weg zum Einzelhändler gestohlen werden.

Nicht zuletzt ist natürlich der Endverbraucher selbst zu nennen, der eine Schwachstelle darstellt. Dem Endverbraucher ergeben sich mit dem Erwerb

unterschiedliche Nutzungsrechte. Auf Grund dessen ergibt sich für den Endverbraucher:

- Er kann das Musikstück hören ohne ein zweites oder drittes Mal dafür bezahlen zu müssen.
- Er kann das Album verleihen. In diesem Fall übergibt er vorübergehend die Nutzungsrechte an Dritte. Er erlangt diese erst mit der Rückgabe wieder.
- Er könnte auch Kopien anfertigen.
- Oder er vernichtet es, wenn ihm danach ist.

Daraus ergibt sich die Zielsetzung, dass ein Endverbraucher an der Nutzung seiner Kopie gehindert werden muss, solange er dafür nicht bezahlt hat. Das gebräuchlichste Szenario in diesem Zusammenhang ist das ein Verbraucher ein Produkt erwirbt, davon Kopien erstellt und diese dann an Dritte weiterverteilt, welche nicht für diese Kopie bezahlt haben (auf legalem Wege, der Wertschöpfungskette entsprechend). Die Weitergabe von Kopien schließt die Plattenfirmen und auch den Künstler bei weiteren Veräußerungen aus.

Es gibt jedoch noch weitere Aspekte, die man berücksichtigen sollte. Es ist nicht gerade einfach Kopien anzufertigen, die exakt die gleiche Qualität wie ihre Originale besitzen. In manchen Fällen bedarf es sogar eines hohen Aufwands, gerade wenn spezielles Equipment zum Einsatz kommt. Das Anfertigen einer Kopie einer DVD z.B. bedarf momentan eines recht hohen finanziellen Aufwands, wenn man die gleiche Qualität wie das Original erreichen will. Hingegen das Kopieren von CD auf eine gewöhnliche Audio-Kassette ist sehr günstig, geht allerdings auch mit einem erheblichen Verlust der Qualität einher.

Die Verteilung der Kopien von Hand zu Hand erfordert immer die Anwesenheit einer Person und ist somit ziemlich zeitaufwändig. Diese Methode der Verbreitung illegaler Kopien war jahrelang die „Normalität“. Durch den Einsatz des Internets ist dieser Zeitaufwand auf ein Minimum reduziert worden.

Unauthorised Internet Sites (Figs IFPI, Oct 2002)		
Web/FTP	200,000 pirate sites	100 million unauthorised music files
Peer to Peer Networks (Simultaneous users and files. Figs IFPI, Oct 2002)		
Service	Users	Files
KaZaA	2.74 million	481 million
IMesh	927,000	162 million
OpenNap (153 servers)	404,000	204 million

Gnutella (inc Morpheus)	112,000	19.6 million
All services	4.5 million	900 million

Abbildung 2.1: Kopien und Massenverbreitung ohne Berechtigung. Quelle [Becker-2003]

Einen Einblick in die zunehmende Verbreitung von Kopien über das Internet gibt Abbildung 2.1. Der Nutzen von Internet Sharing ist für die Endverbraucher viel höher als die traditionelle Verbreitung von Hand zu Hand. Gründe dafür sind z.B.:

- Die Tools, die das Erstellen von Kopien ermöglich, sind über das Internet frei erhältlich oder sehr kostengünstig.
- Das Kopieren mit dem PC ermöglicht eine viel bessere und genauere Qualität als analoge Mechanismen.
- Der Austausch von Kopien kann ohne das Verlassen der eigenen vier Wände geschehen. Die meiste Arbeit nehmen einem sogar die Sharing-Programme direkt ab.
- Das Publikum, welches an diesen Tausch teilnimmt ist erheblich größer.
- Mit der Einführung von Breitband Internet ist der Austausch sehr viel schneller geworden.

Die Kombination der oben genannten Tatsachen macht Content Sharing über das Internet zu einem sehr verbreiteten Phänomen. Die Musikindustrie zieht daraus ihre Schlüsse und reagiert entsprechend, um ihren geistigen Eigentum vor illegaler Verbreitung zu schützen. Daraus ergibt sich für die Musikindustrie einen wachsenden Bedarf an DRM-Systemen.

2.2 Kryptographische Grundlagen

Der Schutz der durch DRM-Systeme erreicht wird, basiert auf kryptographische Verschlüsselungstechniken. Dabei werden die digitalen Inhalte verschlüsselt und ausgeliefert. Der so verschlüsselte Inhalt kann nur durch den Erwerb eines Rechteobjektes, welches den Schlüssel beinhaltet geöffnet und betrachtet werden. Die folgenden Abschnitte erläutern die aktuellen Verschlüsselungstechniken, die bei DRM-Systemen zum Einsatz kommen.

2.2.1 Was ist Kryptographie

Durch kryptographische Verfahren lassen sich viele Sicherheitsprobleme verringern, die bei der Übertragung von Daten existieren. Durch den Einsatz von Verschlüsselungstechniken kann erreicht werden, dass Daten während der Übertragung nicht mitgelesen werden können und so vor Dritten geschützt werden.

Kryptographie bedeutet nicht die Nutzung eines geheimen Algorithmus. Diese Art der Sicherheit ist sehr unzuverlässig. In der Vergangenheit sind solche

Mechanismen tatsächlich zum Einsatz gekommen [Becker-2003]. Solche Mechanismen lassen sich nicht standardisieren und gewährleisten keine Qualität. Damit sind diese Verfahren nutzlos.

Kryptographie basiert vielmehr auf der Grundlage von Schlüsseln. Ein Schlüssel ist dabei eine geheime Information welche von Algorithmen zur Ver- und Entschlüsselung benutzt werden. Gerade diese Separierung von Schlüsseln und Algorithmen macht es möglich, das unterschiedliche Teilnehmer dieselben Algorithmen mit unterschiedlichen Schlüsseln verwenden können.

[Fuhrberg-2001] beschreibt Verschlüsselung als ein Prozess bei dem eine offene Nachricht unter Verwendung von bestimmten Algorithmen und mit Hilfe eines Schlüssels (z.B. eine bestimmte Zeichenkette), in eine verschlüsselte Nachricht transformiert wird, die keine Rückschlüsse auf den ursprünglichen Inhalt zulässt. Entschlüsselung ist der umgekehrte Prozess, bei dem aus der verschlüsselten Nachricht, durch einen identischen Algorithmus und unter Verwendung eines Schlüssels, die offene Nachricht wieder gewonnen werden kann.

Der Schlüssel stellt meistens ein sehr langes Bitmuster dar (und damit eine große Zahl). Der Raum aller möglichen Zahlen, die als Schlüssel verwendet werden können, bezeichnet man als Schlüsselraum. Die Sicherheit die moderne Algorithmen bieten, basiert nicht auf der Geheimhaltung der Algorithmen selbst, sondern auf der Geheimhaltung der Schlüssel und die Größe des Schlüsselraums. Die Algorithmen sind für Jedermann frei zugänglich.

Alle Anforderungen die an eine Verschlüsselung gestellt werden lassen sich auf vier Grundkonzepte zurückführen [Fischer-1998].

Vertraulichkeit

Die wichtigste Forderung an den Verschlüsselungsverfahren ist die Gewährleistung der Vertraulichkeit der Daten. Die Daten sollen nur den Personen zugänglich sein, die autorisiert sind diese zu lesen. Zusätzlich kommt noch die Wahrung der Anonymität gegenüber Dritten mit ins Spiel.

Integrität

Die Integrität der Daten hat erst mit Einführung der elektronischen Nachrichtenübertragung an Bedeutung gewonnen. Zu Zeiten als Nachrichten noch per Postkarte ausgetauscht wurden, gab es noch keinen Anlass für die Gewährleistung der Integrität. Die Originalität der Daten, bzw. des Textes muss nicht gesondert gesichert werden, da es entsprechend aufwändig ist diese zu ändern. Es Bedarf zum Beispiel der gleichen Handschrift. Bei der elektronischen Nachrichtenübermittlung muss die Originalität der Daten gewährleistet werden. Die Daten werden als eine Folge von Binärinformationen übertragen. Es ist für den Empfänger nicht möglich herauszufinden, ob es sich dabei um dieselben Daten handelt die der

Absender abgeschickt hat. Moderne Verschlüsselungstechniken müssen also gewährleisten, dass die Daten nicht geändert werden können.

Authentizität

Die Gewährleistung der Authentizität von Daten ist ein entscheidendes Sicherheitskriterium. Kryptographische Verfahren müssen es ermöglichen den Sender einer Datei zweifelsfrei zu identifizieren. Das ist gerade mit Hinblick auf den Abschluss rechtskräftiger Verträge durch den Austausch elektronischer Dokumente wichtig.

Beweisbarkeit von Transaktionen

Mit der Verbreitung des elektronischen Zahlungsverkehrs gewinnt die Nachweisbarkeit von einmal durchgeführten Transaktionen zunehmend an Bedeutung. Jedoch gehört zu einer Transaktion meistens nicht nur ein einzelnes Dokument. Die klassischen Verfahren zur Verschlüsselung lösen dieses Problem nicht ohne entsprechende Erweiterungen und werden in dieser Arbeit nicht weiter erläutert.

2.2.2 Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung (auch Secret-Key-Verfahren genannt) verwendet zum Ver- und Entschlüsseln von Daten den gleichen Schlüssel. Dieser Schlüssel muss zwischen den einzelnen Kommunikationspartnern auf sicherem Wege ausgetauscht werden. Die Sicherheit, die diesem Verfahren zugrunde liegt basiert allein auf der Geheimhaltung des Schlüssels. Abbildung 2.2 illustriert diesen Zusammenhang.

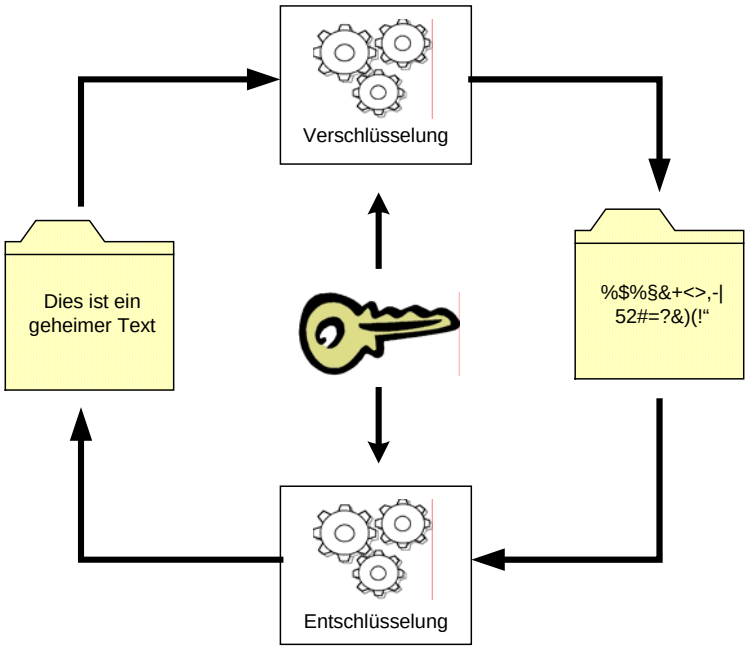


Abbildung 2.2: symmetrische Verschlüsselung.

Als Beispiel für symmetrische Verschlüsselung ist die Caesar-Chiffre zu nennen. Bei diesem Verschlüsselungsverfahren handelt es sich um eine Substitutions-Chiffre. Die wesentliche Eigenschaft einer Substitutions-Chiffre ist, dass jeder Buchstabe durch einen anderen Buchstaben ersetzt wird.

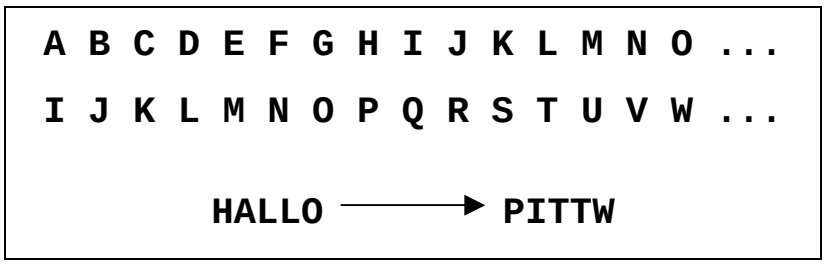


Abbildung 2.3: Beispiel für ein Substitutionsverfahren.
Quelle [Fischer-1998]

Das Verfahren ist allerdings leicht zu „brechen“, da für jeden Buchstaben im Klartext immer auch der gleiche Buchstabe im Chiffretext verwendet wird. Bestimmte Buchstaben kommen in einem Text häufiger vor als andere. Durch eine Häufigkeitsanalyse ist es somit möglich den Klartext zu ermitteln.

Symmetrische Verschlüsselungsverfahren lassen sich in zwei Kategorien einteilen.

- *Blockalgorithmen*.
  Hierbei werden Blöcke von Daten gleichzeitig verschlüsselt.
- *Bitstromverschlüsselungsverfahren*.
  Der offene Text wird Bit für Bit verschlüsselt.

Blockverschlüsselungen verarbeiten immer Klartextdaten gleicher Länge. Dabei wird die gleiche Anzahl von Bits verschlüsselt. Die Blockalgorithmen lassen sich in verschiedenen Modi betreiben.

Der einfachste Modus ist der *Electronic Code Block* (ECB) Modus. Dieser besagt, dass es für jeden Block des offenen Textes einen Block mit verschlüsseltem Text gibt. Wie in einen Wörterbuch lässt sich jeder verschlüsselte Text für einen offenen Text nachschlagen. Als ein Nachteil des ECB Modus ist die Tatsache zu nennen, dass eine dritte Person bei einer derart verschlüsselten Nachricht Blöcke entfernen oder die Reihenfolge verändern kann. Er kann auch die Blöcke verschiedener Nachrichten zusammenwürfeln, falls jeweils der gleiche Schlüssel verwendet wurde [Schmeh-2001].

Ein weiterer Modus ist der *Cipher Block Chaining* (CBC). Bei diesem Modus wird zunächst jeder offene Block mit dem vorangegangenen verschlüsselten Block durch eine XOR Operation verknüpft. Somit können in einer Nachricht beliebig viele gleiche Blöcke auftreten, ohne dass diese bei der Betrachtung des Chiffre-Textes auffallen.

DES

Der *Data Encryption Standard* (DES) ist der wohl populärste Verschlüsselungsalgorithmus. DES wurde 1976 als amerikanischer Standard veröffentlicht [Beutelsbacher-2000]. Beim DES handelt es sich um einen Blockalgorithmus. Er verwendet einen 64 Bit langen Schlüssel wovon allerdings nur 56 Bit zur Verschlüsselung verwendet werden. Der Klartext muss als eine Folge von Bits vorliegen. Dieser Klartext wird dann in Blöcken zu je 64 Bit eingeteilt, welche dann der Reihe nach verschlüsselt werden. Dazu wird zunächst einmal der zu verschlüsselnde 64-Bit Block initial permutiert (IP). Der resultierende 64-Bit Strom wird dann in zwei Teile L (für Links) und R (für Rechts) aufgeteilt, die aus je 32-Bit bestehen. Anschließend wird 16-mal der gleiche Vorgang wiederholt (man spricht von 16 Runden des DES):

- Auf L wird die Funktion F angewendet. Deren Ergebnis wird mit R exklusiv-oder-verknüpft, das Ergebnis dieser Verknüpfung wird zum neuen L.
- L wird zum neuen R.

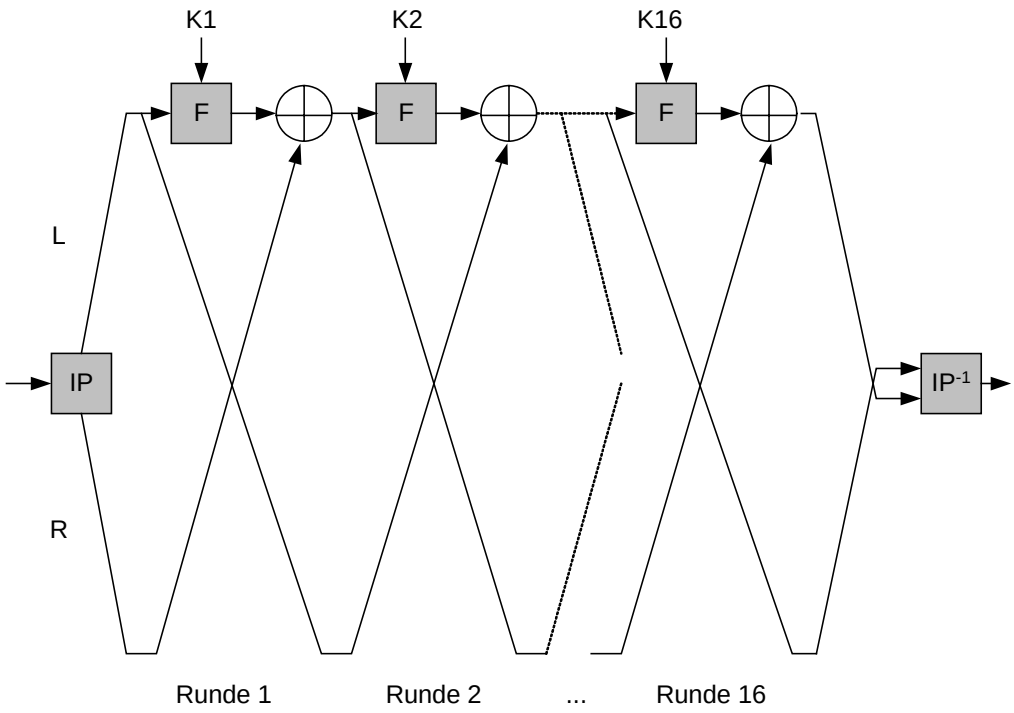


Abbildung 2.4: Die 16 Runden des DES.
Quelle [Schmeh-2001]

Zum Schluss werden die 64 resultierenden Bits noch einmal permutiert (IP^{-1}), und zwar genau umgekehrt wie bei der Anfangspermutation.

Das Verfahren ist auf Grund des zu kurzen Schlüssels (effektiv nur 56 Bit) leicht angreifbar. Aus diesem Grund raten viele Experten von diesen Verfahren ab.

IDEA

Der *International Data Encryption Algorithmus* (IDEA) wurde 1990 in der Schweiz entwickelt. Auch der IDEA ist ein Blockalgorithmus mit einer Blockgröße von 64 Bit und einer Schlüssellänge von 128 Bit. Der IDEA ist im Vergleich zum DES ein noch recht neues Verfahren, über den recht wenig Untersuchungen veröffentlicht worden sind. Durch den großen Schlüsselraum von 2^{128} möglichen Schlüsseln ist es sehr unwahrscheinlich diesen durch einfaches Probieren zu erraten [Fuhrberg-2001].

Bei diesen Algorithmus werden jeweils 64 Bit Klartext zu 64 Bit Chiffretext verschlüsselt. Dabei greift IDEA auf einfache Modulo-Arithmetik zurück. Von Interesse ist allerdings nicht das Ergebnis, sondern der Rest dieser Operation. IDEA führt dabei acht Verschlüsselungsrunden durch.

AES

Neben IDEA und DES gibt es noch den *Advanced Encryption Standard* (AES) Algorithmus. AES (oder auch Rijndael genannt) wurde als Nachfolger des DES bestimmt. Der Algorithmus wurde als Sieger eines dreijährigen Wettbewerbs um einen neuen Krypto-Standard der US-Standardisierungsbehörde NIST[1] verkündet.

Bei AES handelt es sich ebenfalls um einen Blockalgorithmus, welcher mit einer variablen Schlüssel- und Blocklänge arbeitet. Die Länge muss lediglich ein Vielfaches von 32 Bit entsprechen. Bei steigender Rechenleistung kann so die Schlüssellänge entsprechend angepasst werden, so dass auch in Zukunft mit einem starken Schlüssel gearbeitet werden kann, welcher dann nicht die Schwachstelle darstellt. AES lässt sich sowohl in Hardware, als auch in Software implementieren [Fuhrberg-2001].

2.2.3 Asymmetrische Verschlüsselung (Public-Key)

Die asymmetrischen Verschlüsselungsverfahren benötigen, im Gegensatz zu symmetrischen Verfahren, keinen sicheren Kanal zur Übertragung des Schlüssels. Die Entwicklung dieses Verfahrens wurde Mitte der 70er Jahre begonnen. Es basiert auf der Erkenntnis, dass man Schlüsselpaare verwenden kann, die sich nicht voneinander ableiten lassen [Fischer-1998]. Diese Schlüssel hängen zwar über mathematische Verfahren voneinander ab, bei einem gut entwickelten asymmetrischen Verfahren kann aber trotz der Kenntnis eines Schlüssels der andere nicht abgeleitet werden.

Ein Teilnehmer besitzt bei diesen Verfahren jeweils ein Schlüsselpaar. Dieses Paar besteht aus einem privaten und einem öffentlichen Schlüssel (Public Key). Der private Schlüssel wird geheim gehalten, während der öffentliche Schlüssel jedem bekannt sein kann.

[1] National Institute of Standards and Technology

Der Einsatz der asymmetrischen Verschlüsselung lässt sich mit folgendem Ablauf beschreiben:

- Beide Teilnehmer erzeugen jeweils ein Schlüsselpaar (privater Schlüssel und öffentlicher Schlüssel).
- Teilnehmer A übermittelt seinen öffentlichen Schlüssel an B.
- Dieser Schlüssel wird von Teilnehmer B zur Chiffrierung der Nachricht benutzt, und übermittelt die verschlüsselte Nachricht an A.
- A entschlüsselt die Nachricht mit seinem privaten Schlüssel.
- Teilnehmer B behält den öffentlichen Schlüssel von A für weitere Nachrichten.
- Sobald Teilnehmer A eine Nachricht an B senden will, benötigt er analog dessen öffentlichen Schlüssel.

Die Übertragung der Nachricht ist gesichert, weil der zum Entschlüsseln (private Key) benötigte Schlüssel nicht mit übertragen wird. Ausschließlich der öffentliche Schlüssel wird übertragen, welcher nur zum Verschlüsseln verwendet wird. Aus diesem Grunde wird kein sicherer Kanal zur Übertragung benötigt.

Diffie-Hellmann

Die amerikanischen Wissenschaftler Whitfield Diffie und Martin Hellman stellten 1975 ein mathematisches Verfahren vor, welches den diskreten Logarithmus zur Lösung des Schlüsselaustauschproblems verwendet [Schmeh-2001].

Dieses Verfahren basiert auf einer verbesserten Einwegfunktion, die einfach zu berechnen, umzukehren jedoch praktisch unmöglich ist. Diese Funktion benutzt eine Zusatzinformation, die es möglich macht, die eigentlich unmöglich zu lösende Umkehrfunktion zu berechnen. Diese Zusatzinformation bildet den privaten Schlüssel, während der öffentliche Schlüssel die Einwegfunktion ausmacht.

Der diskrete Logarithmus bietet eine solche Einwegfunktion (auch Falltürfunktion genannt). Der oben beschriebene Ablauf einer Nachrichtenübermittlung lässt sich mathematisch folgendermaßen erklären:

Beide Teilnehmer erstellen ein Schlüsselpaar (öffentlicher Schlüssel und privater Schlüssel). Dazu erstellt jeder Teilnehmer eine Primzahl p und eine natürliche Zahl g, die kleiner ist als p. Diese beiden Informationen bilden den öffentlichen Schlüssel. Als privaten Schlüssel denken sich beide Teilnehmer eine weiter natürliche Zahl aus, die kleiner ist als p. Teilnehmer A denkt sich x und Teilnehmer B denkt sich y aus.

- A berechnet die Zahl $a = g^x \bmod p$ und schickt diese an B.
- B berechnet die Zahl $b = g^y \bmod p$ und schickt b an A.

- A kann nun $k_1 = b^x \bmod p$ berechnen.
- B berechnet analog $k_2 = a^y \bmod p$.

Die Formeln für k_1 und k_2 können nun wie folgt umgeformt werden:

- $k_1 = b^x \bmod p = (g^y \bmod p)^x \bmod p = g^{yx} \bmod p$
- $k_2 = a^x \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$

Diese Umformungen zeigen, dass beide Teilnehmer mit dem selben Schlüssel arbeiten, ohne das dieser ungesichert übertragen wurde.

[Fischer-1998] bemängelt, dass beide Teilnehmer aktiv an der Schlüsselgenerierung teilnehmen müssen, was eine Email-Verschlüsselung sehr unkomfortabel macht.

Je größer die verwendete Zahl (Länge des Bitmusters), desto sicherer ist das Verfahren, aber auch aufwendiger [Schmeh-2001].

RSA

Die Entwickler Ronald Rivest, Adi Shamir und Len Adleman veröffentlichten 1977 ein weiteres Verfahren, welches zu einem Quasistandard im Internet wurde [Fuhrberg-2001]. Das Verfahren beruht auf der Eigenschaft, dass es einfach ist zwei Primzahlen zu finden und das Produkt zu bilden. Umgekehrt ist aber kein Algorithmus bekannt, der für eine gegebene Zahl in akzeptabler Zeit die Primfaktoren ermitteln kann.

RSA funktioniert wie folgt:

- Teilnehmer A wählt zwei große Primzahlen *p* und *q*, die sich in ihrer Länge deutlich unterscheiden müssen. A berechnet dann das Produkt $n = p*q$. Die unterschiedliche Länge ist deswegen notwendig, da sie sonst aus *n* bestimmt werden können, indem in der Umgebung von $\sqrt{n}$ alle Primzahlen getestet werden.
- A wählt seinen öffentlichen Schlüssel *e* so, dass *e* und *(p-1)*(q-1)* keinen gemeinsamen Primfaktor außer 1 haben.
- Jetzt muss noch eine weitere Zahl *d* erzeugt werden, bei der gelten muss, dass *(e*d-1)* ohne Rest durch *(p-1)*(q-1)* teilbar ist.
- Aus diesen Zahlen bildet sich das Schlüsselpaar. Der öffentliche Schlüssel besteht aus den Zahlen *n* und *e*, während sich der private Schlüssel aus den Zahlen *n* und *d* zusammensetzt.
- Die Verschlüsselung berechnet sich dann mit der Formel $c = m^e \bmod n$, wobei *m* die zu verschlüsselnde Nachricht und *c* die verschlüsselte Nachricht ist.
- Der Empfänger kann nun wieder aus *c* und seinem privaten Schlüssel die Originalnachricht *m* ermitteln. Dazu wird die oben erwähnte Formel zur Entschlüsselung umgewandelt und lautet: $m = c^d \bmod n$.

Die Sicherheit bei RSA wird durch die Auswahl geeigneter Primzahlen sichergestellt. Dabei ist wichtig, dass sie einerseits wirklich zufällig gewählt werden und sie müssen andererseits lang genug sein, um bei steigender Rechnerleistung auch in 20 Jahren noch einer Primfaktorzerlegung standhalten zu können. Die Nachrichten, die mit diesen Verfahren verschlüsselt werden können, dürfen nicht länger als *n* sein. Dafür wird man im Normalfall die Nachrichten in Blöcke zerlegen, um diese Bedingungen zu erfüllen.

Aktuelle Implementierungen asymmetrischer Verschlüsselungsverfahren unterstützen derzeit Schlüssellängen zwischen 512 Bit und 2048 Bit. Man hat errechnet, dass der finanzielle Aufwand um einen 512 Bit langen Schlüssel zu brechen weniger als 1Mio $ beträgt und diese Größe weiter abnehmen wird. Deshalb wird als Mindestschlüssellänge für RSA mittlerweile 1024 Bit angegeben.

2.3 DRM-Systeme

2.3.1 Grundlegende Betrachtungen

Digital Rights Management soll in Zukunft dafür sorgen, dass sich digitale Inhalte in der Praxis wie ein anderes Verbrauchsgut verhalten und nicht in beliebiger Anzahl ohne größeren Aufwand massenweise Kopien in Originalqualität hergestellt werden können.

Ein DRM-System setzt sich dabei nicht einfach nur aus einem Kopierschutzsystem zusammen, sondern enthält weitere technische Elemente, welche im Ergebnis die Verwaltung und Verrechnung der in Anspruch genommenen digitalen Werke erlaubt. [Becker-2003][2] beschreibt ein DRM-System bestehend aus folgenden Komponenten:

- Benutzeridentifizierung/Authentifizierung
- Verschlüsselung
- Kopiersperre oder –kontrolle
- Zugangs- und Nutzungskontrolle
- digitale Wasserzeichen
- digitale Fingerabdrücke
- manipulationssichere Hardware
- Zahlungssysteme

[2] Auch zu finden unter:
http://www.datensicherheit.nrw.de/Daten/WS06122002/guennewig.pdf

Digital Rights Management beschränkt sich dabei nicht ausschließlich auf die Zurverfügungstellung über das Internet. Ebenso können digitale Inhalte auf CD's, oder DVD's durch DRM-Systeme geschützt und verwaltet werden.

Benutzeridentifizierung

Der Austausch von digitalen Inhalten verlangt nach sicheren Übertragungseinheiten („*secure containers*" – [Becker-2003]). Es werden nicht nur die digitalen Güter ausgetauscht, sondern es „fließen" entsprechende Geldbeträge. Es ist nicht einfach nur der sichere Transfer digitaler Inhalte der im Mittelpunkt steht, sondern auch der sichere Transfer von Nutzungs- und Kopierrechten.

Der Schutz von Kopierrechten endet nicht zuletzt mit der erfolgreichen Übertragung von Inhaltsbezogenen Informationen. Nach der Übertragung dieser Informationen ist insbesondere die Nutzungskontrolle digitaler Inhalte wichtig. Diese Kontrolle muss solange gewährleistet werden wie die Inhalte im Besitz einer Person sind.

Bei Authentifizierung unterscheidet man „User authentication" und „Message authentication". Folgendes Beispiel (aus [Becker-2001]) verdeutlicht die Bedeutung der Benutzeridentifizierung und der Authentifizierung:

Ein Benutzer kontaktiert einen Multimedia Server und beabsichtigt eine Musikdatei herunterzuladen. Der Server versichert sich durch Benutzeridentifizierung, dass es sich um den Benutzer Alice handelt, welcher die entsprechende Berechtigung besitzt diese Datei herunterzuladen. Anschließend bekommt der Server weitere Anfrage von Alice, die weitere Dateien herunterladen möchte. Der Server übermittelt den zu bezahlenden Preis und die entsprechende Bankverbindung für die Bezahlung.

Durch *message authentication* kann Alice sich versichern, dass die Daten die Ihr durch den Server übermittelt wurden stimmen und dass das Geld nicht an einen Hacker übermittelt wird, welcher ihr falsche Informationen anstelle die des Servers zukommen lassen hat.

Verschlüsselung

Die in Abschnitt 2.2 erläuterten kryptographischen Verfahren werden für die Verschlüsselung von digitalen Inhalten bei Online-Übertragungen angewendet.

Für individualisierte und insbesondere kostenpflichtige Dienste sollten, wenn immer möglich, die Mediendaten verschlüsselt werden. Damit sind sie vor unberechtigtem Zugriff auf dem Übertragungsweg geschützt.

Kopiersperre oder –kontrolle

Das zugrunde liegende DRM-System muss in der Lage sein, Mechanismen zur Verfügung zu stellen, die die Kontrolle der Nutzung regeln. Diese hat zu gewährleisten, dass die digitalen Inhalte, die von einer Person bezogen

wurden, auch nur dieser einen Person zugänglich sind. Die Weitergabe dieser Inhalte muss unter Kontrolle des Anbieters bleiben.

Bei mobilen Endgeräten, wie z.B. Handys, gibt es den Anwendungsfall der Superdistribution (siehe auch Abschnitt 2.3.5). Dieser Fall beschreibt die Möglichkeit der Weitergabe des digitalen Inhaltes an Dritte. Die Weitergabe kann zwar gestattet sein jedoch muss der Empfänger seine eigene Lizenz erwerben. Durch diesen Mechanismus behält der Anbieter die Kontrolle der Weitergabe und bezieht trotz Kopie den entsprechenden Gegenwert.

digitale Wasserzeichen/ digitale Fingerabdrücke

Durch kryptographische Verfahren können digitale Inhalte soweit geschützt werden, dass der Benutzer diese nur in Verbindung mit einer gültigen Nutzungslizenz betrachten kann. Die Nutzungslizenz beinhaltet im Allgemeinen den symmetrischen Schlüssel für die Entschlüsselung. Auch die sichere Übertragung der Daten ist somit gewährleistet.

Neben der eigentlichen Verschlüsselung von digitalen Inhalten ist das digitale Wasserzeichen ein weiterer Schutz. Der eigentliche Inhalt in nicht verschlüsselter Form lässt sich durch ein digitales Wasserzeichen mit weiteren Informationen versehen. Diese Informationen können Urheber, Anbieter, Content-ID, etc. sein.

Digitale Wasserzeichen verändern den eigentlichen Inhalt dahingehend das es visuell oder akustisch nicht möglich ist diese Veränderung wahrzunehmen. Die Veränderung muss so robust erfolgen, dass es nicht möglich ist, diese durch Manipulation unberechtigt zu entfernen. Dabei sind verschiedene Manipulationen denkbar: Analog-Digital-Wandlung, Digital-Analog-Wandlung, Abspielgeschwindigkeit, Farbtiefe, Kompression, Verzerrung, Ausschneiden von Bildteilen.

Das digitale Wasserzeichen ist vergleichbar mit der Steganographie. Bei der Steganogrphie wird versucht eine Botschaft vor dem Zugang Unbefugter zu schützen. Für den nicht eingeweihten Betrachter ist es nicht erkennbar, dass eine versteckte Botschaft überhaupt vorhanden ist. Als Grundlage hierfür dient das sogenannte Datenrauschen. Das bedeutet, dass elektronische Daten einer gewissen Fehlertoleranz unterliegen. Dementsprechend kann man gewisse Datenformen (Audiodateien und Bilder) leicht manipulieren und so seine Daten unterbringen, ohne dass das Gesamtbild bzw. der Ton sich verändert.

manipulationssichere Hardware

Der Aufwand der Schutzmechanismen, die in einem manipulationssicheren System eingesetzt werden, hängt von den geschätzten Fähigkeiten und Mitteln des Angreifers ab.

Ein hardwarebasierter Schutzmechanismus lässt sich zum Beispiel durch Einchip-Systeme realisieren. Diese Systeme präsentieren sich als ein isolierter Baustein ohne Systembusleitungen. Die Hersteller begründen mit

genau dieser Tatsache auch die Sicherheit dieser Systeme. Es zeigt sich allerdings bei Pay-TV-Piraten, das auch solche Systeme nicht sicher sind. Es ist durchaus möglich – mit entsprechenden Aufwand – geheime Daten aus diesen Systemen auszulesen.

Zahlungssysteme

Seit Einführung der Digitalisierung und der Verbreitung von Kommunikationsnetzen ist eine Vielzahl von elektronischen Zahlungssystemen entstanden. Dabei werden monetäre Werte auf elektronischem Wege ausgetauscht. Die Entwicklung auf diesem Gebiet schreitet ständig voran (bzw. wird ständig intensiviert). Das Hauptaugenmerk wird nach [Becker-2003] auf folgende Bereiche gelegt:

- *Security aspects*: Das Hauptproblem bei traditioneller Bezahlung sind gefälschte Geldscheine. Mit der Entwicklung von elektronischen Bezahlverfahren soll eine höhere Sicherheitsstufe erreicht werden.
- *Commerce over communication networks*: Die Abwicklung einer Geldwert-Transaktion zwischen zwei Geschäftspartnern, welche miteinander über ein Netzwerk kommunizieren, erlaubt keinen Austausch von monetären Werten auf traditionelle Weise. Weil die Geschäftspartner sehr weit voneinander entfernt sein können ist der gegenseitige Austausch, monetärer Werte über ein Netzwerk notwendig.

DRM-Systeme erlauben die Abrechnung nach Benutzung („Pay-per-Use“) oder nach Zahlung einer monatlichen Pauschale ("Subscription").

2.3.2 DRM-Referenz Architektur

Die DRM-Referenz Architektur beschrieben in [Rosenblatt-2001] besteht im wesentlichen aus drei Komponenten. Es gibt den *Content Server*, welcher digitale Inhalte vorhält. Vom *License Server* erhält ein Benutzer die Nutzungsrechte zum Betrachten der Dateien. Der *Client* bildet das Endgerät (z.B. ein Handy, PDA, oder PC) mit dem der Benutzer die Dateien betrachtet, bzw. verwertet.

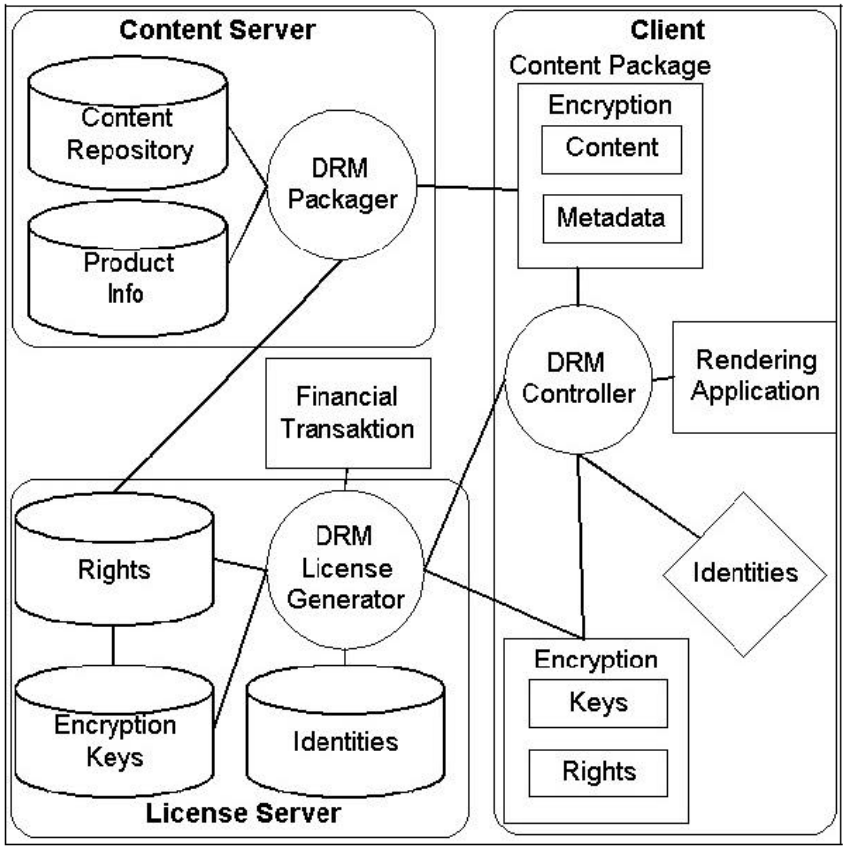


Abbildung 2.5: Die DRM-Referenz Architektur
Quelle [Rosenblatt-2001]

Die DRM-Referenz Architektur beschreibt nur die wesentlichen Komponenten. Nahezu alle modernen DRM-Systeme entsprechen dieser Architektur (siehe hierzu auch Abschnitt 2.3.4-2.3.6). Abhängig vom System-Design gibt es bei einigen DRM-Systemen unterschiedliche, bzw. zusätzliche Komponenten.

Der Ablauf einer Transaktionen geschieht in folgenden Schritten:

1. Ein Benutzer tätigt eine Anfrage an einem Web- oder FTP-Server. Er lädt sich verschlüsselte digitale Inhalte inklusiv Metadaten auf sein Endgerät herunter.
2. Durch „Doppelklick“ wird eine automatische Anfrage gestellt. Diese aktiviert den DRM-Controller. Einmal aktiviert, sucht der DRM-Controller die nötigen Informationen um die gewünschte Lizenz ausstellen zu können.
3. Der DRM-Controller sendet die Identität des Benutzers und des digitalen Inhaltes (*Content Package*) zum *License Server*.
4. Der Lizenz Server identifiziert den Benutzer mit Hilfe der Identities-Datenbank.
5. Die Rechte werden gemäß der Anfrage des Benutzers erfasst und in der Rights-Datanbank festgehalten.
6. Sollte für diese Transaktion eine Zahlung notwendig sein, so wird diese entsprechend ausgelöst (*financial Transaction*).

7. Die Lizenz wird durch den *DRM License Generator* erstellt und gleichzeitig auch verschlüsselt.
8. Dem Benutzer wird die Lizenz zugeschickt.
9. Der DRM-Controller empfängt die Nutzungslizenz und entschlüsselt diese. Danach wird der empfangene Inhalt dem Wiedergabegerät freigegeben.
10. Der Client kann den Inhalt nutzen.

Das oben erläuterte Referenzmodell unterstellt, dass die Kommunikation zwischen dem *Client* und dem *DRM License Generator* einem bestimmten Protokoll unterliegt. Die Daten die zwischen diesen beiden Komponenten ausgetauscht werden beschreiben ein bestimmtes Rechte Protokoll, welches durch die *Rights Expression Language (REL)*standardisiert ist.

2.3.3 Rights Expression Language (REL)

Viele Organisationen und Unternehmen haben die Notwendigkeit für eine standardisierte Rechte Sprache (REL) erkannt und haben entsprechende Dialekte entworfen. Eine Rechte-Sprache bietet eine Möglichkeit Zugriffsrechte zu formulieren.

Eine REL muss bestimmte technische Voraussetzungen erfüllen, damit die oben erwähnte Funktionalität realisiert werden kann. Die wohl wichtigste Voraussetzung besteht in der Maschinenlesbarkeit. Aus diesem Grund werden alle modernen REL's in XML formuliert. XML Dokumente können von maschinen gelesen und interpretiert werden. Dadurch ist dieses Format so erfolgreich und dient als allgemeines Daten-Austauschformat.

Eine REL sollte es gestatten Zugriffsrechte, Bezahlungsdetails, Sicherheitsinformationen und Verschlüsselungsinformationen zu formulieren [Becker-2003].

Die Anforderungen an eine REL sind von Anwendung zu Anwendung sehr verschieden. Von daher ist es erforderlich, dass die REL erweiterbar ist.

Nach [Becker-2003] besteht ein REL aus folgenden drei Komponenten:

- *Rights*. Hierbei handelt es sich um ein Element zum ausdrücken von Zugriffsrechten. Zugriffsrechte können detaillierter beschrieben werden als Vorbedingungen, die notwendig sind um ein Recht überhaupt gewähren zu können.
- *Assets.* Diese Komponente repräsentiert den eigentlich digitalen Inhalt, welchen es zu schützen gilt.
- *Party.* Hierbei kann es sich um irgendeinen Teilnehmer handeln z.B. eine Person oder ein Objekt. Als Personen wären z.B. der Autor, Hersteller, Content Provider oder Verbraucher zu nennen.

ODRL

Bei Open Digital Rights Language handelt es sich um einen freien Standard für eine REL. Das root element bildet das *rights* element. Es drückt ein bestimmtes Recht aus.

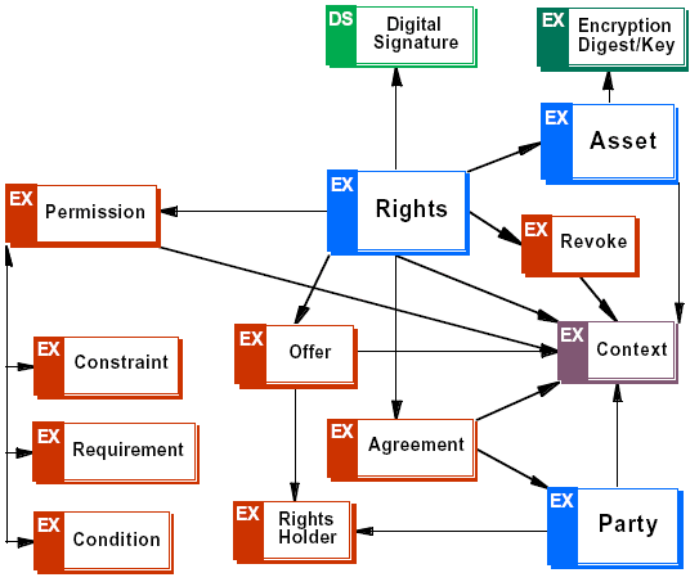


Abbildung 2.6: ODRL Foundation Model
Quelle [odrl.net]

Wie in Abbildung 2.6 zu sehen ist, besteht auch ODRL aus den zuvor genannten drei Komponenten.

Die ODRL stellt die Grundsemantik zur Verfügung, aus welcher, angepasst an die Bedürfnisse der unterschiedlichen Industriebereiche (wie z.B. eBooks, Musik, Software etc.), einzelne kompatible Sprachen zur Rechteverwaltung und -wahrung geschaffen werden können. ODRL ist eine offene freie Sprache und dem W3C als Working Group-Basis vorgeschlagen.

2.3.4 DRM für mobile Endgeräte

Mit Einführung der GPRS (*General Packet Radio Service*) Technologie ist die Nutzung des Internets auf mobilen Endgeräten wie Handys oder PDA's möglich. Damit sind fast alle Inhalte, die sich auf einem PC nutzen lassen auch auf einem mobilen Endgerät nutzbar. Das Internet weitet sich mit dieser Technologie immer mehr auf den mobilen Sektor aus. Es ist für den Verbraucher möglich E-Mail, Web-Browsing und MMS zu nutzen. Er kann so Daten zwischen mobilen Endgerät und Internet austauschen.

Dieser Trend ist schon soweit fortgeschritten, dass z.B. Handys nicht mehr einfach nur zum Telefonieren genutzt werden, es handelt sich bei diesen Geräten vielmehr um Multimedia Geräte. Diese besitzen farbige Displays, können mehrstimmige (polyphone) Klingeltöne abspielen und verfügen über integrierte Kameras.

Aus diesem Grund finden Angebote für mobile Endgeräte wie Klingeltöne, farbige Bilder Filme, etc. immer mehr Absatz. Auch Informationsdienste wie Nachrichten oder aktuelle Sportereignisse sind über solche Geräte abrufbar.

Für diese digitalen Inhalte besteht ein großer Schutzbedarf durch DRM Technologie. DRM Technologie findet somit immer mehr Verwendung bei mobilen Endgeräten.

Die DRM Techologie, die auf mobilen Endgeräten genutzt wird und die, die im Internet Verwendung findet, unterscheiden sich in gewissen Punkten. Zum einen sind es die reinen Unterschiede zwischen stationären Geräten wie PC's und mobilen Geräten wie Handys, zum anderen der Unterschied zwischen der Zuordnung zu einer bestimmten Person. DRM-Systeme im Internet und somit auf PC's sind nach [Becker-2003] weit weiniger erfolgreich als auf mobilen Endgeräten. Es wird damit begründet, das bei PC's kein fälschungssicherer Speicher zur Verfügung steht. Die Architektur eines PC's und der verwendeten Betriebssysteme ist wohl bekannt und Angriffe auf DRM-Systeme auf einem PC sind entsprechend erfolgreich.

Die Hersteller mobiler Endgeräte entwickeln dagegen ihre eigenen Betriebsysteme und sind auch in der Lage die Hardware so zu gestalten, dass selbst fälschungssichere Speicher möglich sind. Somit ist es weitaus schwieriger diese Systeme zu umgehen.

Ein weiterer Unterschied besteht in der Zuordnung eines Benutzers zu einem Gerät. Bei einem Handy ist immer eine bestimmte Person mit diesem Gerät assoziiert. Das ist ein großer Unterschied zu einem PC, der von vielen verschieden Personen genutzt werden kann. Für die Vertrauensbasis in einem DRM-System ist das von Vorteil.

Ein weiterer wichtiger Aspekt ist die Interoperabilität. Es ist ungünstig wenn ein Verbraucher digitale Inhalte bei Provider A bezieht und diese nur auf Gerät X verwenden kann; und zusätzlich noch Inhalte von Provider B bezieht und diese wiederum nur mit Gerät Y verwenden kann. Es bedarf einem einheitlichen Standard, der die Zusammenarbeit der verschiedenen DRM-Systeme untereinander garantiert. Eine Spezifikation eines solchen Standards, hat die *Open Mobile Alliance* (*OMA*) herausgebracht (siehe Seite 23).

Die Möglichkeit qualitativ hochwertige digitale Inhalte auf mobilen Endgeräten zu verwenden obliegt dem Wunsch diese untereinander auszutauschen. Wenn einen Verbraucher der Klingelton eines Bekannten gefällt sollte die Möglichkeit bestehen, dass Benutzer A den Klingelton zu Benutzer B transferieren kann. Dieser Transfer kann durch eine Kabelverbindung, eine kabellose Verbindung (z.B. Bluetooth) oder MMS geschehen. Der Empfänger muss sich allerdings ein Nutzungsrecht besorgen, damit er den Inhalt nutzen kann. Dadurch wird der Anbieter von digitalen Inhalten nicht außen vor gelassen und erhält den entsprechenden Gegenwert. Des Weiteren ist die Last der Datenübertragung auf den Servern des Anbieters vermindert worden.

Die Übertragung fand ausschließlich zwischen den beiden Endverbrauchern statt (Peer-To-Peer Verbindung).

In den nächsten Abschnitten folgt eine Darstellung verschiedener DRM-Systeme. Hauptaugenmerk für diese Arbeit liegt dabei auf das DRM-System der Open Mobile Alliance.

2.3.5 Open Mobile Alliance

Die Open Mobile Alliance (kurz OMA) hat einen DRM-Standard für mobile Services herausgebracht. Die OMA besteht aus ca. 250 verschiedenen Unternehmen.

Durch den starken Wachstum in der Telekommunikations Branche ist ein Bedarf für ein DRM-Standard entstanden. Dehalb haben sich Firmen wie: Ericsson, Motorola, Nokia, Openwave, Siemens, Sony Ericsson und Vodafone zusammengesetzt und einen Vorschlag für ein DRM-Standard beim WAP Forum[3] eingereicht. Zeitgleich wurde durch die 3GPP[4] ein weiterer DRM-Standard vorgeschlagen. Alle Bemühungen wurden schließlich unter der Schirmherrschaft der OMA zu einem DRM-Standard vereint.

Hauptziel der OMA ist es ein Standard zu schaffen der einfach und in kurzer Zeit zu implementieren ist. Aus diesem Grunde wurde die Spezifikation in einem Bottom-Up Verfahren entwickelt, bei dem zuerst die grundlegenden Mechanismen definiert wurden. Weitere Mechanismen wurden später hinzugefügt.

Der OMA DRM-Standard [openmobilealliance.org] wurde gegen Ende 2002 herausgegeben. Die Spezifikation besteht dabei aus drei Dokumenten. Diese beschreiben die allgemeine Architektur, die *Rights Expression Language* (REL), und das DRM Container Format.

Die Spezifikation beschränkt sich dabei auf das Verpacken des digitalen Inhaltes und der Rechtedefinition, weniger auf die sichere Übertragung. Dieser Aspekt wird in der Spezifikation klar definiert. Der Fokus liegt dabei auf mobile Inhalte und weniger auf hochwertige digitale Inhalte, wie Filme oder Musik. Für diesen Bereich wird auf zukünftige Versionen verwiesen.

Der OMA Standard [OMA DRM Architecture] beschreibt in seiner Spezifikation drei unterschiedliche Auslieferungsmechanismen: *forward-lock*, *combined delivery* und *separate delivery*. Ein Endgerät, welches den OMA Standard unterstützen möchte, muss zumindest den Mechanismus *forward-lock* bereitstellen. Die folgenden Mechanismen bauen aufeinander auf und sind optional.

---

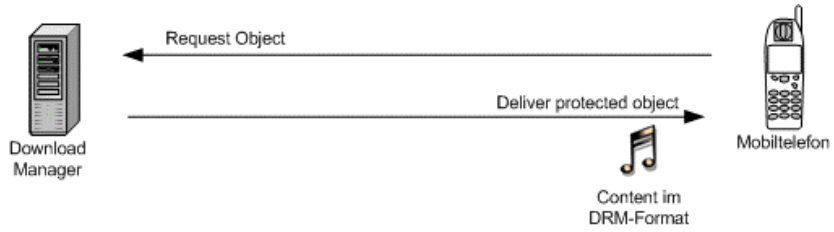
[3] http://www.wapforum.org
Das WAP Forum ist Bestandteil der Open Mobile Alliance.

[4] http://www.3gpp.org

forward-lock

Der Auslieferungsmechanismus *forward-lock* ist der kleinste gemeinsame Nenner des OMA-Standards. Hierbei werden die Daten nicht verschlüsselt und sind im Prinzip ungeschützt. Der Mechanismus sagt nichts weiter aus, als dass der Inhalt nicht weitergegeben werden darf. Dabei werden die Daten in ein anderes Format überführt. Dieses Format nennt sich DRM-Message und ist im Prinzip nichts weiter als eine Multipart-Message. Die eigentlichen Nutzdaten (Ein Bild, oder Klingelton) werden dabei binär oder Base64 encoded eingebettet.

Das Endgerät muss sicherstellen, dass ein empfangenes Objekt diesen Formates nicht weitergegeben werden kann.



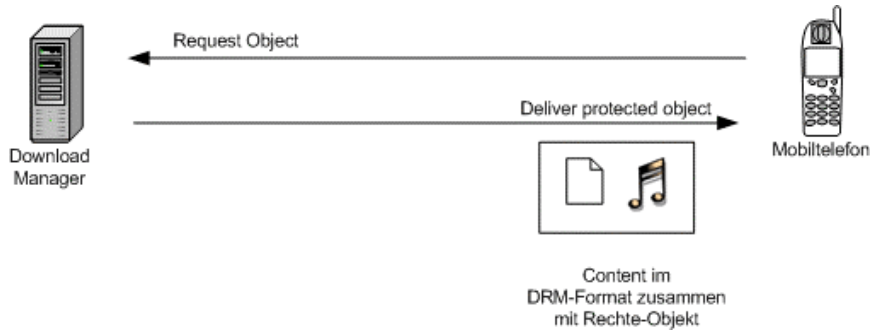

```
HTTP/1.1 200 OK
Content-type: application/vnd.oma.drm.message;
              boundary=boundary-1
Content-Length: 622

--boundary-1
Content-type: audio/midi
Content-Transfer-Encoding: binary

…jpeg image in binary format…
--boundary-1--
```

Abbildung 2.7: Forward-Lock

Beim *forward-lock* stellt das Endgerät (In diesem Fall ein Mobiltelefon) ein Request. Der Server (Download Manager) sorgt dafür, dass der Klingelton in das DRM-Message Format überführt wird. Abschließend wird das Objekt zum Mobiltelefon übertragen.

combined-delivery

Combined-delivery erweitert den *forward-lock* Mechanismus. Zusätzlich wird ein Rechtebereich zur DRM-Message hinzugefügt. Damit steht dem Nutzer dieses Mechanismus eine feinere Abstufung der Nutzungsrechte zur Verfügung. Der Rechtebereich wird mit Hilfe der *Rights Expression Language* formuliert; er liegt als Klartext vor. Abbildung 2.8 zeigt ein Beispiel.




```
HTTP/1.1 200 OK
Content-type: application/vnd.oma.drm.message;
              boundary=boundary-1
Content-Length: 1012

--boundary-1
Content-type: application/vnd.oma.drm.rights+xml
Content-Transfer-Encoding: binary

<o-ex:rights
   xmlns:o-ex=”http://odrl.net/1.1/ODRL-EX”
   xmlns:o-dd=”http://odrl.net/1.1/ODRL-DD”>
   <o-ex:context>
      <o-dd:version>1.0</o-dd:version>
   </o-ex:context>
   <o-ex:agreement>
      <o-ex:asset>
         <o-ex:context>
            <o-dd:uid>cid:example001@DRMprovider.biz</o-dd:uid>
         </o-ex:context>
      </o-ex:asset>
      <o-ex:permission>
         <o-dd:play/>
      </o-ex:permission>
   </o-ex:agreement>
</o-ex:rights>
--boundary-1--
Content-type: audio/midi
Content-ID: <example001@DRMprovider.biz>
Content-Transfer-Encoding: binary
…jpeg image in binary format…
--boundary-1--
```

Abbildung 2.8: Combined-Delivery

Genau wie beim *forward-lock* Mechanismus wird zuerst eine Anfrage durch ein Mobiltelefon gestellt. Der Server erstellt ein Nutzungsrecht und formuliert dieses in REL. Aus Abbildung 2.8 ist ersichtlich, dass der eigentliche Inhalt mit dem Rechteobjekt durch eine UID fest assoziiert wird. Die so erweiterte DRM-Message wird anschließend zum Mobiltelefon geschickt. Das Mobiltelefon trägt dafür Sorge, dass der Klingelton nicht weitergegeben werden kann. Das

Nutzungsrecht, welches in REL der DRM-Message hinzugefügt wurde, zeigt dem Mobiltelefon in welcher Art und Weise das Bild verwendet werden darf.

separate-delivery

So wie *combined-delivery* den *forward-lock* Mechanismus erweitert, so erweitert *separate-delivery* den *combined-delivery* Mechanismus. Der wesentliche Unterschied hierbei ist, dass das Rechteobjekt separat vom eigentlichen Inhalt übertragen wird.

Die digitalen Inhalte werden bei dieser Auslieferungsart verschlüsselt. Hierzu wird der AES Algorithmus mit einer Schlüssellänge von 128Bit und *Cipher Block Chaining* (siehe Kapitel 2.2) verwendet. Das Format, welches den verschlüsselten Inhalt kapselt, nennt sich DRM Content Format (*DCF*). Das DCF Objekt wird durch den Server direkt zum Mobiltelefon ausgeliefert. Das Rechteobjekt hingegen wird in einem weiteren anschließenden Schritt via WAP-Push übermittelt.

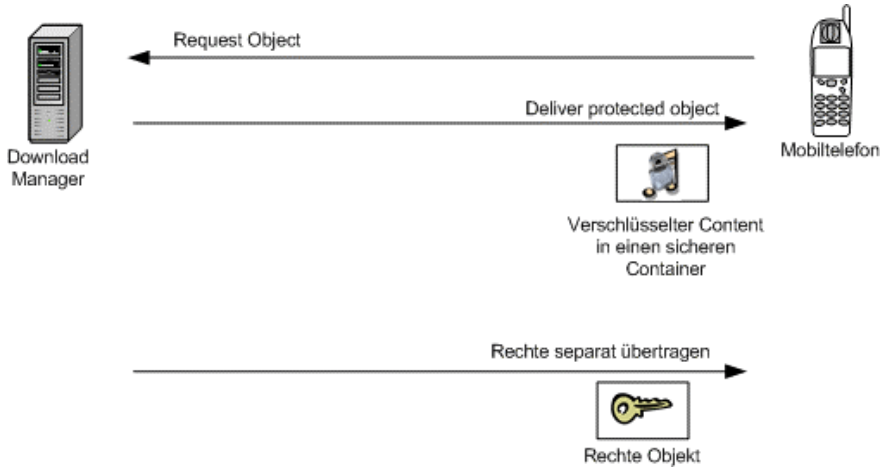


Abbildung 2.9: Separate-Delivery

Erst jetzt ist das Mobiltelefon in der Lage den Inhalt zu nutzen, entsprechend der im Rechteobjekt formulierten Nutzungsbedingungen. Der Anwender kann den Inhalt auch an ein weiteres Mobiltelefon transferieren, der Empfänger muss jedoch ebenfalls erst ein Nutzungsrecht erwerben. Diesen Vorgang kann er nach Erhalt des Inhaltes durchführen. Damit unterstützt der OMA Standard den Mechanismus der *Superdistribution* (siehe Kapitel 2.3.4).

2.3.6 Microsoft – Windows Media Rights Manager

Das Unternehmen Microsoft ist eines der führenden Anbieter von Digital Rights Management Systemen. Microsoft hat mit dem Windows Media Rights Manager ein System geschaffen, das sehr verbreitet ist.

Der Windows Media Rights Manager verwendet das Format *Windows Media File* (wm, wma, wmv). Die neueste Version des Windows Media Rights Managers enthält sowohl eine Server-, als auch eine Client-Komponente. Beides ist im mitgelieferten *Software Development Kit* (SDK) enthalten. Microsoft verteibt den Windows Media Rights Manager kostenlos. Man muss jedoch eine Lizenz geben lassen und diese jährlich erneuern

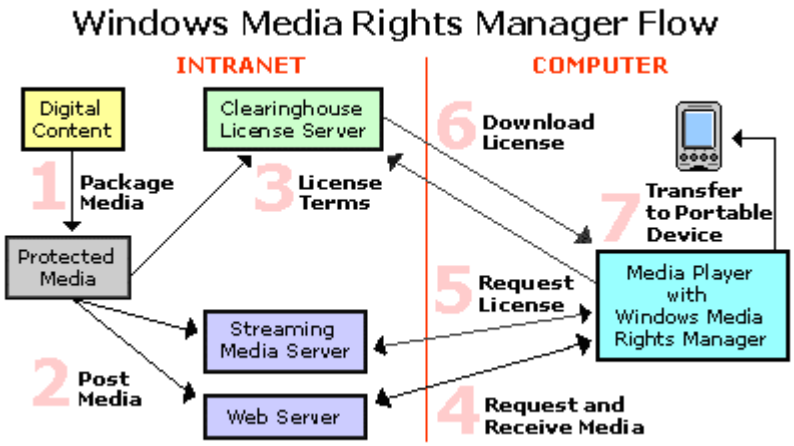


Abbildung 2.10: Windows Media Rights Manager
Quelle [microsoft.com]

Der Ablauf ist stark an die DRM-Referenz Architektur (siehe Kapitel 2.3.2) angelehnt. Der eigentliche Inhalt wird verschlüsselt und in ein bestimmtes Format überführt. Nachdem der Client das Objekt erhalten hat, muss erst ein Rechteobjekt erworben werden, um den Inhalt nutzen zu können.

Der Ablauf beim Windows Media Rights Manager ist wie folgt [microsoft.com]:

1. *Packaging*. Bei diesem Vorgang wird der Inhalt verschlüsselt. Der Schlüssel wird in einem separatem Rechteobjekt hinterlegt. Die Datei die an dieser Stelle generiert wird erhält die Endung .wmv (Windows Media Video), oder .wma (Windows Media Audio). In dieser Datei sind noch zusätzliche Informationen enthalten, wie z.B. eine URL von der sich das Rechteobjekt beziehen lässt.
2. *Distribution.* Die verschlüsselte Datei kann anschließend auf einem Server zum Download bereitgestellt werden. Es ist ebenfalls denkbar, das die Datei auf CD gebrannt werden kann, oder per E-Mail verschickt wird. Der Austausch dieser Datei ist gestattet.
3. *Establishing a License Server.* Der Anbieter von Inhalten, welche durch Windows Media Rights Manager geschützt sind, richten ein License Server (auch Clearing House genannt) ein. Dieser Server übernimmt die separate Auslieferung der Rechteobjekte.
4. *License Aquisition.* Sobald der Benutzer versucht den erworbenen Inhalt zu nutzen, muss er für diesen erst eine Nutzungslizenz erwerben. Dazu wird der Benutzer zu einer URL (hinterlegt im verpackten Inhalt) weitergeleitet, bei der er eine Lizenz erwerben kann.
5. *Playing the media file.* Damit der Benutzer die Datei abspielen kann benötigt er eine Abspielsoftware, welche den Windows Media Rights Manager unterstützt. Die Datei wird dann entsprechend der Nutzungslizenz abgespielt.

2.3.7 Adobe Content Server

Als digitale Inhalte lassen sich nicht nur Filme oder Musikdateien bezeichnen. Auch Texte liegen in digitaler Form vor. Die Firma Adobe ist durch das weit verbreitete Dateiformat *Portable Document Format* (PDF) bekannt geworden. Adobe stellt zum Darstellen dieser Datei eine frei verfügbare Software zur Verfügung (Adobe Acrobat Reader). Zur Erstellung von Dateien diesen Formates benötigt man den Acrobat Writer, welcher kostenpflichtig ist.

Der große Vorteil dieses Formats liegt darin, dass das Layout, die Schriften und die Grafiken bei der Transformation beibehalten werden. Somit ist das Format plattformübergreifend nutzbar.

Durch den *Adobe Content Server* ist die Firma Adobe auch auf dem Markt der DRM-Systeme vertreten. Damit lassen sich Dokument als *electronic Book (eBooks)* über das Internet vertreiben. Weil das PDF-Format ein Read-only Format ist, besitzt es schon eine gewisse Nutzungseinschränkung und ist somit gut für DRM Zwecke geeignet.

Der Adobe Content Server besteht aus:

- Repository,
- Packager
- Distribution-Server
- GBLink-Modul

Das Repository stellt eine Datenbank dar. Dieses muss vor der ersten Nutzung von Adobe authorisiert werden.

Eine in das PDF Format umgewandelte Textdatei, wird durch den Packager des Content Servers verschlüsselt und verpackt es zusammen mit einem Bild des Covers zu einem eBook.

Bei diesem Prozess sind zwei eBook-Formate möglich:

- *eBoox Exchange (EBX)*. Dieses Format wurde durch das Open eBook Forum vorgeschlagen. Folgende Rechte können damit spezifiziert werden:
  - Lesen, innerhalb eines bestimmten Zeitraumes.
  - Kopieren von Teil- Ausschnitten eines eBooks in die Zwischenablage (evtl. innerhalb eines bestimmten Zeitraumes, oder eine bestimmte Anzahl von Kopien).
  - Drucken von Seiten aus dem eBook (evtl. innerhalb eines bestimmten Zeitraumes, oder eine bestimmte Anzahl von Seiten).
  - Ausleihen des eBooks und Weitergabe des Verleihrechtes.

  Das EBX Format erlaubt die freie Auswahl der Länge des Schlüssels beim Packaging Prozess.

- *PDF Merchant*. Für die Erstellung dieses Formates benötigt man eine ID und eine Lock-Datei, die eine digitale Signatur enthält, welche bei

Adobe erhältlich ist.
Dieses Format gestattet eine etwas andere Art der Rechtevergabe: Drucken, Kopieren, Bearbeiten, Kommentieren, in ein anderes Format umwandeln und jedes dieser Rechte ändern.

Nachdem das PDF Dokument in eins der Formate überführt wurde, steht es auf einem Web Download Server zum Download bereit.

Die Content ID und Nutzungsbedingungen werden in einer eBook Datenbank gespeichert. Der Zugriff zu diesen Nutzungsrechten erfolgt über den Fullfillment-Server (Rechte Server).

Ein Kunde erfragt ein Nutzungsrecht über eine Webseite eines Anbieters. An dieser Stelle bezahlt der Kunde das eBook. Anschließend erstellt der Server des Anbieters ein Dokument, welches die Kauferlaubnis enthält, und schickt es dem Kunden.

Der eBook Reader des Kunden schickt dieses Dokument wieder zurück zum Server des Händlers. Jetzt wird eine Nutzungslizenz generiert welche dem Kunden zurückgeschickt wird.

Der Kunde ist an dieser Stelle in der Lage, sich das eBook vom Download Server des Händlers herunterzuladen.

Der Rechte-Server schickt eine Bestätigung über die erfolgreiche Transaktion an einen eCommerce Server des Händlers.

Dem Kunden steht die Erlaubnis zu das eBook an Dritte weiter zu geben. Diese müssen dann ebenfalls eine Nutzungslizenz erwerben.

3 Anforderungen an ein DRM-Framework

3.1 Definition

Ein Framework besteht aus einer Menge von Objekten, die eine generische Lösung für eine Reihe verwandter Probleme implementieren. [wikipedia.org] beschreibt ein Framework als ein teilweise komplettes Software Sub-System, welches realisiert werden muss. Es definiert die grundlegende Architektur und legt fest an welcher Stelle gewünschte Anpassungen vorgenommen werden müssen.

[Gamma-2001] definiert ein Framework als „eine Menge kooperierender Klassen, welche die Elemente eines wiederverwendbaren Entwurfs für eine bestimmte Art von Software darstellen. Ein Framework bietet eine Architekturhilfe beim Aufteilen des Entwurfs in abstrakte Klassen und beim Definieren ihrer Zuständigkeiten und Interaktionen. Ein Entwickler passt das Framework für eine bestimmte Anwendung an, indem er Unterklassen der Frameworklassen bildet und ihre Objekte zusammensetzt."

In [thefreedictionary.com] wird ein Framework als eine Struktur beschrieben, die ein Gerüst umschließt, welches als Basis zur Konstruktion benutzt wird.

Die einheitliche Behandlung aller möglichen DRM-Systeme innerhalb einer Anwendung, lässt sich demnach durch ein Framework realisieren. Das Framework definiert grundlegende Arbeitsabläufe durch entsprechende Schnittstellen und vorgegebene Komponenten.

Der Einsatz eines Frameworks erleichtert die Programmierung. Unterschiedliche DRM-Systeme lassen sich durch ein Framework als ein homogenes System ansprechen. Man kann so auf fertige Dienste und Bausteine zurückgreifen und ist nicht gezwungen, jedes DRM-System individuell mit unterschiedlichen Schnittstellen zu implementieren.

Die Nutzer dieses Frameworks sind Anwendungsprogrammierer, die die Auslieferungsplattform täglich in neue Applikationen integrieren. Die Nutzung des DRM-Frameworks ist für die Anwendungsprogrammierer transparent. Die Auslieferungsplattform unterscheidet selbstständig, ob es sich um geschützten, digitalen Inhalt handelt und dieser vorher durch das DRM-Framework verschlüsselt werden muss, oder ob es sich um gewöhnlichen digitalen Inhalt handelt, der direkt ausgeliefert werden kann.

Durch den Einsatz eines Framework erhöht sich die Produktivität für die Programmierer, die dadurch nicht jedes einzelne DRM-System explizit in die Auslieferungsplattform integrieren müssen. Ein Framework schirmt viele Funktionen (wie z.B. Datenbankzugriffe) ab, mit denen sich der Programmierer nicht beschäftigen muss. Es wird ferner ein einheitliches Programmiermodell geschaffen, an das sich alle Nutzer des Frameworks halten. Das erhöht die Übersichtlichkeit einzelner Projekte ➔ Wirtschaftlichkeit.

3.2 Anforderungen

Das DRM-Framework soll dem Anwendungsprogrammierer Aufgaben abnehmen und ihn bei der täglichen Arbeit unterstützen. In diesem Abschnitt definiere ich Anforderungen an ein DRM-Framework und lege somit auch die Zuständigkeiten der einzelnen Komponenten fest.

1. *Einheitliche Schnittstellen*. Das DRM-Framework wird durch Komponenten gebildet, wobei jede einzelne Komponenten klar definierte Aufgaben übernimmt. Das Zusammenwirken der Komponenten untereinander wird durch Schnittstellen definiert. Der Anwendungsprogrammierer ist damit an ein Grundgerüst gebunden.
2. *Integration in die Auslieferungsplattform*. Das Framework sollte sich leicht in die bestehende Auslieferungsplattform integrieren lassen. Die vorhandenen Inhalte sollen durch das DRM-Framework entsprechend geschützt werden können. Dazu ist es wichtig Komponenten zu definieren, die ihre eigenen Zuständigkeiten besitzen.
3. *Einheitliche Formulierung von Nutzungsbedingungen*. Jedes DRM-System besitzt die Möglichkeit Lizenzen auszuliefern. Diese werden auf unterschiedlichste Art und Weise in den einzelnen DRM-Systemen generiert. Hierbei ist es wichtig, dass die Formulierung der Nutzungsbedingungen auf eine einheitliche Art vorgenommen wird.
4. *Speichern von Schlüsseln*. Jedes DRM-System schützt seine Inhalte durch kryptographische Verschlüsselungstechniken. Die dabei verwendeten Schlüssel müssen zwischengespeichert werden, damit bei einem späteren Abruf der Nutzungsbedingungen der Schlüssel herausgegeben werden kann.
5. *Erweiterbarkeit*. Weitere DRM-Systeme müssen sich leicht in das DRM-Framework einfügen lassen. Das DRM-Framework muss eine transparente Nutzung der unterschiedlichen DRM-Systeme gestatten.
6. *OMA DRM*. Als Referenzimplementierung soll das OMA DRM-System realisiert werden. Die Auslieferung von Nutzungsrechten und Inhalten soll separat geschehen. Deshalb ist das Modell *separate Delivery* (siehe Kapitel 2.3.5) zu implementieren.
7. *Benutzerverwaltung (optional)*. Jeder Anwender der durch DRM geschützte Inhalte von Anbietern bezieht, sollte bei einer späteren erneuten Transaktion wieder erkannt werden. Die Identifizierung kann beispielsweise durch die Mobilfunknummer geschehen. Dadurch können einzelne Transaktionen einen bestimmten Benutzer zugeordnet werden. Im Falle einer Superdistribution wäre somit eine Provisionierung für den Benutzer möglich, der als erstes den Inhalt bezogen hat.

Im folgenden Abschnitt erläutere ich die einzelnen Komponenten und deren Schnittstellen. Dabei sollen alle eingangs erläuterten Anforderungen abgedeckt werden.

4 Konzeption eines DRM-Frameworks

Die Darstellung der DRM-Referenz Architektur aus Kapitel 2.3.2 beschreibt die grundlegenden Komponenten, aus denen ein DRM-System besteht. Zusammengefasst besteht es aus den Komponenten *Client*, *Content-Server* und *License-Server*.

Der Content-Server benutzt dabei eine Packager-Komponente, welche für die Verschlüsselung der digitalen Inhalte verantwortlich ist. Die Integration der Packager-Komponente in den Content-Server ist im Prinzip eine Erweiterung. Der Content-Server ist ein eigenständiger Server zur Auslieferung der digitalen Inhalte. Das Schützen der Inhalte wird dabei an die Packer-Komponente delegiert.

Der License-Server ist ebenfalls ein eigenständiger Server, welcher ausschließlich die Auslieferung der Rechte übernimmt. Der *Client* initiiert Anfragen beim *Content-Server* und beim *License-Server*. Die Auslieferungsplattform der Firma „arvato mobile“ kapselt alle Anfrage an den Content-Server und dient als API für die Anwendungsentwickler. Die Integration eines DRM-Systems in die Auslieferungsplattform sollte somit als Erweiterung angesehen werden, welche alle DRM-spezifischen Aufgaben kapselt. Mit der Einführung eines Frameworks für DRM wird somit ein API zur Erweiterung der Auslieferungsplattform geschaffen.

Die vorgestellten DRM-Systeme aus Kapitel 2.3.5 – 2.3.7 bestehen aus den beschriebenen Komponenten der DRM-Referenz Architektur. Damit dient die DRM-Referenz Architektur als Basis für die Entwicklung des DRM-Frameworks.

In den folgenden Kapiteln werde ich erläutern, welche Betrachtungsweisen es für Rechtemodelle gibt. Danach zeige ich den Entwurf einer Lösung, welche sich an die DRM-Referenz Architektur hält und Verbesserungsmöglichkeiten offen hält. Der Übergang zur eigentlichen Lösung erfolgt im Anschluss. Bei der Beschreibung der endgültigen Lösung für das DRM-Framework gehe ich auf die einzelnen Bestandteile der gewählten Architektur ein. Dabei gehe ich auf das API und dessen Funktionsweise ein.

4.1 Rechtemodelle

Für die Definition der Nutzungsrechte sind verschiedene Rechtemodelle denkbar. Ein DRM-System formuliert die Nutzungsbedingungen vorzugsweise in einer beschreibenden Sprache wie ODRL. Damit sind Formulierungen möglich, die jede Art der Nutzung beschreiben können.

Denkbar sind z.B. die einmalige Nutzung eines Klingeltons zur Vorschau, oder eine zeitlich begrenzte Nutzung.

Die Handhabung der Nutzungsrechte kann grundlegend auf unterschiedliche Art und Weise realisiert werden. Dabei könnte zu jedem einzelnen Produkt eine eigene Nutzungslizenz erstellt werden, oder generelle Nutzungslizenzen, wobei die Schlüssel diesen Lizenzen zugeordnet werden.

4.1.1 Produkt - Lizenz

Die Zuordnung eines einzelnen Produktes zu einer Nutzungsbedingung bedeutet, dass jede einzigartig ist und einen eigenen Schlüssel verwendet. Für dieses Modell ist eine Vielzahl von Schlüsseln erforderlich, die bei jeder neuen Inhalterstellung erzeugt werden müssen. Der Schlüssel wird dann unter der Produktnummer in einer Datenbank hinterlegt und die Nutzungsbedingung in einer weiteren Tabelle assoziiert mit der Produktnummer.

4.1.2 Schlüssel - Lizenz

Eine andere Betrachtung setzt Nutzungsbedingung und Schlüssel in Beziehung. Ein Produkt wird dabei ebenfalls einer Nutzungsbedingung zugeordnet, jedoch wird jedes Produkt mit dem Schlüssel der zugehörigen Nutzungslizenz verschlüsselt.

Diese unterschiedlichen Betrachtungen sind bei einer möglichen Implementierung zu berücksichtigen und beeinflussen die Softwarearchitektur.

Im weiteren Verlauf werde ich erläutern, wie ich über eine Zwischenlösung von der DRM-Referenz Architektur, zum DRM-Framework gelange.

4.2 Von der DRM-Referenz Architektur zur Zwischenlösung

Die DRM-Referenz Architektur beschreibt im Wesentlichen drei Komponenten, welche das System bilden. Der Client (z.B. ein mobiles Endgerät) ist für die Darstellung der digitalen Inhalte verantwortlich und muss in der Lage sein, diese Inhalte im Rahmen der Nutzungsrechte dem Verbraucher zugänglich zu machen. Dazu befindet sich auf jedem Client eine Softwarekomponente (in der DRM-Referenz Architektur *DRM Controller* genannt). Der Client muss sicherstellen, dass alle in den Nutzungsrechten enthaltenen Schlüssel, nach außen hin unzugänglich sind. Ebenso darf der digitale Inhalt nicht über andere Komponente zugänglich sein, sodass das DRM-System unterlaufen werden kann.

Die Softwarekomponenten, die der Client nutzt um den digitalen Inhalt im Rahmen der Nutzungsrechte bereitzustellen, wird von den Endgeräte-Herstellern implementiert und richtet sich an ein bestimmtes DRM-System. Den meisten DRM-Systemen ist gemeinsam, dass die Inhalte getrennt von den Nutzungsrechten ausgeliefert werden (siehe hierzu Kapitel 2.3).

In der DRM-Referenz Architektur wird der *Content Server* beschrieben, der die Komponente *DRM Packager* beinhaltet. Diese Komponente muss nicht zwangsläufig im Content Server integriert sein, sondern ließe sich auch als separate Softwarekomponente auf einem dedizierten Server realisieren. Für die Realisierung des DRM-Frameworks wird dieser Aspekt aufgenommen. Der Content Server ist ein eigenständiger Service welcher schon vorhanden und and das DRM-Framework anzubinden ist.

Die letzte Komponente aus der DRM-Referenz Architektur, ist der *License Server*. Diese Komponente liefert die Nutzungsrechte zusammen mit den symmetrischen Schlüsseln aus. Für das DRM-Framework wird daher eine

Komponente benötigt, welche diese Aufgaben kapselt und entsprechend des zu verwendendem DRM-Systems eine Nutzungslizenz ausliefert.

Mein erster Ansatz besteht darin, die Teilkomponenten der DRM-Referenz Architektur auf einfache Softwarekomponenten abzubilden. Über die Komponente *LicenseManager* können unterschiedliche Nutzungsrechte an die digitalen Inhalte gebunden werden. Jedes Nutzungsrecht bekommt dabei einen eigenen symmetrischen Schlüssel, der in einer Datenbank gespeichert wird. Die Nutzungsrechte werden ebenfalls in einer Datenbank gespeichert und können so jederzeit durch die *Auslieferungsplattform* ausgeliefert werden.

Für die Erstellung der geschützten Inhalte ist die Komponente *DRMPackager* verantwortlich. Die Rohdaten der Inhalte werden dabei an diese Komponente übergeben, und werden dann in ein bestimmtes DRM-Format konvertiert. Die Konvertierung beinhaltet die Verschlüsselung der Daten. Der dazu notwendige Schlüssel bezieht der *DRMPackager* direkt von der Komponente *LicenseManager*. Damit ist der DRMPackager in der Lage DRM geschützte Inhalte an die Auslieferungsplattform zu übergeben. Die separate Auslieferung von Nutzungsrechten und Inhalten, sowie die kombinierte Auslieferung der geschützten Inhalte und Nutzungsrechte wird somit realisiert (siehe Kapitel 2.3.5 OMA DRM-System).

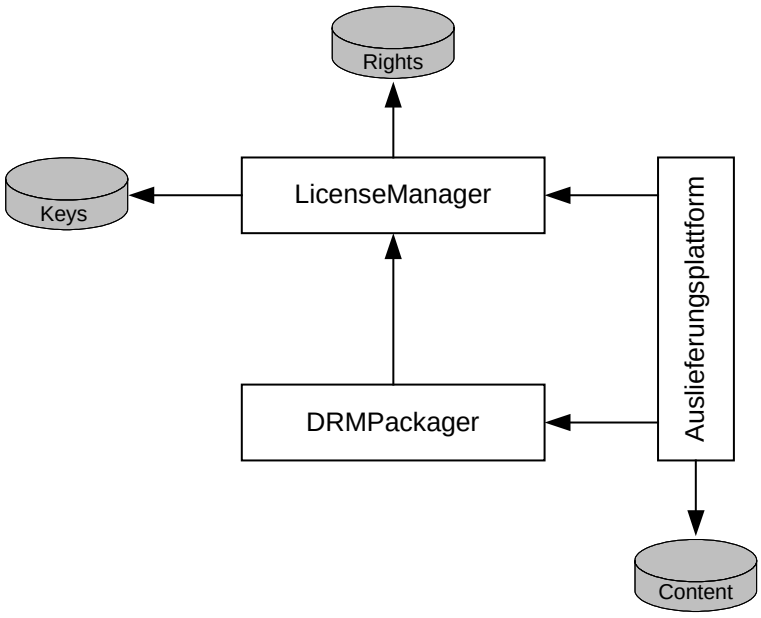


Abbildung 4.1: Erste Zwischenlösung

Dieses Verfahren beruht jedoch auf einer direkten Datenbank-Kommunikation. Für eine flexiblere Architektur wäre eine Zugriffsschicht vor der Datenbank denkbar, die alle möglichen Anfragen in Datenbankabfragen transformiert und die Ergebnisse als Objekte zurückliefert.

Nachteilig ist auch die unflexible Wahl eines bestimmten DRM-Systems, (siehe Kapitel 3, Anforderungen). Die *Auslieferungsplattform* muss als

eigenständige Softwarekomponente entscheiden können, welches DRM-System zum Einsatz kommen soll.

Die Architektur, so wie sie in Abbildung 4.1 dargestellt ist, hat so keinen Framework-Charakter. Damit ein Framework entsteht, so dass die Benutzer dieses als Rahmen für Erweiterungen nutzen können, müssen Schnittstellen definiert werden, die erweiterbar sind. Es muss möglich sein neue Klassen, bzw. ganze Komponenten, in das Framework zu integrieren.

4.3 Übergang zu einem Framework

Das Framework sollte von den zugrunde liegenden Datenbanken entkoppelt sein. Dazu ist die Erstellung von Datenbankschichten notwendig, die als Adapter fungieren. Alle Datenbank relevanten Anfragen werden durch eine Zwischenschicht (Abstraktionsschicht) durchgeführt. Als Erweiterung zur Architektur aus dem vorherigen Kapitel führe ich die Abstraktionsschichten *RightsStorage* und *KeyStorage* ein. Beide Komponenten erhalten eine Schnittstelle, welche die wesentlichen Methoden für den Zugriff auf die Datenbank bereitstellt. Die Komponenten *DRMPackager* und *DRMRightsManager* erhalten ebenfalls eine Schnittstelle. Durch die Schnittstellen ist der Austausch der Implementierungen leicht möglich.

Die Flexibilität des Frameworks kann durch den Einsatz von XML erhöht werden. Eine Konfigurationsdatei in XML definiert das Zusammenspiel der gewünschten Klassen untereinander. Der Nutzer des Frameworks kann in der XML Datei festlegen, welche Klassen die entsprechenden Funktionalitäten bereitstellen sollen. Ein entsprechender Aufzählungstyp in der DTD der XML Datei ermöglicht die Registrierung mehrerer Klassen, die die unterschiedlichen DRM-Systeme implementieren.

Durch den Einsatz einer *abstrakten Fabrik*[5] ist die Nennung eines gewünschten DRM-Systems möglich, welches unter einheitlichen Schnittstellen genutzt werden kann.

[5] Entwurfsmuster, beschrieben in [Gamma-2001].

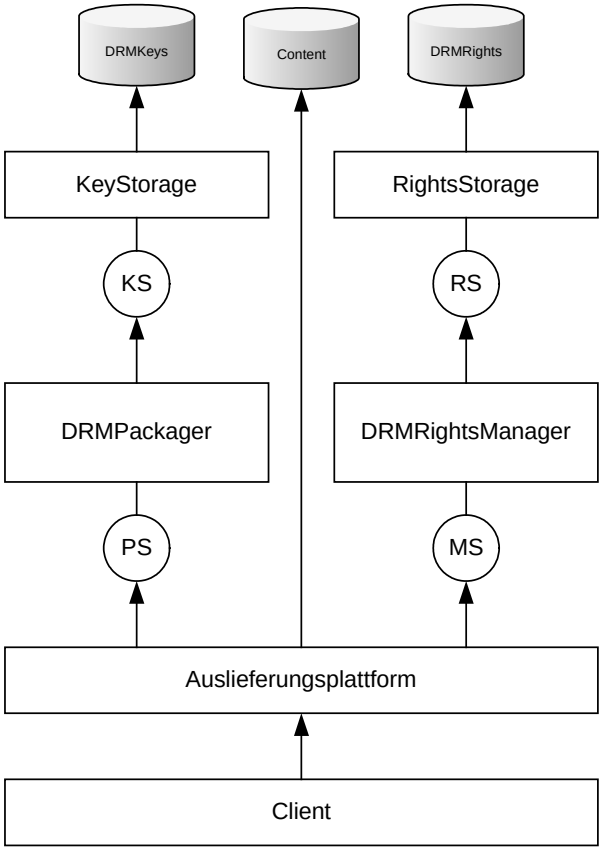


Abbildung 4.2: Architektur (Übersicht), DRM-Framework

4.3.1 DRMSystem/DRMConfig

Die Wahl eines gewünschten DRM-Systems soll zur Laufzeit möglich sein. Dies ist eine konkrete Anforderung aus Kapitel 3. Die Auslieferungsplattform benutzt das API des DRM-Frameworks und muss damit in der Lage sein ein DRM-System auszuwählen. Die Wahl eines DRM-Systems zur Laufzeit erfordert eine Konfigurationsmöglichkeit außerhalb des kompilierten Systems. Viele Systeme lassen sich heutzutage über XML konfigurieren. Beispiele hierfür sind der Applikationsserver JBoss[6], oder auch die Servlet-Engine Tomcat[7]. Beide Systeme beinhalten eine Sammlung von Konfigurationsdateien in Form von XML Dokumenten.

Das DRM-Framework folgt diesem Ansatz und stellt somit eine Konfigurationsdatei in XML zur Verfügung. Das Einlesen der einzelnen Parameter der Konfigurationsdatei übernimmt dabei die Klasse *DRMConfig*, welche über entsprechende Methoden dem System alle Konfigurationsparameter zur Verfügung stellt.

[6] http://www.jboss.org
[7] http://jakarta.apache.org/**tomcat**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE drm PUBLIC "-//Arvato-Mobile, GmbH//DTD Web Application 2.3//EN"
"http://www.arvato-mobile.de/DTD/drm.dtd">
<drm>
  <drmsystems>
    <implementation>
      <id>OMA</id>
      <uri>
        <uriprefix>cid:</uriprefix>
        <uripostfix>@http://drm.handy.de/omadelivery/icon.jsp</uripostfix>
      </uri>
      <contentwrapper>
        <interface>de.handy.drm.oma.OMAWrapped</interface>
        <class>de.handy.drm.oma.OMAPackager</class>
      </contentwrapper>
      <rights>
        <class>de.handy.drm.oma.OMARights</class>
      </rights>
    </implementation>
  </drmsystems>
  <keystorage>
    <class>de.handy.drm.drmstorage.KeyStorage</class>
  </keystorage>
  <rightsstorage>
    <class>de.handy.drm.drmstorage.RightsStorage</class>
  </rightsstorage>
  <resource-ref>
    <res-ref-name>jdbc/test</res-ref-name>
  </resource-ref>
</drm>
```

Abbildung 4.3: drm.xml – Konfigurationsdatei

Abbildung 4.3 zeigt die Beispielkonfigurationsdatei. Unter dem Abschnitt *drmsystems* ist zu sehen, welche DRM-Implementierungen am Framework angemeldet sind. Jede einzelne DRM-Implementierung ist unter dem Punkt *implementation* registriert. Dieser Abschnitt kann sich beliebig oft wiederholen und ermöglicht somit die Registrierung mehrerer DRM-Systeme.

Die Klasse *DRMSystem* dient dem Klienten (Der Auslieferungsplattform) als Fabrik[8]. Der Klient ist mit dieser Klasse in der Lage ein Objekt zu generieren, welches die Funktionsweise eines bestimmten DRM-Systems kapselt. Der Referenztyp des erzeugten Objektes lässt sich in der Konfigurationsdatei unter dem Abschnitt *contentwrapper* deklarieren. An dieser Stelle wird ein Interface und eine Klasse registriert, welche die Methoden des gewünschten DRM-Systems implementieren.

Die Klasse *DRMSystem* stellt folgende Methoden bereit:

- `DRMWrapped getDRMWrapper(String drmSystem) throws DRMException`
- `DRMRightsManager getRightsManager() throws DRMException`

[8] Entwurfsmuster, beschrieben in [Gamma-2001].

Die Methode `getDRMWrapper(String drmSystem)` erstellt ein Objekt, das ein bestimmtes DRM-System implementiert. Als Argument übergibt man der Methode die `Id`, unter welcher das gewünschte DRM-System in der Konfigurationsdatei registriert ist. Im Beispiel der Abbildung 4.3 wäre das der String "`OMA`". Der Rückgabetyp ist ein Interface (registriert unter *drm/drmsystems/implementation/contentwrapper/interface*). Dieses Interface ist Subtyp des Interface *DRMWrapped* und bietet DRM-spezifische Methoden an.

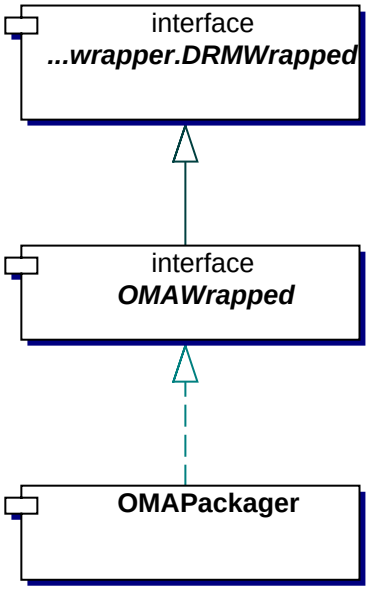


Abbildung 4.4: Interface-Hierarchie für DRMWrapped

Mit der Methode `getRightsManager()` wird ein Objekt zurückgeliefert, welches für die Behandlung von Nutzungsrechten zuständig ist. Auf die Klasse *RightsManager*, bzw. das Interface *DRMRightsManager* wird weiter unten eingegangen.

Die Klasse *DRMSystem* delegiert die Suche nach Konfigurationsparametern an die Klasse *DRMConfig*. Mit diesem Mechanismus ist der Klient in der Lage ein DRM-System zur Laufzeit zu bestimmen und jederzeit weitere DRM-Systeme zu integrieren.

4.3.2 DRMPackager/DRMWrapped

Die Klasse DRMSystem liefert, wie im vorherigen Unterabschnitt beschrieben, ein Objekt zurück, welches die Funktion des gewünschten DRM-Systems kapselt. Dazu werden in der Konfigurationsdatei die Klasse und das zugehörige Interface registriert.

Der DRM abhängige Packager implementiert ein Interface, welches Subtyp des Interface *DRMWrapped* ist. Damit sind alle Schnittstellen, über die die DRM-Systeme angesprochen werden, Subtypen von *DRMWrapped*. Für die Methode `getDRMWrapper(String drmSystem)` aus der Klasse *DRMSystem*

kann daher ein Basis-Interface (*DRMWrapped*) als Rückgabetyp deklariert werden.

Das Basis-Interface *DRMWrapped* stellt folgende Methoden zur Verfügung:

- `void setPlainData(byte[] data)`
- `void setPlainData(InputStream plainData)`
- `void setContentUri(String cUri)`
- `byte[] createEncoded() throws DRMException`

Die ersten beiden Methoden dienen zur Übergabe der Rohdaten an den Packager. Die kann entweder im Ganzen als Byte Array passieren, oder aber als *InputStream*. Dem Packager wird dann noch mit der Methode `void setContentUri(String cUri)` ein eindeutiger Identifier (URI) mitgeteilt, der den digitalen Inhalt identifiziert. Durch Aufruf der Methode `byte[] createEncoded()` wird der Inhalt in das DRM-spezifische Content-Format konvertiert.

Für die Implementierung des OMA DRM-Systems wird später ein Interface erstellt, welches das Interface *DRMWrapped* erweitert. Es werden neue Methoden hinzugefügt, die das OMA DRM-System charakterisieren. Auf diese Details werde ich in Kapitel 5 (Implementierung) näher eingehen.

4.3.3 DRMRightsConstraints

Das Interface *DRMRightsConstraints* wird von der Klasse *RightsConstraints* implementiert. Die Klasse dient zur Formulierung der Nutzungsbedingungen an den *DRMRightsManager*. Es wird für jede Nutzungsbedingung ein neues Objekt *DRMRightsManager* erzeugt. Eine Nutzungsbedingung könnte z.B. sein: Zehn mal abspielen der Datei in einem Zeitraum vom 01.01.2005 – 31.01.2005. Diese Bedingung wird folgendermaßen formuliert:

```
DRMRightsManager rm = drmSystem.getRightsManager("OMA");
DRMRightsConstraints rc = new RightsConstraints();

rc.setDateStart(new GregorianCalendar(2005, 1, 1));
rc.setDateEnd(new GregorianCalendar(2005, 1, 31));
rc.setCount(10);
rc.setRightsUsage(DRMRightsConstraints.PLAY);
rm.addRightsConstraints(rc);
```

Abbildung 4.5: Nutzungsbedingung formulieren

Zuletzt wird diese Nutzungsbedingung dem DRMRightsManager übergeben, der daraus ein Rechteobjekt generiert.

4.3.4 DRMRightsManager/RightsManager

Die Klasse DRMSystem liefert mit der Methode `getRightsManager()` ein Objekt, welches für die Erstellung und Behandlung der Nutzungsrechte verantwortlich ist. Die Klasse *RightsManager* implementiert dazu das Interface *DRMRightsManager* und stellt damit folgende Methoden bereit:

- `void addRightsConstraints(DRMRightsConstraints para)`
- `DRMRights getRights(String uri) throws DRMException`

- `DRMRights getConfiguredRightsFromDb(String cUri) throws DRMException`

Die Methode `getConfiguredRightsFromDb(String cUri)` ist eher für administrative Zwecke gedacht. Damit lässt sich feststellen, welche Nutzungsbedingungen für ein bestimmtes Produkt in der Datenbank hinterlegt sind.

Die Erstellung, bzw. das Abfragen eines bestimmten Rechteobjekts wird über den *RightsManager* wie folgt abgewickelt:

- Der Klient erstellt ein Objekt vom Typ *DRMRightsConstraints* und formuliert damit die Nutzungsbedingung (wie im vorherigen Unterabschnitt gezeigt).
- Diese Nutzungsbedingung wird anschließend dem *DRMRightsManager* übergeben.
- Mit dem Aufruf der Methode `getRights(String uri)` erhält der Klient ein Rechteobjekt vom Typ *DRMRights*.

Der *RightsManager* erstellt das Recht sobald, wenn er die formulierten Nutzungsbedingungen mit der Datenbank abgeglichen hat. Diese Aufgabe delegiert er an die Klasse *RightsStorage*. Diese ist ebenfalls in der Konfigurationsdatei deklariert und lässt sich bei Bedarf austauschen. Der *RightsStorage* kapselt alle relevanten Datenbankanfragen und stellt dem Klienten dafür, über das Interface *DRMRightsStorage,* eine einheitliche Schnittstelle zur Verfügung.

4.3.5 DRMRights

Ein Rechteobjekt wird innerhalb des Frameworks durch eine Referenz vom Typ *DRMRights* repräsentiert. Ein Objekt dieses Typs wird vom *DRMRightsManager* erstellt und zurückgeliefert. Das Interface *DRMRights* ist ein Basisinterface, das als Supertyp für DRM abhängige Rechteobjekte dient. Die OMA Implementierung hat ein spezielles Rechteobjekt und benutzt als Referenztyp ein Subtyp von *DRMRights*. Abbildung 4.6 verdeutlicht diesen Zusammenhang.

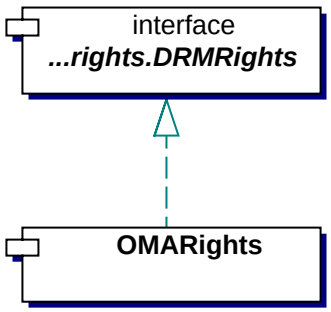


Abbildung 4.6: DRMRights als Supertyp für OMARights

Das Basisinterface *DRMRights* stellt folgende Methoden bereit:

- `String toXml() throws DRMException`
- `byte[] toWbXml() throws DRMException`

- `void setRightsConstraints(List constraints) throws DRMException`
- `void setEncryptionKey(byte[] key) throws DRMException`
- `void setContentUri(String uri)`
- `String getContentUri()`
- `String getFullContentUri()`
- `String getContentUriPrefix()`
- `String getContentUriPostfix()`

Für die externe Repräsentation dienen die Methoden `toXml()` und `toWbXml()`. Damit wird das Rechteobjekt in ODRL konvertiert, bzw. in WBXML. Mit der Methode `setRightsConstraints(List contraints)` werden dem Rechteobjekt Nutzungsbedingungen übergeben, die in die Erstellung des Rechteobjekts mit einfließen.

Für die Nutzung der digitalen Inhalte müssen diese durch das Rechteobjekt zugreifbar gemacht werden. Dazu beinhaltet das Rechteobjekt den symmetrischen Schlüssel. Dieser wird mit der Methode `setEncryptionKey(byte[] key)` gesetzt. Weiterhin wird ein eindeutiger Bezeichner hinterlegt, der den digitalen Inhalt identifiziert. Dies ermöglicht die Methode `setContentUri(String uri)`.

Das Interface *DRMRights* definiert weiterhin noch einige Getter-Methoden. Mit diesen Methoden ist es z.B. möglich einen Pre- und einen Postfix für den URI zu erhalten. Manche DRM-Systeme verlangen neben den eigentlichen Identifier einen immer gleichen Wert vor, bzw. hinter diesem. Diese können in der Konfigurationsdatei deklariert werden.

Zugriffsschichten – DRMRightsStorage/DRMKeyStorage

Die Klassen für die Zugriffsschichten sind in der Konfigurationsdatei unter dem Abschnitt *keystorage* und *rightsstorage* deklariert. Diese Schichten stellen Methoden über Interfaces bereit, um datenbankunabhängig auf die persistenten Daten zuzugreifen. Die Klassen für die Zugriffsschichten kapseln den Zugriff über JDBC [java.sun.com] auf die darunter liegende Datenbank.

4.3.6 RightsStorage

Der RightsStorage wird über das Interface DRMRightsStorage referenziert. Dieses Interface definiert zwei Methoden:

- `boolean lookupRight(String cUri, List rightsRequest)`
- `DRMRights getRightsFromDb(String cUri)`

Die Methode `lookupRight` dient dazu eine Liste von Nutzungsbedingungen gegen die hinterlegten Nutzungsbedingungen in der Datenbank zu prüfen. Dazu wird der Methode der URI des gewünschten Inhalts übergeben und die zu prüfenden Nutzungsbedingungen als *List* Objekt. Die Schicht prüft anschließend, ob alle Nutzungsbedingungen für den gewünschten digitalen

Inhalt entsprechend in der Datenbank hinterlegt sind. Sollten alle Nutzungsbedingungen korrekt in der Datenbank hinterlegt worden sein wird `true`, ansonsten `false` zurückgeliefert.

Die zweite Methode erstellt ein Rechteobjekt anhand des übergebenen URI.

4.3.7 KeyStorage

Auch der *KeyStorage* wird über ein Interface (*DRMKeyStorage*) referenziert. Das zugehörige Interface definiert folgende Methoden:

- **byte[]** lookupKey(Key productId)
- **void** storeKey(Key productId, **byte[]** key)

`lookupKey` sucht nach einem Schlüssel in der Datenbank. Als Parameter wird dabei ein Objekt der Klasse *Key* übergeben. Wobei mit Key nicht der symmetrische Schlüssel gemeint ist, sondern vielmehr ein Wrapper für den URI. Der URI (in diesem Fall *productId*) wird im *KeyStorage* zwischengespeichert. Dieses Zwischenspeichern dient zum schnelleren Auffinden eines Schlüssels, um die darunter liegende Datenbank zu entlasten. Der symmetrische Schlüssel wird anschließend als Byte Array zurückgegeben.

Die Methode `storeKey` nimmt einen Schlüssel und einen URI (Parameter *Key*) entgegen. Der Schlüssel wird im Cache des *KeyStorage* zwischengespeichert und in der Datenbank hinterlegt.

4.3.8 Statische Sicht

DRMPackager

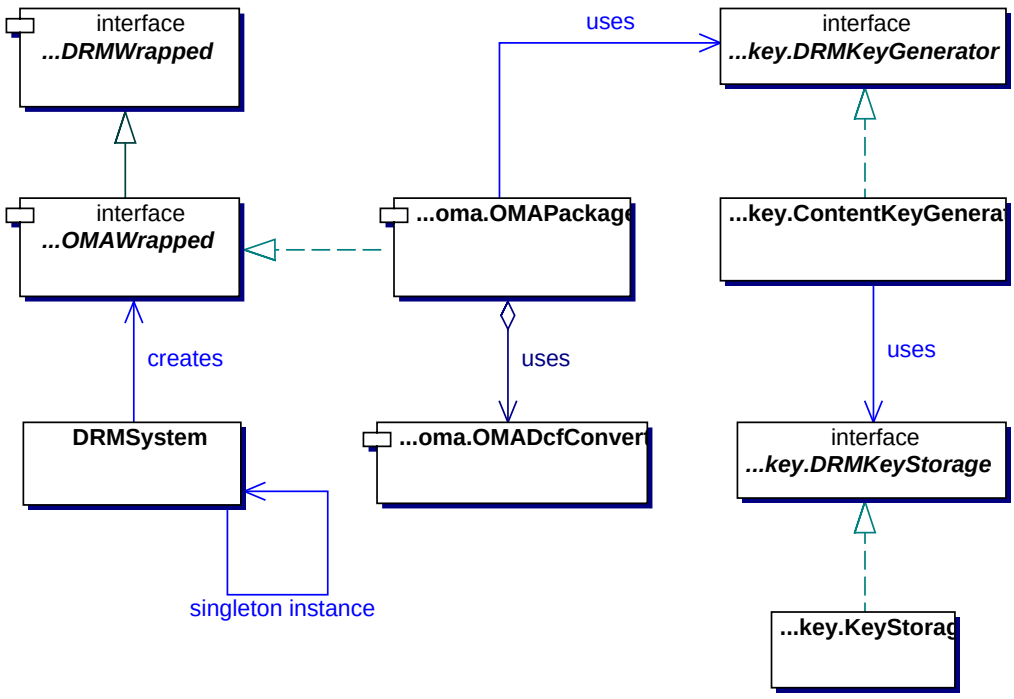


Abbildung 4.7: DRMPackager, statische Sicht

Für die Implementierung des OMA DRM-Systems sind aus der Abbildung 4.7 zwei Elemente in der Konfigurationsdatei deklariert. Zum einen das Interface *OMAWrapped*, zum anderen die Klasse *OMAPackager*, welche dieses Interface implementiert.

Die Klasse *DRMSystem* ist verantwortlich für die Instantiieren eines bestimmten DRM-Systems. Sie gibt dem Klienten ein Objekt als Referenztyp *OMAWrapped* zurück.

Der *OMAPackager* delegiert seine Aufgaben an entsprechende Hilfsklassen. Symmetrische Schlüssel werden vom *ContentKeyGenerator* erzeugt und über den *KeyStorage* in einer Datenbank gespeichert. Der *OMADcfConverter* übernimmt den eigentlichen Verschlüsselungsprozess und erzeugt den OMA DRM geschützten Inhalt im DCF Format.

RightsManager

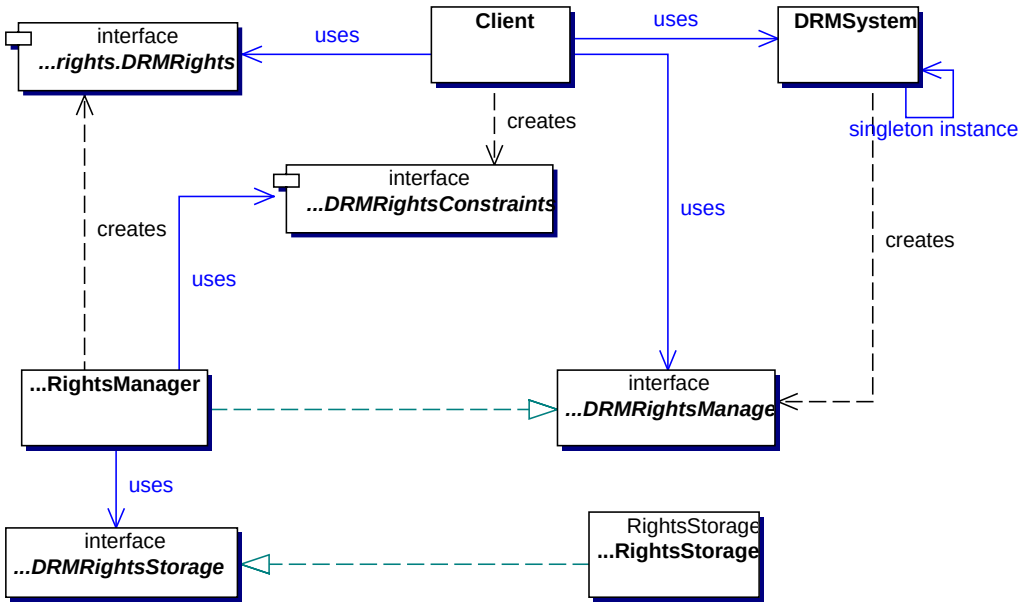


Abbildung 4.8: DRMRightsManager

Der *RightsManager* erstellt mit Hilfe der ihm übergebenen Nutzungsbedingungen (*DRMRightsConstraints*) ein Rechteobjekt (*DRMRights*). Der *RightsStorage* wird zur Überprüfung der Nutzungsbedingungen vom *RightsManager* verwendet. Bei erfolgreicher Überprüfung wird ein Rechteobjekt (*DRMRights*) erzeugt und dem Klienten zurückgegeben.

Im nächsten Kapitel werde ich Details der Implementierung beschreiben. Ich erläutere dabei, welche externen Pakete ich verwendet habe und wie das OMA DRM-System implementiert ist.

5 Implementierung des DRM-Frameworks

Als Referenzimplementierung für das DRM-Framework ist, nach den Anforderungen, das OMA DRM-System zu wählen. Aus diesem Grund werde ich in diesem Kapitel die Implementierung beispielhaft an diesem DRM-System durchführen.

5.1 Die Sprache der Implementierung

Die Implementierung des DRM-Frameworks wird mit der Sprache Java realisiert. Diese Sprache wird im Umfeld der serverseitigen Applikationen, der Firma arvato mobile, vorzugsweise eingesetzt. Die Anwendungen werden dabei mit Hilfe J2EE[9] Technologie realisiert. Vor diesem Hintergrund ist die Erstellung einer Java-Komponente in Form eines JAR Pakets zu realisieren, welche das DRM-Framework darstellt.

Die Sprache Java ursprünglich Oak genannt wurde in einem Zeitraum von 18 Monaten vom amerikanischen Softwarehersteller Sun Microsystems entwickelt. Das eigentliche Ziel dabei war nicht die Schaffung einer neuen Programmiersprache, sondern vielmehr die Entwicklung einer vollständigen Bestriebssystemumgebung, inklusive virtuelle CPU. Gerade das Internet dient als Hauptbetätigungsfeld der Sprache Java. Java Applets sind in der Lage als kleine Programme im Browser ausgeführt zu werden. Heutzutage wird Java hauptsächlich zur Entwicklung von Webanwendungen eingesetzt.

Die Sprache Java lehnt sehr stark an die Syntax von C und C++ an. Obwohl die Sprache Java sich von der Syntax sehr stark mit der von C und C++ gleicht, unterscheiden sich die Sprachen grundlegend. Java integriert Funktionalitäten wie Multithreading, strukturiertes Exceptionhandling oder graphische Fähigkeiten. Ein weiterer wesentlicher Unterschied zu C und C++ ist das Fehlen von Pointern, separaten Headerdateien und Mehrfachvererbung, auf die bewusst verzichtet wurde.

Ein wichtiger Bestandteil der Sprache Java ist der Garbage-Collector. Hierbei handelt es sich um einen niedrig priorisierten Thread, welcher eine automatische Speicherbereingung durchführt. Im Gegensatz zu C und C++ werden Objekte in Java nur durch den Aufruf des `new` Operators im Speicher angelegt. Sobald ein Objekt nicht mehr referenziert wird kann es durch den Garbage-Collector wieder aus dem Speicher entfernt werden. Dieser Mechanismus schließt eine sehr große Fehlerquelle (das Vergessen der manuellen Speicherbereinigung) aus, die bei der klassischen Programmierung in C und C++ immer wieder auftritt.

5.1.1 DRMSystem

Diese Klasse ist eine *Fabrik*[10], die ein Objekt zurückgibt, welches das gewünschte DRM-System repräsentiert. Die Klasse DRMSystem bezieht

[9] J2EE: Java 2 Platform, Enterprise Edition. http://java.sun.com/j2ee/index.jsp
[10] Entwurfsmuster, beschrieben in [Gamma-2001].

dabei die Informationen über die möglichen DRM-Systeme aus einer Konfigurationsdatei. Diese Konfigurationsdatei ist in XML formuliert.

In der Konfigurationsdatei ist es möglich mehrere DRM-Systeme zu registrieren. Jedes System wird dazu in einen Abschnitt *drmsystems/implementation* gebettet und deklariert weitere Eigenschaften. Der Abschnitt *implementation* kann sich dabei beliebig oft wiederholen.

Die Konfigurationsdatei wird durch die Konfigurationsklasse *DRMConfig* eingelesen. Sie stellt Methoden bereit, um alle registrierten DRM-Systeme zu erhalten. Zurückgeliefert wird dann beispielsweise eine Liste sämtlicher Klassen (mit komplettem Package-Pfad). Anhand dieser Liste ist es der Klasse *DRMSystem* möglich durch den Reflection-Mechanismus[11] die entsprechende Klasse zu instantiieren.

Die Klasse *DRMConfig* ist für das Einlesen der Daten aus der Konfigurationsdatei verantwortlich. Das Format der Konfigurationsdatei ist, wie in Kapitel 4 bereits beschrieben, in XML formuliert. Das Einlesen dieser Daten erfolgt mit Hilfe einer externen Bibliothek (*org.apache.commons.digester.Digester*). Dieses Package stellt Klassen zur Verfügung, die es auf sehr einfache Art erlaubt Werte aus einer XML Datei als Objekte einzulesen.

Die Klasse *DRMConfig* kann die eingelesenen Werte dann über Getter-Methoden anderen Klassen zugänglich machen. Die Erstellung eines *OMAWrapped* Objektes erfolgt in der Klasse *DRMSystem* folgendermaßen:

- Der Klient erstellt eine Instanz der Klasse DRMSystem.
- Mit diesem Objekt initiiert er die Erstellung eines Objektes vom Typ *OMAWrapped*.

```
DRMSystem system = DRMSystem.getDRMSystem();
OMAWrapped oma = (OMAWrapped)system.getDRMWrapped("OMA");
...
```

Abbildung 5.1: Instanz von OMAWrapped erstellen

Die zweite Zeile in Abbildung 5.1 erzeugt auf Grund des aktuellen Arguments „OMA“ ein Objekt vom Typ *OMAWrapped*. Dieses Objekt muss vor der Zuweisung noch zum richtigen Typ (*OMAWrapped*) gecastet werden, da die Methode in der Signatur ein Objekt vom Typ *DRMWrapped* zurückliefert. *OMAWrapped* ist, wie in Abbildung 4.4 zu sehen ein Subtyp von *DRMWrapped*.

Die Erstellung dieses Objektes erfolgt in der Methode `getDRMWrapped()` mit Hilfe des Reflection-Mechanismus.

[11] Durch den Reflection-Mechanismus ist es möglich zur Compilezeit noch unbekannte Klassen zu laden. Diese können dann wie bekannte Klassen benutzt werden.

```
private static List impl;
...
int len = impl.size();
for(int i=0; i < len; i++) {
   DRMConfigImplementations cim = (DRMConfigImplementations )impl.get(i);
   // if DRM-System is registered in drm.xml, try to create
   // an instance.
   if(cim.getId().equalsIgnoreCase(drmSystem)) {
      try {
         // obtain class object from wrapper class.
         requestedDRMClass = Class.forName(cim.getWrapperClass());
         // create an instance from class object
         requestedDRM = (DRMWrapped) requestedDRMClass.newInstance();
      } catch (IllegalAccessException e) {
         throw new DRMException(DRMErr.CANNOT_CREATE_PACKAGER);
      } catch (InstantiationException e) {
         throw new DRMException(DRMErr.CANNOT_CREATE_PACKAGER);
      } catch (ClassNotFoundException e) {
         throw new DRMException(DRMErr.CANNOT_CREATE_PACKAGER);
      }
   }
}
```

Abbildung 5.2: Code-Ausschnitt aus DRMSystem

Abbildung 5.2 zeigt einen Ausschnitt aus der Klasse *DRMSystem*. Die Variable *impl* ist eine Liste, in der alle registrierten DRM-System hinterlegt sind. Diese Variable wird im Konstruktor der Klasse *DRMSystem* gefüllt. Dazu verwendet *DRMSystem* die Klasse *DRMConfig*.

Es wird die Liste mit allen registrierten DRM-Systemen durchiteriert, und die *Id* mit dem übergebenen Argument „OMA“ verglichen. Sobald die Id's übereinstimmen kann eine Instanz erzeugt werden. In der Klasse *DRMConfigImplementations* ist die komplette Konfiguration eines DRM-Systems hinterlegt. Über die Methode `getWrapperClass()` kann die Klasse festgestellt werden, die den eigentlichen Wrapper bilden soll.

Das zugehörige Klassen-Objekt wird durch die Methode `Class.forName(cim.getWrapperClass())` erzeugt. Die eigentliche Instantiierung des Objekts durch den Reflection-Mechanismus erfolgt mit der Methode `newInstance()` des Klassen-Objekts.

5.1.2 DRMPackager

Zur Generierung der DRM geschützten digitalen Inhalten gibt es die Komponente *DRMPackager*. Diese Komponente besteht aus mehreren Software-Artefakten, abhängig von der jeweiligen DRM-Implementation.

Die Referenzimplementierung des DRM-Frameworks besteht aus dem DRM-System OMA. Digitale Inhalte werden nach der OMA Spezifikation in das DCF Format konvertiert. Dieses Format dient zur separaten Auslieferung von digitalen Inhalten und Rechteobjekten. Implementiert ist im Framework das OMA Auslieferungsmodell *separate delivery* (siehe Kapitel 2.3.5).

Der OMAPackager besteht demnach aus folgenden Klassen und Interfaces:

- **class** OMAPackager
- **interface** OMAWrapped
- **interface** DRMKeyGenerator

- **class** ContentKeyGenerator
- **class** OMADcfConverter
- **interface** IDownloadDescriptor
- **class** DownloadDescriptor
- **interface** DRMObjectCache

Die Klasse *OMAPackager* repräsentiert dabei das eigentliche Wrapperobjekt welches die Inhalte erzeugt. Es implementiert das Interface *OMAWrapped*.

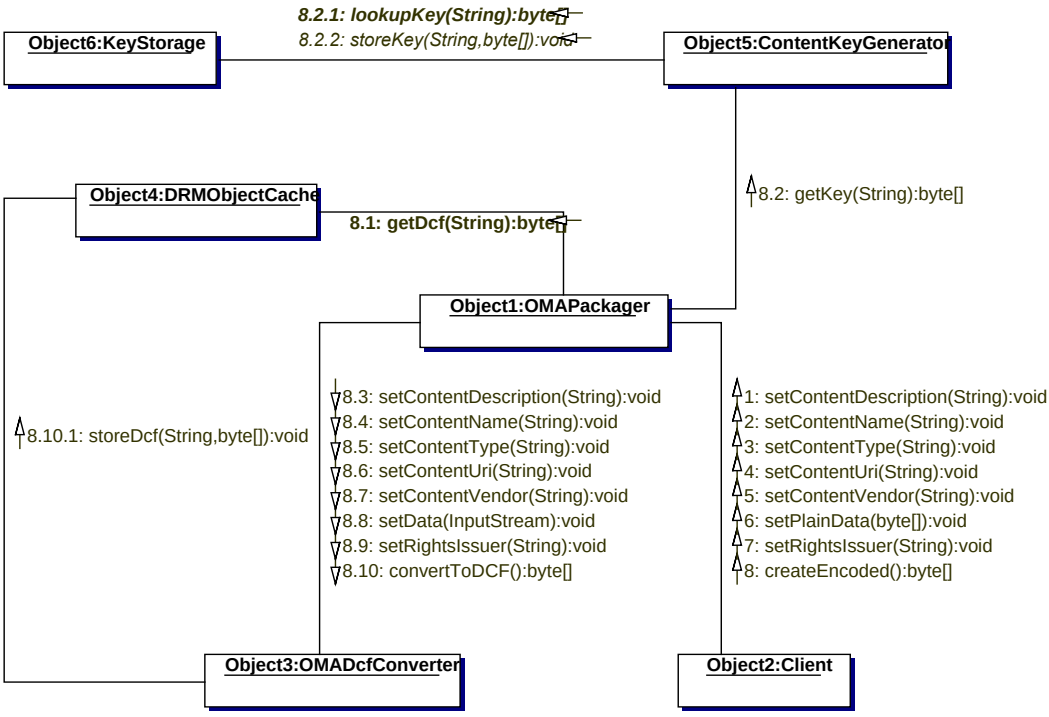


Abbildung 5.3: Ablauf des Packager-Prozess

Das Erstellen der DRM geschützten digitalen Inhalte entspricht dem Ablauf aus Abbildung 5.3. Über verschiedene Setter-Methoden werden dem Packager die Metadaten des Inhalts bekannt gemacht. Es erfolgt dann die Übergabe der Rohdaten (In der Abbildung als Byte Array). Als letzter Schritt wird die Methode `createEncoded()` aufgerufen. Diese Methode liefert den DRM geschützten Inhalt als Byte Array zurück.

Der *OMAPackager* versucht als erstes den DRM geschützten digitalen Inhalt aus dem *DRMObjectCache* zu holen. Wenn dieser noch nicht zu einem früheren Zeitpunkt erstellt worden ist, muss er jetzt erstellt werden. Für die Erstellung braucht der Packager als erstes einen symmetrischen Schlüssel. Diesen bekommt er durch die Klasse *ContentKeyGenerator*. Der *ContentKeyGenerator* versucht zuerst einen symmetrischen Schlüssel aus einer Datenbank zu beziehen. Wenn dieser Versuch erfolglos war, generiert diese Klasse einen neuen und legt diesen über den *KeyStorage* in der Datenbank ab. Nach der OMA Spezifikation werden für die symmetrischen Schlüssel, Schlüssel der Länge 128Bit, mit dem AES Algorithmus und dem CBC Modus verwendet.

Symmetrische Verschlüsselung lässt sich mit Hilfe der JCA (*Java Cryptography Architecture*), bzw. JCE (*Java Cryptography Extensions*) realisieren. Dabei handelt es sich um ein API welches im Java Sprachkern fest verankert ist und Factory-Methoden zur symmetrischen und asymmetrischen Verschlüsselung bereitstellt. Konkrete Implementierungen sind wenig vorhanden, werden allerdings von Drittanbietern angeboten.

Um den Algorithmus, so wie er in der Spezifikation verlangt ist verwenden zu können, greife ich auf ein externes Package (cryptix-jce-api[12]) zurück, welches eine Implementierung des AES Algorithmus bereitstellt.

Nach der Erstellung wird der geschützte Inhalt noch im Cache zwischengespeichert und steht bei der nächsten Anfrage schneller im Zugriff.

Als nächster Schritt müssen jetzt die Rohdaten mit Hilfe des symmetrischen Schlüssels verschlüsselt werden. Diese Aufgabe wird durch den *OMADcfConverter* erledigt. Diese Klasse bekommt alle Metadaten, die der *OMAPackager* zuvor bekommen hat, übermittelt. Die OMA Spezifikation sieht für den separaten Auslieferungsmechanismus (*separate delivery*) das Format DCF vor. Dieses Format beinhaltet unter Anderem Angaben zum Rechteinhaber (die URL von der die Rechteobjekte bezogen werden können), Metadaten, wie Name, Beschreibung, Identifier etc., und den verschlüsselten Inhalt selbst. Eine genaue Beschreibung des Formats findet sich im Anhang. Bevor der DRM geschützte Inhalt an den Klienten zurückgegeben wird, wird dieser in einem Cache des *OMAPackager* zwischengespeichert. Bei einer erneuten Anfrage kann jetzt der Inhalt direkt aus den Cache bezogen werden, und muss nicht mehr den kompletten Packager-Prozess durchlaufen.

Das Ergebnis wird zum Schluss als Byte Array an den Klienten zurückgegeben.

5.1.3 DRMRights

Der *RightsManager* muss ein generelles Objekt als Rechteobjekt zurückliefern. Durch das Interface *DRMRights* erschaffe ich ein Referenztyp, welcher als generelles Rechteobjekt angesehen werden kann.

Das Rechteobjekt, das durch den *RightsManager* zurückgeliefert wird, ist ein Rechteobjekt entsprechend dem gewünschten DRM-System. In der Konfigurationsdatei wird hinter dem XML/Tag *rights/class* eine Klasse deklariert, die die Implementierung des Rechteobjektes für das gewünschte DRM-System bereitstellt. Über das Interface *DRMRights* wird das Rechteobjekt referenziert.

[12] http://www.cryptix.org

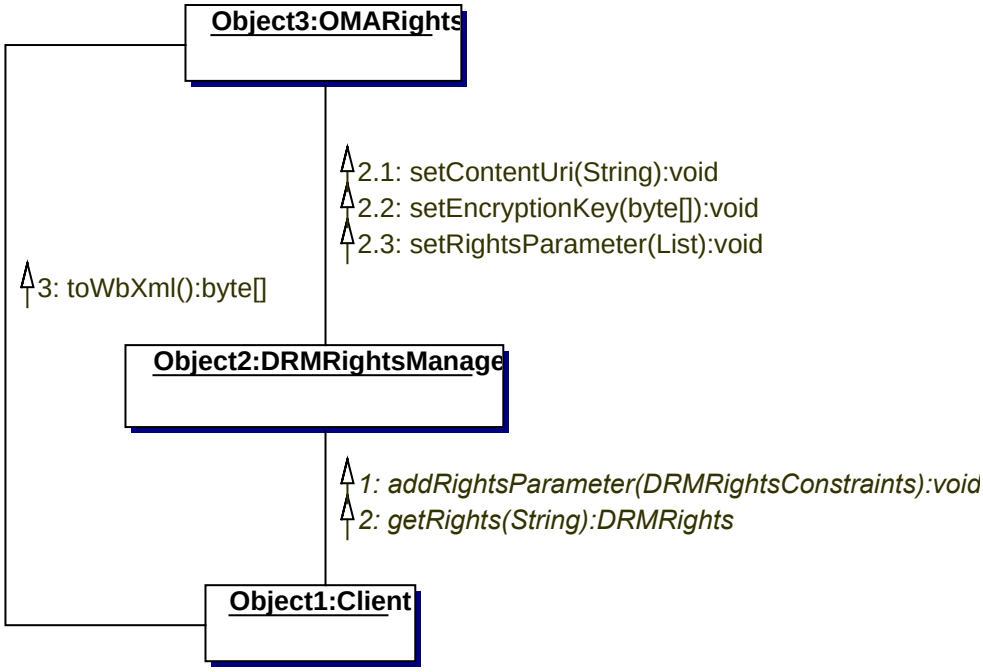


Abbildung 5.4: Erstellung eines Rechteobjekts

Das Rechteobjekt muss für die externe Darstellung in verschiedene Formen konvertiert werden können. Die am meisten verbreitete Form ist die Darstellung in XML. Speziell für DRM-Systeme gibt es den XML-Dialekt ODRL. Dieser Dialekt wird auch vom OMA-Standard benutzt. Die Methode `toXML()` konvertiert die Daten des Rechteobjektes in ODRL.

Bei mobilen Endgeräten wird die Nutzungslizenz auf Grund der effizienteren Übertragung in einer anderen Darstellung erwartet. Hierfür ist die Methode `toWbXML()` vorgesehen. Die Daten des Rechteobjektes werden dabei in eine binäre XML Darstellung (WBXML[13]) konvertiert.

Für die XML Darstellung beinhaltet die Klasse *OMARights* mehrere Konstanten für die einzelnen XML Tags. In der Methode `toXML()` werden alle Metadaten und Nutzungsbedingungen in einem String als XML Dokument zusammengesetzt.

Die Klasse *OMARights* besitzt eine innere Member Klasse *WbXMLHandler*, welche die Konvertierung zu WBXML übernimmt. Diese Klasse benutzt dazu die erstellte XML Darstellung mittels `toXML()`. Dieser String wird durch einen SAX-Parser in die einzelnen Bestandteile der WBXML Darstellung zerlegt. Als Ergebnis entsteht ein Byte Array, welches die WBXML Darstellung des Rechteobjektes ist.

5.2 Ausnahmen

Fast alle Methoden des Frameworks können Ausnahmen (Exceptions) auslösen. Ausnahmen dienen einer übersichtlichen Fehlerbehandlung innerhalb einer Applikation. In vielen Fällen werden Rückgabewerte dazu

[13] WBXML: siehe http://www.w3.org/TR/wbxml/

benutzt um Fehler, die in einer Methode auftreten können, anzuzeigen. Fehler könnten dabei durch einen Rückgabewert von -1 oder 0 angezeigt werden. Diese Darstellung ist jedoch willkürlich und führt zu unangenehmen Programmflüssen. Ein weiteres Problem besteht darin, dass diese Rückgabewerte häufig nicht abgefragt werden, da man der Meinung sein könnte, an dieser Stelle könne kein Fehler auftreten.

Java bietet für die Behandlung von Fehler, bzw. Ausnahmen einen eigenen Mechanismus an. Dieser Mechanismus heißt *Exception*. Nicht jeder Fehler muss zwangsläufig zu einem Programmabbruch führen. Man nennt dieses Ereignis dann eine Ausnahme und keinen Fehler. Der Exception Mechanismus in Java bietet die Möglichkeit aus einem bestehenden Pool von Exceptionklassen eine geeignete zu wählen. Eine *NullPointerException* zeigt z.B. an, dass ein Parameter kein gültiges Objekt referenziert. Die Unterscheidung der verschieden Arten von Ausnahmen wird durch eine sehr große Klassenhierarchie realisiert. Neue und bestehende Ausnahmen werden fast alle von der Basisklasse *Exception* abgeleitet. Dabei ist zu beachten, dass fast keine dieser Klassen wirklich neue Eigenschaften hinzufügt. In [Esser-2001] wird dieser Misstand beschrieben. Die Meldung der Ausnahme wird schon im Namen der Subklasse deutlich, was dazu führt das es pro Objekt eine Klasse gibt.

In [Esser-2001] wird eine alternatives Design-Muster vorgeschlagen, welches ich für das DRM-Framework aufgegriffen habe. Dieses Design-Muster basiert dabei auf das Facade-Pattern. Damit wird erreicht, dass nicht jede Ausnahme-Art mit einem neuen Typ beantwortet wird. Diese Art der Exception stellt eine typsichere Konstante zur Verfügung. Somit werden unterschiedliche Ausnahme-Arten durch unterschiedliche Instanzen und nicht durch Subtypen repräsentiert.

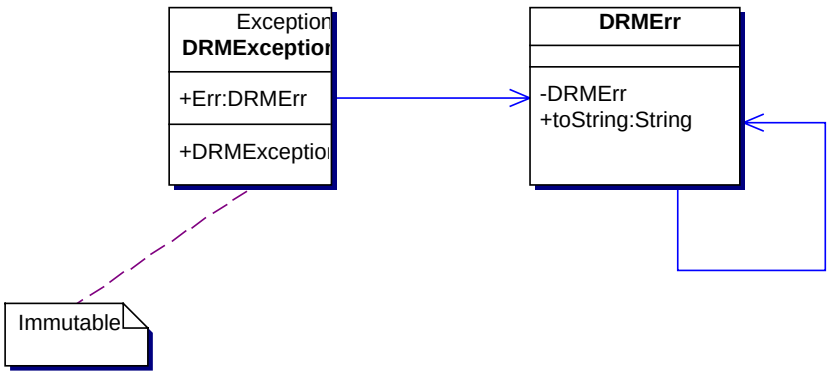


Abbildung 5.5: DRMException, typsichere Konstante

Damit ist der Benutzer in der Lage jeden Methodenaufruf mit nur einer Exception abzufangen. Damit er aber weiß welcher Fehler letztendlich aufgetreten ist, kann er sich auf die in der Instanz der Exception verborgenen Konstante beziehen, und diese zum Vergleich heranziehen.

```
…
catch(DRMException e) {
   if(e.Err == DRMErr.NOKEY) {
      …
   }
}
```

Abbildung 5.6: Vergleich einer DRMException

Da es sich bei der verborgenen Information (Die Fehlerart), um eine Konstante handelt, ist es möglich mit dem == Operator auf Gleichheit zu prüfen. Es muss nicht `.equals()` verwendet werden.

Mit diesem Idiom ist es möglich fehlerhafte Angaben zur Compilezeit zu erkennen, und trotzdem gibt es nur eine Ausnahmeklasse, was den Code wesentlich übersichtlicher gestaltet.

5.3 Anwendungsbeispiel

Zur Verdeutlichung der Abläufe im DRM-Framework demonstriere ich die Benutzung anhand eines konkreten Beispiels. Im Beispiel wird die separate Auslieferung (*separate delivery*) der OMA Implementierung gezeigt.

Der Benutzer eines mobilen Endgerätes (Handy, PDA, etc) gelangt über eine URL auf eine Seite (z.B. HTML oder JSP), von der er verschiedene digitale Inhalte beziehen kann. Dazu wählt der Kunde einen Link, der den Inhalt identifiziert. Nach dieser Auswahl gelangt er zu einer weiteren Seite, auf der er seine Handynummer angibt.

Dieser Vorgang ist nur eine Möglichkeit. Der Ablauf einer Bestellung kann durchaus auch anders gestaltet sein. Wichtig ist es nur, dass der Kunde sich für einen digitalen Inhalt entschieden hat. Nach seiner Auswahl bekommt der Kunde einen *Download-Descriptor* auf sein Handy zugeschickt. Der *Download-Descriptor* gibt dem Kunden erste Informationen (Dateiname, Dateigröße, etc.) über das zu empfangene Produkt. Das Versenden des *Download-Descriptors* könnte z.B. über ein JAVA-Servlet geschehen. Abbildung 5.7 zeigt einen Code-Ausschnitt aus einem Servlet.

```
 1 DRMSystem drmsystem = DRMSystem.getDRMSystem();
 2 OMAWrapped packer = (OMAWrapped)drmsystem.getDRMWrapper("OMA");
 3
 4 byte[] plainData…..
 5
 6 packer.setContentDescription("geschütztes Wallpaper");
 7 packer.setContentName("Löwe");
 8 packer.setContentType("image/jpeg");
 9
10 packer.setContentUri("4711");
11 packer.setContentVendor("handy.de");
12 packer.setIconUri("http://drm.handy.de/iconUri");
13 packer.setRightsIssuer("http://drm.handy.de/GetRights?cUri=4711");
14 packer.setPlainData(plainData);
15 String dd = packer.getDownloadDescriptor("http://drm.handy.de/iconUri",
16                "http://drm.handy.de/deliveryServlet?contentUri=4711",
17                "http://drm.handy.de/notify.jsp",
18                "http://drm.handy.de/thanks.html",
19                "http://drm.handy.de/info.jsp",
20                "http://drm.handy.de/iconUri",
21                "installParam");
22
23 response.setContentType(„application/vnd.oma.dd+xml“);
24 out.write(dd.getBytes());
```

Abbildung 5.7: Download-Descriptor versenden

Im Beispiel wird zuerst eine Instanz von *DRMSystem* erstellt. Über diese Instanz wird ein DRMPackager Objekt angelegt, welches die OMA Implementierung kapselt (Zeile 2). Die Rohdaten des digitalen Inhaltes liegen in diesem Beispiel als `byte[]` vor (Zeile 4). Diese müssen zuerst in das DCF Format konvertiert werden. Aus diesem Grund werden dem DRMPackager in Zeile 6 bis 14 alle notwendigen Metadaten, sowie die Rohdaten selbst übergeben. In Zeile 13 wird dem DRMPackager mitgeteilt, von wo der Kunde das zugehörige Rechteobjekt beziehen kann. Diese Informationen werden im DRM geschützten Inhalt hinterlegt.

In der letzten Zeile wird der *Download Descriptor* erstellt. Dieser wird über den DRMPackager generiert. Da der *Download Descriptor* die Dateigröße des geschützten Inhaltes beinhaltet, wird intern (im DRMPackager) vorher die Konvertierung vorgenommen. Zeile 22 ist eine Angabe im *Download Descriptor,* die dem Endgerät die URL des digitalen Inhaltes mitteilt.

Nachdem der Kunde den *Download Descriptor* empfangen hat, bestätigt er den Empfang des Inhaltes. Die Auslieferung erfolgt wie in Abbildung 5.8 beschrieben.

```
 1 DRMSystem drmsystem = DRMSystem.getDRMSystem();
 2 OMAWrapped packer = (OMAWrapped)drmsystem.getDRMWrapper("OMA");
 3
 4 packer.setContentUri("4711");
 5 packer.setContentType("image/jpeg");
 6 byte[] enc = packer.createEncoded();
 7
 8 response.setContentType(„application/vnd.oma.drm.content“);
 9 response.setContentLength(enc.length);
10 response.setContentHeader(„X-Oma-Drm-Separate-Delivery“, „12“);
11
12 out.write(enc);
```

Abbildung 5.8: Den Inhalt ausliefern

Es müssen nicht wieder alle Metadaten dem DRMPackager übergeben werden. Diese wurden zuvor bei der Auslieferung des *Download Descriptors* übergeben.

Der erstellte DRM geschützte Inhalt ist im DRMPackager in einem Cache zwischengespeichert. Von daher muss nur noch deklariert werden, welches Objekt aus dem Cache bezogen werden soll (Zeile 4 und 5). Zeile 6 holt den zuvor erstellten Inhalt aus dem Cache und wird in Zeile 12 ausgeliefert.

Nachdem die Auslieferung angestoßen wurde, erhält der Kunde den DRM geschützten Inhalt. Dieser Inhalt ist so allerdings nicht zu gebrauchen. Der Kunde muss zuerst eine Nutzungslizenz erwerben, um den Inhalt freizuschalten. Der Download des eigentlichen Inhaltes ist an dieser Stelle vollständig. Der Erwerb der Nutzungslizenz kann auch später erfolgen.

Die Lizenz bezieht der Kunde über sein Handy. Das Gerät wird ihn darauf aufmerksam machen, dass er die im DRM geschützten Inhalt angegebene URL kontaktieren muss, um die Nutzungslizenz zu erwerben. Unter dieser URL ist wieder ein Servlet hinterlegt, welches die Auslieferung der Nutzungslizenz vornimmt.

```
 1 String cUri = request.getParameter("cUri"); // Produktnummer z.B. 4711
 2
 3 DRMSystem drmsystem = DRMSystem.getDRMSystem();
 4 DRMRightsManager rm = drmsystem.getRightsManager("OMA");
 5
 6 DRMRightsContraints rp = new RightsConstraints();
 7
 8 rp.setCount(count);
 9 rp.setRightsUsage(DRMRightsContraints.DISPLAY);
10 rm.addRightsParameter(rp);
11
12 DRMRights rights = rm.getRights(cUri);
13
14 byte[] wbxml = rights.toWbXml();
15
16 SMS sms = new SMS(wbxml);
17 sms.send();
```

Abbildung 5.9: Nutzungslizenz ausliefern.

In Abbildung 5.9 wird in Zeile 1 zuerst der URL-Parameter `cUri` gelesen. Dieser Parameter war bei der Auslieferung des Inhaltes in Abbildung 5.7 bzw. 5.8 zuvor übergeben worden.

Über ein Objekt der Klasse *DRMSystem* wird in Zeile 4 eine Instanz des *DRMRightsManager* erzeugt. Über diesen wird ein Objekt der Klasse *DRMRightsConstraints* bezogen, welches zur Formulierung der Nutzungsbedingungen dient. Zeile 8 und 9 legen die Nutzungsbedingung fest. Es beschreibt eine Nutzung des Inhaltes von genau drei Mal. Die Benutzung bedeutet in diesem Fall „Anzeigen“ (`DRMRightsConstraints.`***`DISPLAY`***). Dem *DRMRightsManager* wird dieses Objekt dann übergeben (Zeile 10). Damit ist man jetzt in der Lage das Rechteobjekt (Die Nutzungslizenz) zu erstellen (Zeile 12).

Da die Auslieferung der Nutzungslizenz über WAP-Push erfolgt, wird das Rechteobjekt in eine binäre Darstellung konvertiert (Zeile 14). Die

Auslieferung erfolgt dann durch die Klasse SMS (Nicht Bestandteil des DRM-Frameworks). Der Kunde erhält nach wenigen Sekunden eine SMS, die die Nutzungslizenz beinhaltet. Nachdem der Kunde diese gespeichert hat, ist der Inhalt, gemäß der Nutzungslizenz, freigeschaltet.

Nach dreimaliger Betrachtung ist die Lizenz ungültig und muss wieder neu bezogen werden. Im gezeigten Beispiel ist kein Bezahlprozess implementiert. Dieser ist Bestandteil der Auslieferungsplattform und kann an unterschiedlichen Stellen auftreten.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Der Schutz von digitalen Inhalten gewinnt, mit der Verbreitung des Internets als neuen Vertriebsweg, immer mehr an Bedeutung. Die Möglichkeit Daten jederzeit und in nahezu beliebiger Menge zu beziehen, zwingt Anbietern von digitalen Inhalten über Schutzmaßnahmen nachzudenken. Der Schutz des geistigen Eigentums ist primäres Ziel welches von Digital Rights Management Systemen verfolgt wird. Der Schutz bezieht sich dabei nicht auf das Verhindern von Kopien, sondern vielmehr darauf die Nutzung einzuschränken.

DRM-Systeme verschlüsseln die digitalen Inhalte und machen sie nur im Zusammenhang mit einem Nutzungsrecht brauchbar. Die Verfahren, die bei dieser Technik zum Einsatz kommen, basieren auf kryptographische Verschlüsselungstechniken. Die digitalen Inhalte werden dabei verschlüsselt und können somit durch kopieren weitergegeben werden. Die Nutzung kann nur im Zusammenhang mit dem zugehörigen Rechteobjekt erfolgen, welches den kryptographischen Schlüssel enthält, der notwendig ist um den Inhalt lesbar zu machen.

Die Architektur heutiger DRM-Systeme basiert im Kern auf der DRM Referenz-Architektur (vorgestellt in Kapitel 2.3.1). Diese Tatsache habe ich in meiner Lösung als Ausgangspunkt betrachtet, um eine einheitliche Sicht auf die unterschiedlichen DRM-Systeme zu bekommen.

6.1.1 Abgleich an die Anforderungen

Nach der Erkenntnis, dass DRM-Systeme im Allgemeinen die gleiche Architektur zu Grunde liegt, ist es möglich ein API zu erstellen, das allgemeine Schnittstellen für die Benutzung der DRM-Systeme definiert (**Punkt 1** aus den Anforderung in **Kapitel 3**).

Dieses API wird in ein Framework verpackt, das durch die Konfiguration über XML einfach erweiterbar ist. Ein neues DRM-System lässt sich durch Erweitern und Implementieren der vordefinierten Schnittstellen leicht in das Framework integrieren. Diese Anforderung, beschrieben in **Punkt 5** aus **Kapitel 3** ist damit umgesetzt.

Die Anforderung das Framework leicht in die Auslieferungsplattform integrieren zu können erfüllt sich ebenfalls. Durch die Definition eines eindeutigen API kann diese leicht um DRM-Funktionalität erweitert werden. Damit erfüllt sich **Punkt 2** aus den Anforderungen in **Kapitel 3**.

Die Nutzungsbedingungen werden mit Hilfe der Klasse *DRMRightsConstraints* einheitlich definiert. Diese können dem *DRMRightsManager* zur Behandlung übergeben werden, der damit dann ein DRM abhängiges Rechteobjekt erstellen kann. Die Transformation in die DRM abhängige Darstellung erfolgt in der jeweiligen Klasse, welche das Rechteobjekt repräsentiert. Damit erfüllt sich **Punkt 3** aus den Anforderung in **Kapitel 3**.

Das Erstellen von Schlüssel kann durch viele Zugriffe zu einer hohen Belastung des Servers führen. Dies ist ein Grund warum die symmetrischen Schlüssel in einem Zwischenspeicher abgelegt werden. Es muss weiterhin gewährleistet sein, dass bei jedem Zugriff auf ein Produkt der gleiche symmetrische Schlüssel ausgeliefert wird. Von daher werden die Schlüssel in einer Datenbank abgelegt (**Punkt 4** der Anforderung in **Kapitel 3**).

Punkt 6 der Anforderungen aus **Kapitel 3** verlangt die Implementierung der OMA-DRM Spezifikation. Dieses System habe ich als Referenzimplementierung für das DRM-Framework umgesetzt. Dabei wurde speziell der Auslieferungsmechanismus *separate delivery* umgesetzt, der Inhalt und Rechteobjekt jeweils separat ausliefert.

6.2 Ausblick

Ein letzter Punkt der Anforderungen welcher als Ausblick gedacht ist, wäre die Einführung einer Benutzerverwaltung. Diese könnte vorsehen, dass der Käufer eines Produktes mit seiner Handynummer registriert wird. Der DRM geschützte Inhalt, der dann erstellt wird enthält diese Handynummer. Sollte der Käufer das Produkt an einen Dritten weitergeben muss dieser, um das Produkt verwenden zu können, ein Rechteobjekt erwerben. Beim Erwerb dieses Rechteobjekts durch den Dritten kann dem Anbieter mitgeteilt werden, wer der ursprüngliche Käufer war und diesem eine Provision gutgeschrieben werden.

Für diesen Mechanismus ist keine Änderung des DRM-Frameworks notwendig. Es muss lediglich ein „Add-On“ erstellt werden, welches die Verwaltung übernimmt. Die Handynummer kann über das API des DRM-Frameworks mit in die Erstellung der DRM geschützten Inhalte einfließen. Das Einführen einer Provision eröffnet neue Möglichkeiten für den Anbieter und für den Kunden. Für den Kunden wird es attraktiver DRM-geschützte Inhalte zu erwerben, da er für die Weitergabe und der späteren Aktivierung, eine Provision erhält. Der Anbieter behält dabei die Kontrolle der Nutzungsrechte und kann attraktive digitale Inhalte anbieten.

7 Glossar und Literaturverzeichnis

7.1 Glossar

Chiffretext (Geheimtext)
In der Kryptographie bezeichnet Geiheim-, Krypto-, Cipher-, Chiffre-, bzw. Schlüsseltext aus einem Geheimtextraum eine verschlüsselte Nachricht. Sie ist durch Verschlüsselung aus einem Klartext entstanden.

DRM
siehe. „Digital Rights Management“

Digital Rights Management
Digital Rights Management ist ein umschließender Begriff für bestimmte Mechanismen, welche die Nutzung von kopiergeschützten, digitalen Inhalten durch den Urheber einschränken.

FTP
Das File Transfer Protocol (engl. Datenübertragungsverfahren) ist ein in RFC959 spezifiziertes Netzwerkprotokoll zur Dateiübertragung über TCP/IP-Netzwerke. FTP ist in der Anwendungsschicht von TCP/IP angesiedelt und wird benutzt um Dateien zwischen Client und Server zu transferieren. Anders als HTTP benutzt FTP mehr als eine Verbindung zur Kommunikation. Zuerst wird durch den Port 21 eine Verbindung zur Authentifizierung und Befehlsübertragung geöffnet. Der Server reagiert auf diesem Kanal mit einem Statuscode. Zur eigentlichen Datenübertragung wird dann eine weiter Verbindung eröffnet.

GPRS
Bei GPRS (General Packet Radio Service) handelt es sich um eine Entwicklung des GSM-Mobilfunk-Standards um paketorientierte Datenübertragung. Die Daten werden beim Sender in Pakete umgewandelt, als solche übertragen und beim Empfänger wieder zusammengesetzt. Die GPRS-Technik ermöglicht bei der Bündelung aller 8 GSM-Kanäle theoretisch eine Datenrate von 171,2 kBits/s. Im praktischen Betrieb ist die Anzahl der parallel nutzbaren Kanäle jedoch durch die Fähigkeit des Mobilgerätes begrenzt.

JAR
Eine JAR Datei ist ein Archivformat, welches SUN zum Verbund von Klassen und Mediendaten einführte. Alle Klassen einer Komponente können so innerhalb eines Archivs gebündelt werden.

Kapselung
Unter Kapselung versteht man ein objektorientiertes Programmierkonzept, bei dem zusammengehörige Daten und Funktionen zusammengefasst werden. Diese bilden eine Klasse. Der Zugriff (lesend und schreibend) auf die Daten

ist nur über eine vordefinierte Schnittstelle möglich. Daten (Variablen) können nur über Funktionen verwendet werden.

MMS
MMS (Multimedia Messaging Service) ist als Nachfolger von SMS (Short Message Service) und EMS anzusehen. Es bietet die Möglichkeit mit einem Mobiltelefon multimediale Nachrichten zu anderen mobilen Endgeräten zu versenden. Im Gegensatz zu SMS, welches nur kurze Nachrichten (in der Regel bis zu 160 Zeichen) erlaubt, ist es mit MMS möglich längere Nachrichten, die auch Formatierung (fettgedruckt, schräggestellt, u.ä.) und kleinere Bilder beinhalten zu erzeugen.

Persistenz
Persistenz bezeichnet die Fähigkeit Datenstrukturen (oder Objekte) in nicht-flüchtigen Speichermedien wie Dateisystemen oder Datenbanken zu speichern. Datenstrukturen, die diese Fähigkeit nicht besitzen existieren nur im Hauptspeicher des Computers und gehen verloren, sobald das Programm endet. Persistente Datenstrukturen können dagegen beim Ende des Programms gespeichert und beim erneuten Start aus dem Speichermedium wiederhergestellt werden.

PDA
„Personal Digital Assistent“. Dies sind Geräte welche ursprünglich als elektronische Organizer konzipiert wurden. Über die Jahre wurden diese Geräte immer vielseitiger. So vereinen moderne Geräte viele Funktionen wie Organizer, Uhr/Wecker, Adressbuch, Task List, Notizblock und Taschenrechner in einem Gerät. Ein großer Vorteil von PDA’s ist die Möglichkeit seine Daten mit einem Desktop-Computer zu synchronisieren.

URL
Uniform Resource Locator ist ein Uniform Resource Identifier (URI) der eine Ressource über ihren primären Zugriffsmechanismus, d.h. dem Ort (engl. Location) der Ressource im Internet identifiziert. Der Begriff URL wird (historisch bedingt) oft synonym zum Begriff Uniform Resource Identifier (URI) verwendet, obwohl es sich bei URLs nur um eine Unterart von URIs handelt. In den meisten Fällen, in denen der Begriff URL verwendet wird, sind jedoch tatsächlich Uniform Resource Identifier gemeint

W3C
Das World Wide Web Consortium, oder auch W3C, ist das Gremium zur Standardisierung des World Wide Web betreffender Techniken. Gründer und Vorsitzender ist Tim Berners-Lee, der auch als Erfinder des World Wide Web bekannt ist. Es wurde 1994 gegründet.

WBXML

WBXML dient dazu, die im XML-Format vorliegenden Informationen in Binärdaten umzuwandeln und in eine Form zu bringen, die WAP-Clients beziehungsweise deren Mikro-Browser interpretieren können. Das WBXML-Format berücksichtigt die Struktur von XML-Dokumenten und ermöglicht es dem Browser, unbekannte Elemente oder Attribute zu überspringen. Meta-Informationen, wie etwa die Definition des Dokumententyps, werden bei der Umwandlung des Dokuments in das Binärformat entfernt.
WAP Binary XML wurde unter anderem deshalb entwickelt, um die Größe der zu übertragenden XML-Dokumente zu reduzieren. Hintergrund ist die geringe Bandbreite von meist nur 9,6 kBit/s, die bislang in GSM-Mobilfunknetzen für Datentransfers zur Verfügung steht.

WAP-Push
Eine der entscheidenden Neuerung der WAP-Spezifikation 1.2 stellt die so genannte WAP Push Architecture dar. Dies ist auch der Bereich, in dem innerhalb des letzten halben Jahres die meisten neuen Spezifikationen entstanden oder zumindest erweitert worden sind. Ziel der WAP-Push-Architektur ist es, Inhalte verschiedenster Art vom Server zum mobilen Client zu ‘schieben’. Dieses Pushing ist somit ein erster Schritt im Rahmen der zukünftigen Planungen des WAP-Forums, Informationen und Daten auf Initiative des Servers hin zum Client übertragen zu können.
Im heute gängigen Client/Server-Modell stößt der Client zunächst einen Request an, um einen Service oder Informationen vom Server abzurufen. Diesen Vorgang nennt man im allgemeinen Pull-Mechanismus. Das WWW stellt einen typischen Vertreter dieser Kategorie dar. Der Benutzer gibt eine URL als Request an den Server an, die der Server mit einer entsprechenden Webseite als Response beantwortet. Beim Push-Mechanismus dagegen ist der Server auch ohne vorangehenden Request in der Lage, Informationen auf eigene Initiative hin an den Client zu übertragen.

XML
Die Extensible Markup Language, abgekürzt XML, ist ein Standard zur Erstellung strukturierter maschinen- und menschenlesbarer Dateien. XML definiert dabei den grundsätzlichen Aufbau solcher Dateien. Für die konkreten Anwendungsfälle müssen die Details des Dateiaufbaus jedoch weiter spezifiziert werden. XML ist eine Teilmenge SGML.

7.2 Literaturverzeichnis

[microsoft.com]
http://www.microsoft.com/windows/windowsmedia/drm/default.aspx
Internetseite zum Thema „Digital Rights Management“
Zeitpunkt des Zugriffs: 13. Februar 2005 18:00 Uhr

[ffa.de]
http://www.ffa.de/start/content.phtml?page=digikino_technik_drm
Filmförderungsanstalt, German Federal Film Board
Zeitpunkt des Zugriffs: 12. Februar 2005 17:00 Uhr

[wikipedia.org]
http://en.wikipedia.org/wiki/Software_framework
Software Framework.
Zeitpunkt des Zugriffs: 13. Februar 2005 18:00 Uhr

[Becker-2003]
Eberhard Becker, Willms Buhse, Dirk Günnewig, Niels Rump: Digital Rights Management, Technological, Economic, Legal and Political Aspects, Springer-Verlag Berlin 2003

[thefreedictionary.com]
http://www.thefreedictionary.com/framework
Ein Online-Dictionary
Zeitpunkt des Zugriffs: 10. Februar 2005 13:00 Uhr

[Fuhrberg-2001]
Dr. Kai Fuhrberg, Dr. Dirk Häger, Dr. Stefan Wolf: Internet-Sicherheit, Browser, Firewalls und Verschlüsselung., Carl Hanser Verlag 2001

[Fischer-1998]
Dr. Stephan Fischer, Dipl.Wirtsch.-Inf. Achim Steinacker, Dipl.-Ing. Reinhard Bertram, Prof. Dr.-Ing. Ralf Steinmetz: Open Security, Von den Grundlagen zu den Anwendungen, Springer-Verlag Berlin 1998

[Schmeh-2001]
Klaus Schmeh: Kryptographie und Public-key-Infrastrukturen im Internet, 2. aktualisierte Auflage, dpunkt-Verlag 2001.

[Beutelsbacher-2000]
Albrecht Beutelsbacher: Geheimsprachen, Geschichte und Techniken, Verlag C.H. Beck München 1997

[Rosenblatt-2001]
William Rosenblatt, William Trippe, Stephen Mooney: Digital Rights Management, Business and Technology, John Wiley & Sons 2001

[Eckert-2004]
Claudia Eckert: IT-Sicherheit, Konzepte, Verfahren, Protokolle, 3. überarbeitete Auflage 2004, Oldenbourg Verlag2004

[odrl.net]
http://odrl.net/1.1/ODRL-11.pdf
ODRL-Spezifikation
Zeitpunkt des Zugriffs: 07. Februar 2005 16:00 Uhr

[openmobilealliance.org]
http://www.openmobilealliance.org
Homepage der Open Mobile Alliance. Hier liegen alle Spezifikationen zum Download bereit.
Zeitpunkt des Zugriffs: 13. Februar 2005 18:00 Uhr

[OMA DRM Architecture]
http://www.openmobilealliance.org/release_program/drm_v10.html
Die Architektur des OMA DRM Standards.
Zeitpunkt des Zugriffs: 11. Februar 2005 18:10 Uhr

[adobe.com]
http://www.adobe.com/products/contentserver/main.html
Webseite der Firma Adobe, Hier speziell: Adobe Content Server.
Zeitpunkt des Zugriffs: 03. Januar 2005 19:00 Uhr

[Siedersleben-2004]
Johannes Siedersleben: Moderne Softwarearchitektur, dpunkt.verlag GmbH, Heidelberg 2004

[Gamma-2001]
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Entwurfsmuster. Elemente Wiederverwendbarer objektorientierter Software, Addison-Wesley Verlag 2001.

[java.sun.com]
http://java.sun.com/products/jdbc/
Webseite der Firma SUN, Hier speziell: JDBC (Java Database Connectivity).
Zeitpunkt des Zugriffs: 11. Februar 2005 12:00 Uhr

[Esser-2001]
Prof. Dr. Friedrich Esser: Java 2, Patterns, Idioms, Java-Zertifizierung. Galileo Computing 2001.

Anhang

DRM Content Format

Feldname	Typ	Zweck
Version	Uint8	Versionsnummer
ContentTypeLen	Uint8	Länge des ContentType Feldes
ContentURILen	Uint8	Länge des ContentURI Feldes
ContentType	ContentTypeLen octets	Der MIME Type der Rohdaten
ContentURI	ContentURILen octets	Der unique identifier des Inahltes
HeadersLen	Uintvar	Länge der Header Daten
DataLen	Uintvar	Länge der Nutzdaten
Headers	HeadersLen octets	Die Metadaten die im Header definiert werden.
Data	DataLen octets	Die verschlüsselten Daten

Abbildung 7.1: DRM Content Format

Auslieferung der DRM geschützten Inhalte

Die Auslieferung der Inhalte geschieht über das Protokoll HTTP. Die Nachricht verlangt folgende Angaben im Header:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.drm.content;
Content-Length: 1234
X-Oma-Drm-Separate-Delivery: 12

...DRM content in DCF format...
```

Abbildung 7.2: Auslieferung von DCF Objekt

Document Type Definition

```
<!ELEMENT drm (drmsystems, keystorage, rightsstorage, resource-ref)>
<!ELEMENT drmsystems (implementation+)>
<!ELEMENT implementation (id, uri, contentwrapper, rights)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT uri (uriprefix, uripostfix)>
<!ELEMENT rights (class)>
<!ELEMENT keystorage (class)>
<!ELEMENT rightsstorage (class)>
<!ELEMENT resource-ref (res-ref-name)>
<!ELEMENT contentwrapper (interface, class)>
<!ELEMENT interface (#PCDATA)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT uriprefix (#PCDATA)>
<!ELEMENT uripostfix (#PCDATA)>
<!ELEMENT res-ref-name (#PCDATA)>
```

Abbildung 7.3: DTD für drm.xml

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 17.02.2005