

Digital Rights Management

Business and Technology

Bill Rosenblatt, Bill Trippe,
and Stephen Mooney

"This book paints a complete picture of the overall DRM landscape in terms that novices can understand, without sacrificing the under-the-hood details that techies demand."

— Mark Walter
Senior Analyst
The Seybold Report

Visit us at mandtbooks.com

P
R
O
F
E
S
S
I
O
N
A
L

M
I
N
D
W
A
R
ETM



NF-1011

U.S. Patent 10,726,102

Digital Rights Management

Business and Technology

**William Rosenblatt, William Trippe,
and Stephen Mooney**



M&T Books

An imprint of Hungry Minds, Inc.

Best-Selling Books • Digital Downloads • e-Books • Answer Networks •
e-Newsletters • Branded Web Sites • e-Learning

New York, NY • Cleveland, OH • Indianapolis, IN

Digital Rights Management: Business and Technology

Published by
M&T Books
909 Third Avenue
New York, NY 10022
www.hungryminds.com

Copyright © 2002 Hungry Minds, Inc. All rights reserved.
No part of this book, including interior design, cover design, and icons, may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

Library of Congress Control Number: 2001096080

ISBN: 0-7645-4889-1

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/TR/RQ/QR/IN

Distributed in the United States by Hungry Minds, Inc.
Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria, and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS

Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For general information on Hungry Minds' products and services please contact our Customer Care department within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For sales inquiries and reseller information, including discounts, premium and bulk quantity sales, and foreign-language translations, please contact our Customer Care department at 800-434-3422, fax 317-572-4002 or write to Hungry Minds, Inc., Attn: Customer Care Department, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

For information on licensing foreign or domestic rights, please contact our Sub-Rights Customer Care department at 212-884-5000.

For information on using Hungry Minds' products and services in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 317-572-3168 or fax 317-572-4168.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK. THE PUBLISHER AND AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTIONS CONTAINED IN THIS PARAGRAPH. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ACCURACY AND COMPLETENESS OF THE INFORMATION PROVIDED HEREIN AND THE OPINIONS STATED HEREIN ARE NOT GUARANTEED OR WARRANTED TO PRODUCE ANY PARTICULAR RESULTS, AND THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY INDIVIDUAL. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

Trademarks: Hungry Minds, the Hungry Minds logo, M&T Books, the M&T Books logo, and Professional Mindware are trademarks or registered trademarks of Hungry Minds in the United States and other countries and may not be used without written permission. Adobe is a trademark or registered trademark of Adobe Systems Incorporated in the United States and/or other countries. All trademarks are the property of their respective owners. Hungry Minds, Inc., is not associated with any product or vendor mentioned in this book.



Hungry Minds[®] is a trademark of Hungry Minds, Inc.



is a trademark of Hungry Minds, Inc.

Chapter 5

DRM Building Blocks: Protecting and Tracking Content

Many types of technology components fall under the rubric of “DRM.” These include subscription systems such as those found on *The Wall Street Journal* and *Consumer Reports* Web sites, paid download services like QPass and ProQuest; business-to-business (B-to-B) content syndication technologies from vendors such as Vignette and Kinecta and services such as YellowBrix and Screaming Media, and various types of print-on-demand services. However, none of these technologies actually enforces rights models. They may describe rights or leave them to written license agreements, but otherwise they implement fairly standard components such as file transfer protocols, user registration databases, and e-commerce payment processing.

In this chapter, however, we concentrate on the components of systems that use technology to control distribution — to enforce rights models. It turns out that most DRM solutions are variations on a common theme — a DRM reference architecture. In this chapter, we present a DRM reference architecture and examine each of its components. We also discuss the two most prevalent core technologies involved in DRM implementation: encryption and watermarking.

We don’t say much here about specific vendors’ offerings; see Chapters 7, 11, and 12 for those details. Our intent in this chapter is to give you a solid understanding of the general principles of DRM technology to help you evaluate solutions that you may want to adopt. Vendors come and go, but these principles should remain for the foreseeable future.

A DRM Reference Architecture

The components of the DRM reference architecture range from those that must be behind the content provider’s firewall to those that may be elsewhere — on the consumer’s hardware or on a server at a DRM service provider. There are three major components: the content server, the license server, and the client.

The content server

The content server component of the DRM architecture is the one that is most likely to sit behind the content provider’s firewall. It consists of the actual content, information about products (or services) that the content provider wants to distribute, and functionality to prepare content for DRM-based distribution.

Content repository

A content provider who implements a DRM solution has a *repository* of content, which is a collection of content that's either in a suitable format for distribution through the DRM system or (as depicted in Figure 5-1) can be put into the correct format on demand. The repository is often built into the DRM solution, or sometimes the DRM solution interfaces to a content management system that serves multiple purposes (see Chapter 10). Some publishers are building content management systems that serve several distribution channels simultaneously, such as print, Web, syndication (see Chapter 6), and controlled consumer distribution via DRM. From a purely pragmatic standpoint, the repository is a file server or a database system. In either case, it contains content as well as *metadata*, or information about the content.

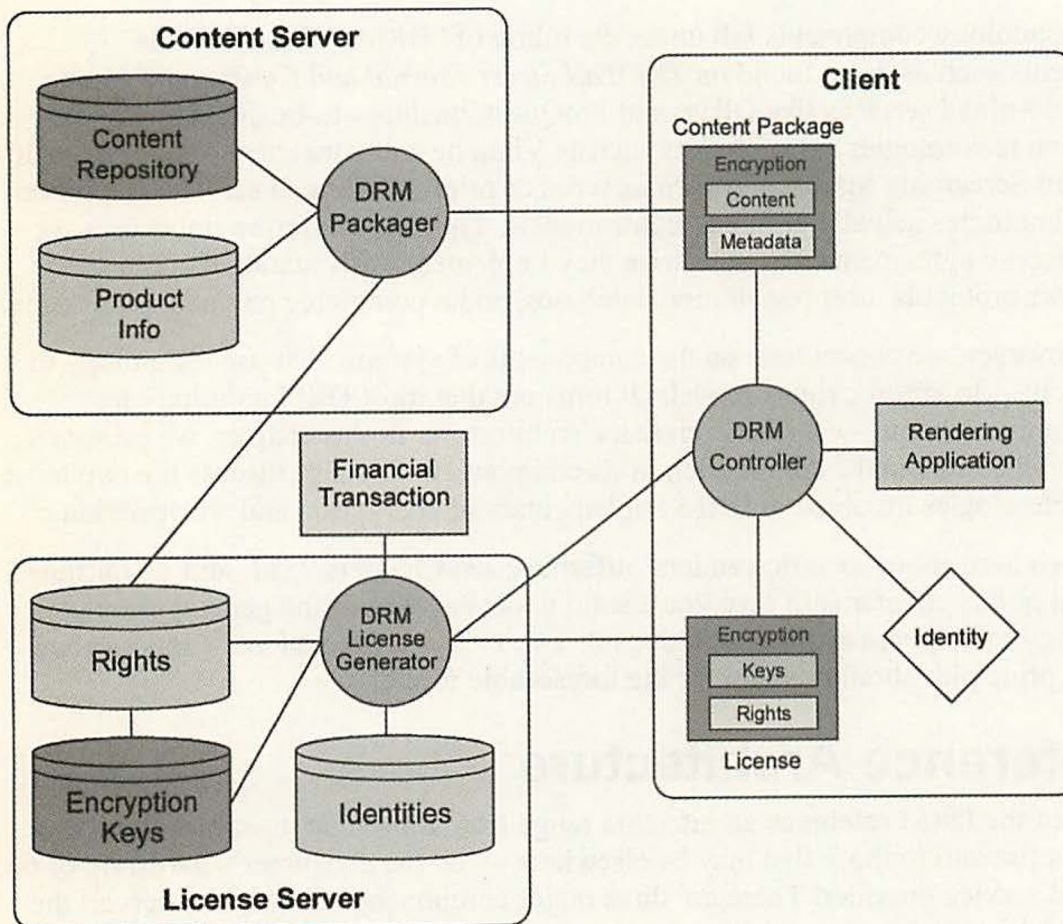


Figure 5-1: The DRM reference architecture.

Product info

Content providers typically have catalogs of product information for their products in physical media. These contain metadata about products, such as price, marketing information, format, physical dimensions, and so on. In implementing DRM, content providers also need to create databases of information about products they intend to sell. If the DRM system is integrated with an e-commerce storefront, the product info is inserted into the storefront system's product catalog.

Notice that both the content repository and the product info database contain metadata. There is a subtle yet important difference between content and product metadata: Price is clearly product

metadata, for example, whereas an internal revision number would be content metadata. Other elements of metadata, such as title, author, number of pages, and file format could be either.

DRM packager

Every encryption-based DRM system contains functionality for preparing content for distribution through the system. We call this functionality the *DRM packager* or *content packager*. The packager does its job either before putting the content into the repository, or (as Figure 5-1 shows) on-the-fly before distribution. Various vendors have trademarked names for content packages, such as InterTrust's DocBox for PDF files and SealedMedia's SoftSeal.

In addition to the content itself, the content package contains metadata. There are various kinds of metadata, but two in particular are prevalent in DRM content packages:

- ◆ **Identification:** A unique number that the publisher can use as a reference for various purposes, such as getting up-to-date pricing information and tracking content usage remotely
- ◆ **Discovery:** Information that can help users locate the content, such as keywords, title, and author

In some DRM systems, the content packages also include metadata on content usage. This is especially common in DRM systems meant to be used offline. In these systems, usage can be tracked locally to the content package and periodically reported back to a server.

Both the content and the metadata in a package are usually encrypted. We discuss encryption in more detail later in this chapter, in the section "Encryption." The metadata does not need to be encrypted, but it must at least be tamperproof (unable to be changed) and inextricably tied to the content. The content package can exist as an indivisible unit; alternatively, the package can simply contain metadata and a link to the content through the identification metadata. The latter is the usual case for streaming media formats (see the "Streaming media" section a little later in the chapter).

The other important task of the packager is to create descriptions of the rights that the content provider wants to allow the user to exercise on the content. As we saw in Chapter 4, DRM solutions enable content providers to specify rights through a user interface (one content item at a time), through a batch process, or through a programming interface for a language such as C++ or Java. This is part of the packaging process.

What happens after rights are specified varies with different DRM technologies. In the older technologies, such as InterTrust's DigiBox (see Chapter 7), rights were bundled in with the content package and sent to the user. Although this scheme is the most straightforward to implement, it has some major shortcomings, such as the following:

- ◆ You may want to specify multiple sets of rights for a given piece of content, such as for different types of users. It is more cumbersome to implement this if rights are packaged along with the content.
- ◆ Conversely, you may want to offer a set of rights that applies to more than one piece of content, such as a subscription service to a body of content. Bundling rights with individual content items makes this difficult to implement.
- ◆ Certain types of content, such as streaming media, must reside on servers rather than on the user's device or hard drive. In that case, you can't distribute the content to users, but you still must distribute information about their access rights.

Therefore, more modern DRM implementations separate rights information from content packages by encapsulating the former in *licenses* (also called *permits* or *tickets*).

The license server

Licenses contain information about the identity of the user or device that wants to exercise rights to content, identification of the content to which the rights apply, and specifications of those rights, which conceptually resemble those we saw in Chapter 4.

A reasonable analogy to DRM licenses in the real world is tickets for planes and trains. When you buy a ticket on the 6 p.m. American Airlines flight from LaGuardia in New York to O'Hare in Chicago, you must show a photo ID to get a boarding pass and get on the plane. The ticket and boarding pass give you (and only you) the right to board that plane (and only that plane). Or if you're a train commuter, you may buy a monthly pass on the New Jersey Transit between Princeton Junction and New York. In that case, you (and only you) get the right to ride certain trains (trains from Princeton Junction to Penn Station and back) every day for a month. Analogously, you may get a DRM license that gives you access to one specific piece of content, or you may get a license to access any item in a collection of content over a certain period of time.

There are several ways to implement licenses; the one we depict in the DRM reference architecture is fairly typical. The DRM packager creates specifications of rights and hands them off to a second set of server components, which we call the license server. In addition, the DRM packaging process creates a set of *encryption keys* that are used to authenticate users and decrypt content. Rights specifications and keys are stored in separate databases. It is, of course, particularly important that the latter be tightly secured. Each of these databases contains unique identifiers – the identification metadata mentioned previously – that link rights and key information to particular content items.

The license server also stores *identities*, which are information about users who exercise rights to content. As we see later in the section “Identification,” the identities of users can take different forms, such as usernames, device information, or biometrics. In our train example, the identification takes the form of the rider's signature. Taken together, identities, rights specifications, and encryption keys constitute sufficient information for the *license generator* to create licenses and send them to users. We see how this is done shortly.

The client

Components of the DRM reference architecture that reside on the user's side are the DRM controller, the rendering application, and the user's identification mechanism. We use the term *client* not only because it is analogous to *server*, but also because the client is really a combination of the user and the device that the user is using.

The DRM controller

The client side contains a collection of functionality that we call the *DRM controller*. Vendors' names for this type of component include Microsoft's Black Box and the Secure Digital Music Initiative's (see Chapter 6) Licensed Compliant Module. The DRM controller is the real nerve center of the DRM system. Figure 5-1 (earlier in this chapter) depicts the DRM controller as an independent piece of software, but it can also reside within a rendering application (player, viewer, and so on), or it can be a piece of dedicated hardware.

The DRM controller does several things:

- ◆ Receives the user's request to exercise rights on a content package.
- ◆ Gathers the user's identity information and obtains a license from the license server.
- ◆ Authenticates the application that performs the rights exercise, such as a rendering application.
- ◆ Retrieves encryption keys from the license, decrypts the content, and releases it to the rendering application.

A typical sequence of events inherent in DRM use on the client side is shown in Figure 5-2.

The first thing that happens is that the user obtains a content package (number 1 in the figure). This could happen in several ways, such as by downloading it from a Web site or FTP server, detaching it from an e-mail message, or reading it from physical media such as a CD-ROM. At some point after the user gets the content package, she makes a request to exercise rights on it (2). The user may do this by choosing a menu item in a rendering application, executing an operating system command, double-clicking a file, or whatever — we say generically that she “activates a DRM controller.”

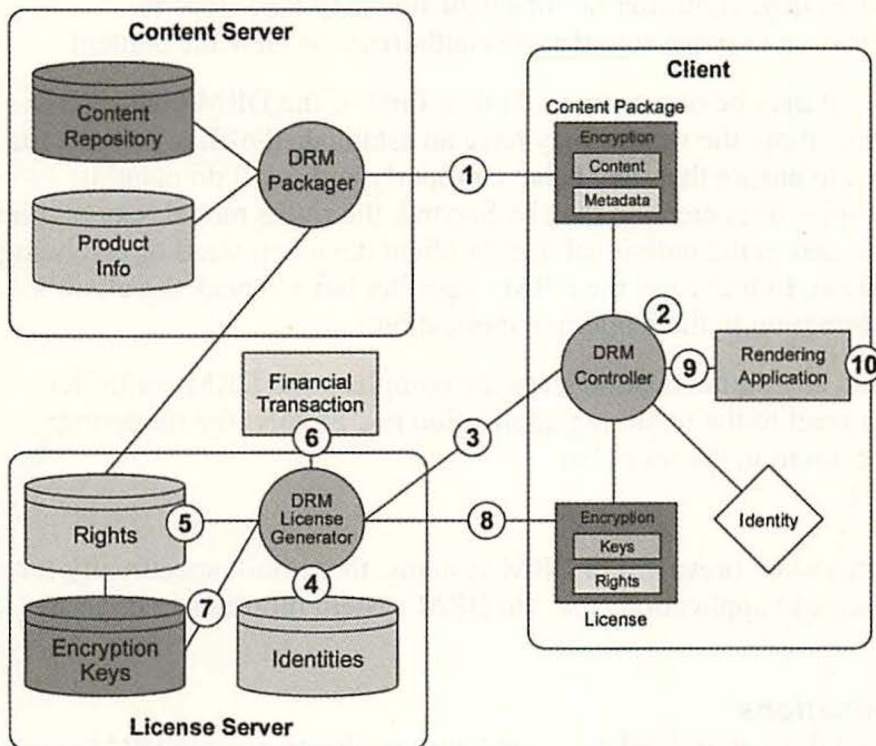


Figure 5-2: Sequence of events in exercising rights to content.

For the remainder of this discussion, assume for the sake of concreteness that the right being exercised is a render right. Thus, a rendering application is necessary to exercise the right. However, it's worth noting that for some transport rights, such as copy and lend, the DRM controller could conceivably perform the job itself. Exercising a copy or lend right involves constructing another content package, which can involve nontrivial tasks such as encryption and registration with content and/or license servers.

Once activated, the DRM controller gathers information necessary for generating a license. In addition to the information the user has already specified about which rights she wants to exercise, this includes obtaining identity information for the user and/or client device and information from the content package, including the content identifier. If the user hasn't used

this DRM system before, the DRM controller must create the identity. This may entail getting the user to fill out a registration form (possibly containing a credit card number) or clicking to accept an end-user license agreement. After it's created, the identity is passed to the license server for insertion into the database of identities.

The DRM controller then sends the identity and content information in a request to the license server (3). The license generator authenticates the client's identity against its identities database (4) and uses the content identifier to look up rights information about the content (5). Then it gathers rights information from the user's license request. If necessary, it kicks off a financial transaction at this point (6). (In some DRM systems, the transaction takes place earlier in the process – sometimes even before the user obtains the content package in the first place.) Or, if the consideration specified in the license is something other than financial, that consideration can be given – the user fills out a demographic survey, gives up some of her Frequent Reader points, or whatever. Finally, the license generator pulls together rights information, client identity information, and encryption keys (7), and creates a license – which is itself encrypted or at least tamperproofed. It sends the license back to the client (8).

At some point in this process, the DRM controller on the client side may take steps to authenticate the rendering application to make sure that it is authorized to view the content.

There are two main reasons why it may be necessary to do this. First, if the DRM controller is separate from the rendering application, the vendor may have an established offline process for certifying rendering applications to ensure that they behave properly and don't do naughty things like make unencrypted copies of content on the sly. Second, the rights model may require that the rendering application be tied to the individual user or client device, instead of just being some generic rendering application. In that case, the DRM controller has to check the client's identity against registration information in the rendering application.

After the license is generated and any authentication steps are complete, the DRM controller can decrypt the content and release it to the rendering application (9). Finally, the rendering application plays or shows the content to the user (10).

Rendering applications

Two types of rendering applications are prevalent in DRM systems: those built specifically for the DRM system and general-purpose applications that the DRM system modifies in order to restrict their behavior.

Standalone rendering applications

Early DRM systems incorporated their own standalone rendering applications with DRM controllers built in. The great advantage of this approach is that the purpose-built rendering application can be made airtight: It can allow only the specific functions that the user is authorized to perform. But this approach has bigger disadvantages. First, the vendor has to distribute the application to its users — which can be done over the Internet or even through floppies or CD-ROMs. Then, the user has to install and learn how to use the new application.

All this is a major hassle. InterTrust's DigiBox, which originally included a standalone viewer, suffered from this problem. The application was several megabytes in size, making it virtually impossible to download over the Internet for users with dial-up connections. InterTrust tried easing the distribution problem by cutting deals with major software vendors; for example, they got the software included with the CD-ROM for America Online. The only successful standalone rendering applications with DRM built in nowadays are those for popular new

formats or devices, such as Microsoft Reader for the eBook format on PCs and handhelds. In other words, obviously, good file formats and rendering applications are far more important to users than are DRM controllers. Yet even in those cases, users must still download DRM controller functionality unless it is bundled in with their devices (as it is, for example, with Microsoft Windows Media Player on PCs).

Plug-ins

The second approach, more common for PC platforms nowadays, is to start with general-purpose playing or viewing applications and modify their behavior. These applications tend to do lots of things that DRM systems don't want them to. Common applications such as Adobe Acrobat Reader have functions like Print, Save As, and Copy to Clipboard, which let the user do things with the content that the DRM system may not want done. The underlying operating system also has functions that are accessible within general-purpose applications that let users do things that they shouldn't, such as the Alt+Prnt Scrn function in Microsoft Windows that lets you to make a bitmap copy of the window open on your screen.

Adobe Acrobat Reader in particular has an architecture that allows for *plug-ins*, which are pieces of code that act as DRM controllers and change the Reader's behavior. Several DRM vendors have implemented Acrobat plug-ins that perform tasks such as prompt the user for a password or collect payment. Plug-ins can also turn off functions such as Print and Save As, but they generally can't override operating system-level functions like Alt+Prnt Scrn. The Netscape Web browser also has a plug-in architecture. Microsoft applications such as Internet Explorer achieve similar results through ActiveX controls.

The plug-in approach has a few advantages: It's likely that users either have the base application already or at least will be motivated to get it for reasons other than viewing one specific piece of content that happens to be protected by some DRM mechanism. Also, the plug-in code is usually smaller in size than a full application would be, which makes distribution less problematic. The disadvantage is that, as implied earlier, it's generally not possible to make an existing application with a plug-in as secure as one that's purpose-built.

Java technology

The plug-in approach still has the disadvantage of requiring users to download and install a piece of software. Some users are afraid to do this; others just don't want to. In the late 1990s, a third approach suggested itself that would virtually eliminate this problem. The silver bullet was Sun Microsystems' Java technology.

Programs written in the Java programming language are supposed to be able to run on any platform that has a core piece of software called a *Java Virtual Machine*, or JVM. JVMs have been implemented for many different platforms, including PCs, Macintoshes, UNIX systems, cell phones, PDAs, and the major Web browsers. The idea is that you can download any piece of Java code (called an *applet*) into your environment and have it run without needing to go through an installation process, as would be necessary with plug-ins or other types of software.

It was thought that you could implement a DRM controller in Java that ran inside a Web browser or standalone on a PC or Mac. You would find a piece of content that you wanted to view or play. You would click the URL, and some Java code would download to your machine; this would be the DRM controller. You would need to download it only once, and it would install automatically and then run. No muss, no fuss.

That was the theory, anyway. The problem was that the “write once, run anywhere” promise of Java was never quite delivered. There were version incompatibilities, performance problems (particularly on the Macintosh platform), and various issues relating to Sun’s lawsuit against Microsoft over the technology. Many thousands of Java applets do exist all over the Web, but no one has been able to successfully implement a DRM controller in Java that has acceptable performance and wide enough platform compatibility. Sun took steps toward building its own DRM controller as part of an e-commerce component called the Java Wallet, but the product was never completed.

In Chapter 7, we outline Microsoft’s approach to rendering application and DRM controller architecture. Briefly, Microsoft solves the problem by separating the DRM controller into its own application and developing a series of rendering applications designed to be compatible with it. Unless someone figures out how to do something similar in Java, this may be the most effective approach to integrating DRM functionality with rendering applications.

Identification

A user has to establish an identity in order to be cleared to exercise a right on content. Authentication of identities is one of the classic problems in the security field. The first thing to worry about is what you’re identifying – the user or the device? This is not a trivial matter. A user may want to view or play content on multiple devices that he owns; for example, he may want to read an eBook on both a PC and a PDA. He may think that he shouldn’t need to pay for separate versions of the eBook for those platforms. Analogously, if a user buys a music CD and wants to listen to it on the cassette player in his car, he feels free to make a cassette copy of the music (and, indeed, is legally in the clear if he does so).

Conversely, there may be a device that more than one person uses regularly, such as a shared PC in an office or the family Internet music player. Should each person who uses the device pay separately? Before the Internet, content providers never seriously considered these questions, because of the limited capability to copy physical media and the limited numbers of people who can pragmatically use a single device. But DRM vendors are now grappling with this issue.

User identification

A user’s identity can be

- ◆ A piece of information that you supply, such as a name, an e-mail address, a phone number, a Social Security number, an account number, a user ID, or a password
- ◆ A piece of information inherent to you, such as a biometric (fingerprint, retina scan, and so on)
- ◆ A set of information that a trusted third party holds about you, such as a digital certificate

Each type of identity has pros and cons. The first of type is the most common: The system prompts you for one of these pieces of information. All are things that others can find out about you. Passwords are somewhat different because they supposedly aren’t used for any purpose other than to get access to a certain system, and they are kept in some place that has greater or lesser degrees of inaccessibility, ranging from highly encrypted password files to cookies on your hard drive to yellow sticky notes on your PC monitor. In general, though, these pieces of information are easily transferable from one person to another, which means that their value for authentication purposes is only approximate.

User identity information is often stored on the machine. The most common way to do this in the world of Web browsers is a *cookie*, which is a text string within a file or a file unto itself. In

addition to containing information about your identity, a cookie can contain things about what pages on a site you visited, the status of your shopping cart, and so on. Cookies can also contain usernames and passwords. One problem with cookies is that they are tied to machines, not people. Someone can walk up to your machine and assume your identity by going to Web sites that use cookies that are stored on your machine.

Unique pieces of information about you, such as biometrics, are the surest method of personal identification. However, they are expensive to implement — no one wants to have a retina scanner or palm reader next to his PC — which is why their use in consumer content distribution is rare, although at least one DRM vendor, Musicrypt (see Chapter 11), uses them.

Digital certificates, the third type of personal identification, are supplied and maintained by third parties called certificate authorities (CAs). You apply for a digital certificate through a CA, which collects various pieces of information about you. The CA, to uphold its reputation as trustworthy, takes steps to verify that you are who you say you are.

There are several commercial CAs, such as Verisign and RSA Security. Verisign issues certificates for businesses that want to do e-commerce on the Web using the standard Secure Socket Layer (SSL) protocol. To apply for an SSL certificate, you have to supply information such as your business's taxpayer ID, DUNS (Dun & Bradstreet) number, and so on — all information that can be verified. Microsoft has a sort of lightweight certificate authority called Passport, which we discuss in more detail in Chapter 7. Some institutions, such as large companies and universities, maintain their own CAs.

CAs store digital certificates in encrypted, secure formats in certificate repositories. They are integrated into certain DRM systems so that when the DRM controller wants to verify a user's identity, it requests a digital certificate from a CA.

Device identification

Authenticating the identities of devices, rather than people, is another matter. The simplest way to identify a device is by putting a unique serial number in it that software can read. Intel tried doing this with some of its microprocessors, only to be met with howls of protest from privacy advocates. Nevertheless, several DRM vendors have figured out ways to extract ID information from PCs that is unique or nearly so. One way is to read serial numbers that are stored on hard disk drives. Disk manufacturers put software-readable serial numbers on their drives starting long before anyone thought about Internet privacy.

Another technique is to authenticate devices based on their IP addresses, which are the basic numerical addresses of every device attached to the Internet (an IP address is a group of four numbers from 0 to 255). This is sometimes used to authenticate users of content in an institutional setting, such as a corporate site license to an expensive market research service. However, as networking technology gets more and more sophisticated, this technique becomes less feasible. For example:

- ◆ It used to be that all devices had IP addresses that were permanently assigned. Nowadays, some network management techniques assign IP addresses dynamically, meaning that a given device might have a different IP address every now and then.
- ◆ It also used to be that all IP addresses were globally unique. There are some increasingly popular networking technologies that assign IP addresses which are only unique within a subnet (a single office, for example). Modern SOHO (small office/home office)

networking equipment often works this way. If IP addresses are not universally unique, it's not possible to identify devices by their IP addresses.

Some DRM vendors, like Microsoft, have tried combining personal and machine IDs into schemes that approximate the situations mentioned at the beginning of this subsection, such as one person owning multiple devices. However, none of the schemes we have seen so far are entirely satisfactory. As we said in this book's Introduction, the original intent of DRM was to replicate aspects of physical media distribution paradigms in the online digital world; identity authentication turns out to be one of those aspects that are hard to emulate correctly.

Service providers

A number of vendors have established themselves as DRM service providers, also called *clearinghouses*. These take much of the complexity of DRM implementation away from publishers by outsourcing it.

Service providers can perform several of the functions shown in the Content Server and License Server boxes in Figure 5-1. Often they perform license server functions, including storage of identities, encryption keys and content rights information, and generation of licenses. Sometimes they also host the actual packaged content and distribute it themselves. And some service providers even outsource the packaging function; just send them your content and metadata, and they take care of everything else – and send you a check and an activity report at the end of each month.

The function of a service provider is to ease the DRM implementation burden for content providers. Some vendors, such as Reciprocal (see Chapter 7) and Digital World Services (see Chapter 11), do nothing but outsource DRM solutions that come from multiple DRM technology vendors. A larger number of vendors outsource DRM solutions based on a single system from a vendor that built its technology specifically with service providers in mind. These service providers sometimes act as CAs, as described earlier, although they usually work with an established commercial CA rather than build their own. Other DRM vendors let content providers choose between licensing their software for implementation in-house and having the vendor outsource the implementation.

Streaming media

We conclude the discussion of the DRM reference architecture with a few thoughts on how DRM works for streaming media formats, such as RealAudio, RealVideo, Windows Media, or QuickTime 4.

Streaming formats differ from standard formats in that they aren't delivered to the user as entire files, as would be the case for text and (most) still image formats, as well as certain audio and video formats such as WAV and MP3. Instead, a server delivers streaming media to the user on demand (that is, when the user initiates playback), a bit at a time, in increments called *packets*.

There is nothing about the DRM reference architecture that doesn't apply to streaming media. Instead of moving the file from the content repository to the packager and then to the client in discrete steps, it happens a packet at a time — including encryption, if used. On the client side, packets can be decrypted and then rendered one at a time as well.

Certain DRM vendors have begun to come out with solutions for streaming formats. DRM solutions that measure data about viewership are useful in the world of streaming media, but those that mainly exist to protect against piracy are of marginal relevance because there is little

reason at this time to worry about the piracy of streaming media. Capturing a video or audio stream in order to make unauthorized copies of it is considerably more trouble than obtaining a file. You would have to set up a bogus player application that captures the stream as it is delivered, packet-by-packet, from the server, and saves the data in a file — often a *very large* file. Furthermore, streaming media often reaches users in poor quality because of missed packets, dropped connections, and so on.

In the future, when broadband delivery of streaming media is not only much more ubiquitous but also much more reliable, the piracy of streaming media will be of concern to content owners. But for now, the streaming media infrastructure is just too unreliable to worry much about piracy. Nevertheless, major record labels and movie studios are taking piracy seriously and embracing DRM solutions for streaming media from vendors like Microsoft, RealNetworks, and InterTrust.

In Chapters 7, 11, and 12, we look at examples of vendors' DRM solutions and see how they match up with the DRM reference architecture.

Encryption

Not all the systems that we classify as DRM use encryption, but it is the most common means of copy protection, and it is the core technology most closely associated with DRM. Much of the constant discussion about the efficacy of DRM systems for copy protection focuses on the power of encryption, even though most instances of DRM systems being cracked have not involved breaking the actual encryption but instead have involved some other security hole. Although everyone has heard of encryption, few people really understand it. Therefore, it's worthwhile to delve into the mysteries of this field.

The field of cryptography is sort of like a black hole. If you took physics, you may recall that black holes are objects of virtually infinite gravity. If you stay far enough away from one, you can observe it but not get much detail about it, but if you get too close, you cross an event horizon and then get sucked into its gravitational field — from which there is no escape. Analogously, black hole subjects, such as cryptography, are those about which you can either learn a superficial amount or get sucked inexorably into the gory details.

In the case of cryptography, the gory details concern things like encryption algorithms and government cryptography policy. This section is an attempt to bring you across the event horizon just enough to give you the information about encryption that you need to evaluate digital rights management solutions — and then bring you back to the safety of normal gravity. If you are interested in crossing the event horizon, the best book on the subject — although slightly out of date by now — is Bruce Schneier's *Applied Cryptography*. A briefer but more up-to-date source of information is the Crypto FAQ on RSA Security's Web site, at www.rsa.com/rsalabs/faq/index.html. For those who would like to read a fascinating recent history of the subject, we highly recommend Steven Levy's book *Crypto*.

The overall goal of cryptography in the digital world is really not so much to protect secrets as it is to emulate physical-world human behavior toward information. The problem with online digital information is that it's far easier for people to access, copy, analyze, and manipulate it than it is in physical analog form. Think of digital cryptography as a set of techniques that restrict access, analysis, and manipulation so that online information exchanges resemble their old-fashioned, real-world counterparts.

For example, one application of cryptography is ensuring that the contents of a message in transit aren't changed. In the physical world, you can take simple steps to make sure that a message doesn't get changed on its way from the sender to the receiver, such as sealing it in an envelope. It is possible to alter the contents of a hard copy message in such a way that the receiver doesn't realize that it has been tampered with, but not easy. In the networked digital world, in contrast, it can be easy to alter a message in transit — or alter the identity of the sender. Cryptography steps in to re-establish the difficulty of that which is easy in the networked digital domain.

The primary goal of cryptography in DRM is to prevent content from being accessible in its native format at all times except when the DRM controller permits it. As mentioned earlier, it is also the responsibility of the rendering application to allow users to do only what the DRM controller permits.

The question most likely to go through the minds of people who evaluate DRM technologies is, "How good is the encryption?" For better or worse, the answer is usually, "Good enough." There is no truly reliable way of measuring the efficacy of encryption systems. However, we can give you a bit of background and some parameters to look at, in order to justify and help you understand this admittedly weak answer.

The strength of encryption algorithms

Cryptography has been around for thousands of years. Encryption algorithms — procedures for doing encryption — have come a very long way since the early days, when they generally consisted of simple things like substitution ciphers. (For example, take a message and substitute H for A, Y for B, E for C, and so on.) Nowadays, encryption algorithms have their bases in advanced mathematical concepts.

Some algorithms are stronger — more impervious to cracking — than others, but one absolute truth remains: There is *absolutely no such thing* as an encryption algorithm that is both provably unbreakable and practical to implement. There is a provably unbreakable algorithm known as a *one-time pad*, but it has no practical implementation on the Internet. In a one-time pad, you encrypt a message with a key that is completely random, has no relationship whatsoever with the keys you generated for previous messages, and is only used once. Furthermore, the key is at least as long as the message it encrypts. The recipient needs the key to decrypt the message. The problem is how to get the key to the recipient without compromising security. In the real world, this was done by sending agents out into the field with keys written down on pads (hence the name), which they would use later when they received messages. But you can't do something similar on the Internet.

In general, there are trade-offs between an algorithm's strength and its implementation practicality: Stronger algorithms take longer to encrypt and decrypt content. You want an algorithm that is strong, but you don't want your users waiting a long time while the DRM system's client decrypts the content. At the same time, because computers get faster and faster, that which takes a long time now may not after a few years.

In other words, you can think of cryptosystems as analogous to antitheft devices for your car (with the difference that the car thieves get faster every year). When you purchase a device such as an ignition cutoff switch or a steering wheel lock (like the Club), you're *detering* theft, not preventing it on any absolute basis. And you choose which antitheft device to purchase based, in part, on how annoying it is to use.

The strength of a cryptosystem derives from several factors, including

- ♦ The length of time it would take for a cracker to break it using a *brute-force attack*
- ♦ The algorithm's susceptibility to various clever forms of *cryptanalysis*, which can make brute-force attacks unnecessary

As a very simple example, consider the combination locks commonly used on luggage. They have three dials with settings from 0 to 9 each. A brute-force attack is one that starts at 000 and works its way up to 999 until the briefcase opens. One form of cryptanalysis for a suitcase lock is to find pieces of information about the suitcase's owner that she may use as a combination, such as her birthday, her age, or the last three digits of her phone number. (Another may be that most people are too lazy to reset the combination to something other than the usual factory default of 000.)

Of course, by most measures, three-digit combination locks on suitcases are not very effective. An experienced burglar can open one in well under an hour, and it would be easy to do so without being detected after the fact. The suitcase owner may find this acceptable. But if this were digital cryptography, she would have to take into account that an algorithm that takes an hour to break today may take a few minutes in a couple of years. This would be tantamount to the three-digit combination magically shrinking to two digits.

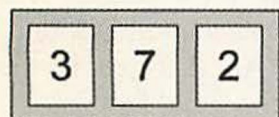
As another (and more germane) example, consider the security of your e-mail account. Whenever you log in to read your e-mail, you enter a username and password. Most e-mail servers store passwords in a file in encrypted form. Therefore, a would-be cracker has two avenues: an attack on the password itself or an attack on the encryption of the password file.

As in the suitcase example, the cracker can try guessing the password by brute force. The length of time it would take a cracker to guess your password through brute force is a function of the number of possible characters in the password and the universe of characters from which users can choose (such as uppercase letters, lowercase letters, numbers, and punctuation). For passwords of up to ten characters that allow upper- and lowercase letters and numerals, there are 839,299,365,868,340,224 possibilities. That makes brute force pretty unlikely. A better possibility may be to guess a particular user's password through knowledge of the user. Many people use personal information such as names of family members, nicknames, and makes of their cars for their passwords. If a cracker were to try breaking the encryption of the password file itself, it becomes an issue of how secure the password file is.

Key length

In cryptography, the most basic measure of an algorithm's strength is the key length. A key is the secret number necessary to decrypt the message; thus, the objective of a cracker is to find the right key. In the preceding examples, the suitcase's combination and the password to your e-mail account are keys. The length of a key is an expression of the number of possible keys in a cryptosystem. Specifically, it's the maximum number of digits in the binary number that makes up the key. If the key can be up to N digits, the number of possible keys is 2 to the n^{th} power. So, for example, a key length of 20 gives you 2 to the 20th power, or 1,048,576 (about a million) possible keys. A key length of 30 gives you 2 to the 30th power, or 1,073,741,824 (roughly a billion) possible keys. In fact, a good rule of thumb is that each 10 of key length multiplies the number of keys by 1000, or adds three decimal digits or an "-illion" to the number of possible keys.

In the earlier two examples, the suitcase combination lock has 1000 possible keys, which is roughly equivalent to a binary key length of 10. The password string, in contrast, has a binary key length of about 60. In Figure 5-3, *Avg. time to brute force crack* assumes that each try takes one second, just for comparison purposes. As you can see, a ten-character alphanumeric password is nominally *much* more immune to brute-force attacks than a three-digit numeric suitcase lock. Even if you had a computer that could crunch a million keys per second, the e-mail password would still take more than 13,000 years to crack by brute force.



Suitcase lock

Decimal key length: 3
 Number of digits per character: 10
 Number of possible keys: 1000
 Binary key length: ~10
 Avg. time to brute force crack: 8.33 minutes

Password: *****

Email password

Alphanumeric key length: up to 10
 Number of digits per character: 62
 Number of possible keys: 839,299,365,868,340,224
 Binary key length: ~60
 Avg. time to brute force crack: 13,307,004,152 years

Figure 5-3: Different key lengths determine the strength of a cryptosystem.

Key lengths make a significant difference in the strength of a cryptosystem. For example, most Web browsers offer the SSL protocol to facilitate secure e-commerce transactions so that people can send their credit card numbers and other personal information over the Internet without worrying (too much) about placing that information into the wrong hands. In North America, Web browsers now use a key length of 128 bits, which is considered very high indeed — not likely to be brute-force crackable by the most powerful computers for many years to come.

It's so good, in fact, that the U.S. government considers the technology that supports 128-bit SSL encryption to be "weapons technology" — unexportable for national security reasons — so Web browsers outside North America must support weaker 40-bit encryption. 128-bit encryption is nominally 309,485,009,821,345,068,724,781,056 (2 to the 88th power) times harder than the 40-bit version to crack by brute force. There are rumors that the National Security Agency (NSA) has computers that can crack 40-bit encryption in a reasonable amount of time.

Factors other than key length

It's important to understand that key lengths are not the only criteria to use in judging encryption systems — just as you shouldn't judge the true power of a car just by its engine's displacement or number of cylinders. Nor are key lengths comparable across different algorithms: You can't say that Algorithm A is stronger than Algorithm B just because Algorithm A has double the key length of Algorithm B.

There are many other factors in considering encryption strength. One is the imperviousness of the cryptosystem to attacks that are more sophisticated than brute force — as are most serious attacks anyway. Some involve heuristics (educated guesses) of the type discussed earlier. Others involve exploiting weaknesses in the system that allow crackers to detect patterns. For example, many cryptosystems involve random number generators; random number generators sometimes exhibit patterns, so after a while it's possible to predict (or at least narrow down) the next number that a random number generator will generate. Code breakers at organizations such

as the NSA have been known to spend years finding the tiniest weaknesses in an encryption scheme in order to crack it.

Still other attacks on cryptosystems involve techniques that have nothing whatsoever to do with finding the key. The algorithm may be sound, but the system of which the algorithm is a part may have holes in it. As it has been said, you can put as many deadbolts on your front door as you like, but if your side window opens from the outside, it doesn't make any difference. This is the most common avenue for crackers.

It's even possible to go beyond the hardware or software of a cryptosystem to break it, through plain old human fallibility. In the early days of the Cold War, the progenitor of the NSA achieved an important breakthrough in cracking Soviet codes because the Soviets reused keys in their supposedly uncrackable one-time pad system. (They did this by discovering that the Soviets were reusing pads with encryption keys on them, either out of laziness or to conserve paper.) It may even be more expedient to bribe someone to divulge a key than to try to crack a cryptosystem at all.

Published versus proprietary cryptosystems

Many cryptography experts feel that the only truly valid way of judging the strength of a cryptosystem is public scrutiny. Only when a cryptosystem is out there in the real world, with real crackers banging on it for an extended period of time, does the public gain confidence in it. Neither pure mathematical analysis nor any expert testimonials, let alone vendors' claims, are good enough anymore.

Unfortunately, the need for public scrutiny clashes with three other factors: the past history of cryptography, the newness of the DRM market, and plain old paranoia.

Throughout cryptography history, it was considered necessary to keep secret the details of the encryption algorithm as well as the key itself. Older algorithms weren't that sophisticated, so if you knew how they worked, you were halfway to breaking them. Furthermore, the information that encryption protected was the stuff of life and death, of political systems, dynasties, and nationhood — slightly more important, you have to admit, than someone's journal article or hit single. In particular, cryptography was the virtually exclusive province of the government.

It wasn't until the 1960s that digital cryptography became common in private industry. Around that time, IBM researchers invented an algorithm that the NSA eventually co-opted as a government standard: the Data Encryption Standard, or DES. DES is quite powerful as an encryption algorithm, as long as the key length is sufficient. The details of DES were published within the United States after a long series of legal and political battles. It was necessary to do so because the private sector was not about to take the government's word on the power of the algorithm for granted: In particular, the private sector didn't trust that the NSA would let private industry keep secrets from *it*.

The same lack of trust has attended all major encryption algorithms since DES. The cryptographic community has a habit of hounding out of the marketplace any vendor that tries to keep the details of its algorithm secret. There are many examples of this:

- ♦ A startup company called Tri-Strata claimed to have a practical implementation of the one-time pad but refused to divulge the details, citing competition; subsequent forced scrutiny by leading cryptographers revealed that the solution wasn't really a one-time pad.

- ◆ Chapter 6 describes the RIAA's panicked refusal to let a researcher give a paper at a conference that explained how he broke the Secure Digital Music Initiative's proposed secure watermarking scheme. The crypto community and other factions eventually shamed the RIAA into backing off and letting the researcher deliver his paper.
- ◆ A group of crackers recently managed to break the proprietary encryption of Gemstar's eBook platform. They threatened to expose the weaknesses that they found in Gemstar's algorithm unless the vendor agreed to do certain things, which included publishing the details of the algorithm.

The only way for a cryptosystem to really take off in the marketplace is if its inventor is willing to invest the time and money — and adopt the attitude — necessary to subject its algorithm to the rigors of scrutiny by an industry full of cryptographic experts who live for opportunities to break yet another encryption scheme.

One published algorithm that has become prominent is the RSA algorithm, named for its inventors, Ronald Rivest, Adi Shamir, and Leonard Adleman, all of MIT. They implemented a specific form of a mathematical technique known as public-key encryption.

In traditional encryption schemes such as DES, two parties have to give each other their keys in order to read messages from each other. (For this reason, this type of encryption scheme is often called *symmetric-key* encryption.) This presents a problem because the two parties have to meet in advance to do this. Whitfield Diffie (now of Sun Microsystems), Martin Hellman of Stanford University, and Ralph Merkle of Berkeley achieved a breakthrough in which they figured out a way around this. (At around the same time, James Ellis, Clifford Cocks, and Malcolm Williamson made a similar discovery in the U.K., but it was done for British Intelligence and so was not made public.)

Here's how public-key encryption works: Each party has two keys, a public key and a private key. You can combine the two keys into one larger key; you do this in a way that makes it possible to recover one of the original keys if you know the other one, but practically impossible if you know neither. In mathematical terms, you need a way of combining the numerical values of the two keys into a third number from which it's virtually impossible to extract the original two numbers. (Think of this as pouring two piles of sand into a bigger pile and trying to determine the number of grains of sand in each of the two original piles — only harder.)

The mathematical name for this is a *one-way function*. Diffie, Hellman, and Merkle found a type of one-way function that has enough different possible inputs to make it worthwhile for cryptography: the multiplication of two large prime numbers. If you take two large primes and multiply them together, you get a larger number; but if you only know that larger number, it's very hard to reconstruct the two primes that generated it. However, if you know the larger number and one of the smaller ones, it's very easy to generate the other smaller number: Just divide one into the other.

In public-key encryption, if I want to send you a message, I ask you for your public key. You can send your public key to me without fear of eavesdropping, because your public key is useless without your private key. I can use my private key in conjunction with your public key to encrypt the message. When you get the encrypted message, you can use your private key to decode it.

The RSA implementation of public-key encryption has a variable key length; most commercial implementations now use a staggering 1024 bits (that's roughly equal to a 2 with 308 zeroes

after it), with some especially sensitive implementations using 2048 bits (or about 2 with 609 zeroes after it). How does this compare to the 128-bit encryption used in U.S. and Canadian SSL? Remember that comparing key lengths in different algorithms is like comparing apples to oranges. 1024-bit RSA is not 8 times stronger than 128-bit SSL. It's necessary to use key lengths that long in RSA because of the mathematics involved. No one would ever use brute force (that is, try all 2^{1024} keys until one works) with RSA because there are well-known techniques for reducing the size of the problem substantially — meaning that in the worst case, the number of attempts necessary to crack the algorithm would be much less than 2^{1024} .

RSA is used in many popular software products, including operating systems from Sun, Microsoft, Apple, and Novell, as well as major Web browsers, e-mail programs, and groupware packages. It is the closest thing there is to a de facto standard. So why doesn't every vendor use it in their encryption implementations? One reason is that it is inefficient to implement, especially with the longest key lengths. Another is that cryptographic research is always advancing, and there are always new algorithms that their inventors optimistically feel will improve upon RSA and other popular algorithms.

More importantly, although the details of the RSA algorithm are publicly available (at least in the United States), it's been covered under a patent owned by RSA Security, Inc., until the patent expired in September 2000. This meant that if you wanted to use the algorithm, you had to pay a license fee to that company or risk probable lawsuit for patent infringement. For a DRM solution vendor, using RSA encryption thus added to the cost of the technology.

Encryption in DRM solutions

So where does this leave you if you need to choose a DRM solution based on its encryption? Unfortunately, some of the encryption schemes used in DRM solutions are both new and proprietary. When you ask a vendor to vouch for the strength of such a cryptosystem, it will usually point out the impressive credentials of the designer or the number of patents it has pending. Our point of view is that if you're really worried about encryption strength, this isn't good enough.

Two things can happen with a vendor like this: If the vendor remains obscure, it's likely that the cryptography community will leave it alone. This may be okay if the vendor is committed to a niche market, but otherwise, the vendor could go out of business. On the other hand, if the vendor becomes big and popular, it becomes a target for cracking. If you're lucky, the cryptosystem is strong, but if not . . . Those who have responsibility for very large-scale deployments of encryption-based DRM technology — such as the heads of online initiatives within major record labels, or eBook product designers — have reason to worry.

DRM technology vendors have finally begun to realize this and are now basing their technologies on trusted public encryption algorithms. Symmetric-key algorithms that came out after DES are now popular for encrypting content: Examples include RC5 and RC6 from RSA Security; BlowFish and TwoFish from Counterpane Labs; and AES (Advanced Encryption Standard), the government's new successor to DES, which is based on a Belgian algorithm called Rijndael (pronounced *RAIN-doll* and meaning "Rhine Valley").

Public-key algorithms aren't used to encrypt content, because they are too inefficient. However, they are used for other purposes, such as generating digital signatures (see below) and encrypting the keys used for symmetric-key algorithms, thereby adding an extra layer of security.

Recent history is rife with tales of cryptosystems designed to protect content that have been broken. This happens despite the strength of the algorithm used because crackers can often find security holes in a system that do not depend on the actual cryptographic algorithm. Although the recently passed Digital Millennium Copyright Act (DMCA; see Chapter 3) provides a legal shield to publishers for certain types of code breaking, you should consider it likely that any encryption scheme popular enough to be in widespread use is also a prime target for crackers. In fact, you should *assume* that a new, proprietary cryptosystem will eventually be cracked. You shouldn't assume otherwise for years. While this book was being written, the security for Adobe and Microsoft eBook DRM systems were both cracked.

Does this mean that encryption is useless? No. If someone cracks an encryption scheme, it doesn't necessarily imply that everyone who receives content in that encrypted format is suddenly going to get it for free. Analogously, just because someone figures out how to pick Master brand combination locks doesn't mean that every high school student's locker is now unlocked. As a more germane example, consider DeCSS, a piece of software written for the Linux operating system that decrypts DVDs. DeCSS was written so that Linux users would have a way of playing DVDs on their machines because the only commercial player software available was for the PC and Macintosh platforms. DeCSS allows decrypted digital files of DVD content to be created; such files can then be used to create pirated DVDs with perfect quality.

DeCSS is the subject of a lawsuit by the major movie studios, who hope to stifle it via DMCA. (At this writing, the case is being heard in a federal appeals court.) The descrambling algorithm within DeCSS has been ported to the PC and Macintosh, but very few people have it. No computers have been distributed with DeCSS preinstalled. No major software vendor is marketing it. In other words, the average computer user is highly unlikely to have DeCSS on his or her machine.

As with many such issues, this ends up being a business decision. How much risk of "bleed" (loss of revenue due to piracy) are you willing to live with? Who are your enemies, and to what lengths are they willing to or can they go to persecute you? And by the same token, as we say several times in this book, who are your friends (customers), and how much inconvenience do you think they are willing to endure?

We conclude this section by noting that encryption in DRM solutions isn't confined to copy-protecting content, although that is certainly the most important use. Some other uses are to verify the contents of a file and the identity of its creator or rightsholder.

Verifying content: digests

Encryption-related technology can be used to ensure that the contents of a file aren't tampered with, just like the example at the beginning of this section with the written message and the envelope. An algorithm that involves a secret key can be used to create a *digest* for a file, which is a unique number that represents the contents of the file in shorthand form.

Say that the digest (sometimes also called a *hash value*) is 128 bits long, meaning that it can be represented in 16 characters appended to the end of a file. (Each character contains 8 bits.) This means that there are 2 to the 128th power possible digests — a number roughly equal to a 3 followed by 38 zeroes. In other words, it is possible that two different files can have the same digest, but *highly* unlikely.

Figure 5-4 shows how this works. In this case, two text files are identical except for one word. Despite this, the digest algorithm generates completely different digests for the two files.

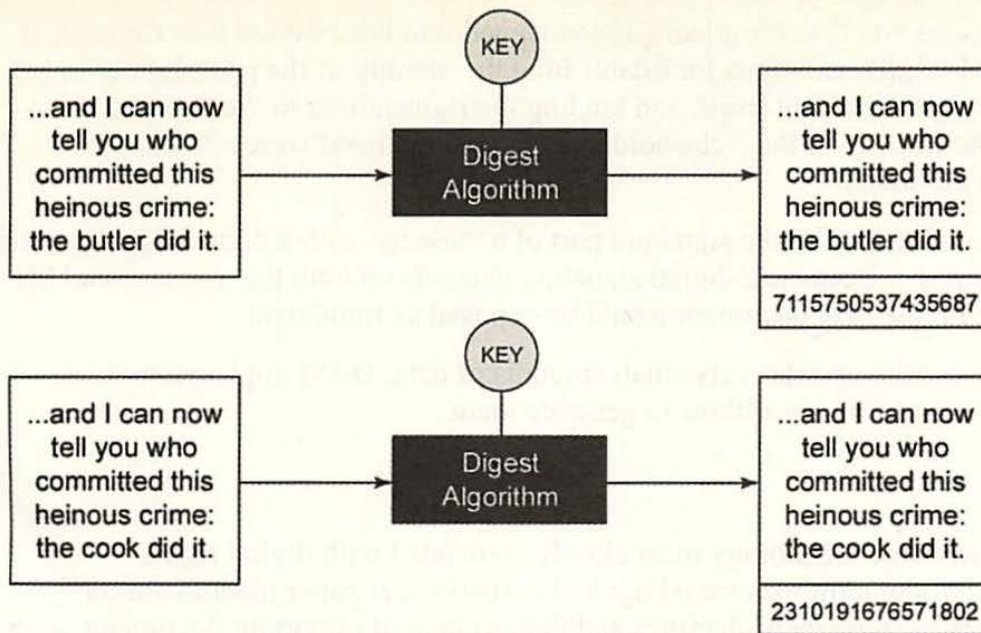


Figure 5-4: Generating unique digest numbers from files.

A rendering application for this type of file can compute the digest before rendering it. If the digest that it computes doesn't match the digest in the file, it means that the file was changed, and the application can refuse to render it. This scheme makes the file tamper-resistant, even if it isn't copy protected. Only those who can crack the digest generation algorithm can change the content without being detected.

This type of tamper resistance scheme is one way to enforce edit rights.

CROSS REFERENCE: Edit rights are discussed in Chapter 4.

Verifying identities: digital certificates

As mentioned earlier in this chapter, certificates are important ways of establishing the identities of both users and organizations. CAs, the organizations that create, store, and manage digital certificates, have businesses that fundamentally depend on their being trustworthy — that is, doing everything possible to guarantee both the validity of the data inside the digital certificates and that they are protected from tampering, theft, and privacy violations. A CA will lose its franchise the minute any of these things happen:

- ◆ It allows a user to create a certificate under false circumstances, such as a fraudulent business.
- ◆ It allows the data inside a certificate to be tampered with.
- ◆ It makes the information inside the certificates that it manages available to third parties or otherwise allows it to be used for any purpose other than the exact ones for which it was intended.

Digital certificates use encryption to accomplish all these goals.

Verifying content and identities: digital signatures

An important variation on the themes of both digests and certificates is the *digital signature*. A digital signature ensures both the contents of a message and the identity of the person who signs it. Digital signatures are now legally acceptable replacements for handwritten signatures, thanks

to the Electronic Signatures Act that President Clinton signed into law in 2000 (see Chapter 3). They are useful in DRM implementations for establishing the identity of the rightsholder of a piece of content, verifying the content itself, and binding the rightsholder to the content. In other words, if *either* the identity of the rightsholder *or* the content itself were altered, the digital signature would not match.

Furthermore, it's not possible to take the signature part of a message with a digital signature and graft it onto another message: Because a digital signature depends on both the message and the signature, the signature on the new document would be exposed as fraudulent.

Because digital signatures take up relatively small amounts of data, DRM implementations usually use public-key encryption algorithms to generate them.

Watermarking

Watermarking is the other core technology most closely associated with digital rights management. Historically, the term *watermark* applied to marks that paper manufacturers embedded in their papers to show their identities and that printers of important documents such as paper money and stock certificates used to prove authenticity.

Generically speaking, watermarks were meant to convey some information about a document in a way that

- ◆ Didn't interfere with the appearance or readability of the document
- ◆ Was inextricably bound together with the document

Fine-quality stationery, of the type you may use to print your resume, has watermarks in it.

If you fast-forward to the digital age, you find that watermarking — called *information hiding*, *data embedding*, or *steganography* by specialists in the field — is used for a variety of purposes with audio, image, and video content. Watermarking is a separate field from encryption, although there is some overlap: For example, some watermarking schemes incorporate encryption, as we discuss later in this chapter in the section “Encrypted watermarks.”

The body of literature on watermarking is much smaller than that of encryption. Encryption is not only the stuff of recondite academic treatises; it's also fodder for popular writings that have to do with subjects such as spying, security, and privacy. Watermarking hasn't captured the broader public's imagination thus far. The only book-length writing on the subject at this point is a collection of technical papers called *Information Hiding Techniques for Steganography and Digital Watermarking*, edited by Stefan Katzenbeisser and Fabien Petitcolas, researchers at the University of Vienna and Microsoft, respectively.

As with paper watermarks, digital watermarks are used to associate metadata with content in a (mostly) permanent way without having to format the content in a special package and use encryption, as discussed earlier in this chapter. Put simply, a watermark is a way to make metadata become *part* of the content, instead of sitting alongside it.

How you do this depends, of course, on the medium of the content. In an image, watermarking software selects a region of the image and varies the colors and light intensities in a certain way. Video watermarking involves similar techniques. In an audio clip, watermarking software inserts ultrasonic information. Images, video, and sound are all “continuous” media with a resolution depth that can be deeper than humans can perceive; watermarking takes advantage of the medium's ability to go “under the radar” of human perception. For this reason, it's not really

possible to watermark a text file: Humans perceive each character of text discretely, so there's nowhere to put information that isn't detectable. (However, a related and very old technique of text steganography is to hide one message in another — for example, by using every third or fifth word for the hidden message.)

Watermark characteristics

Digital watermarks have the following characteristics:

- ◆ **Undetectability:** The watermark doesn't detract from the visual or audible experience of the content.
- ◆ **Robustness:** The watermark survives copying to lower-resolution formats or from digital to analog formats.
- ◆ **Capacity:** The watermark should be able to contain as much data as possible.
- ◆ **Security:** The watermark resists attempts to erase or alter it.
- ◆ **Efficiency:** The length of time it takes to insert and extract the watermark isn't excessive.

As you might imagine, there are trade-offs among these criteria.

Making watermarks undetectable by humans involves finding places in the content where they can go unnoticed. The key to creating a watermark is finding some part of the content where there's a lot of variation in sound frequencies, colors, light intensity, or whatever the medium conveys. Think of it this way: It's much easier to conceal an object in a clump of plants and bushes than in the clear blue sky; similarly, it's easier to conceal an ultrasonic sound in a complex piece of music than in a one-kilohertz sine wave.

However, the watermark must also be robust enough to survive copying and digital-to-analog conversion. If you make a copy of a photo in some low-resolution format (for example, if you copy a TIFF image, of the type that graphics professionals use, to the GIF format common on Web sites), the watermark may be lost. Making a watermark that survives copying requires a maximum amount of space on the image or in the audio or video clip. Thus there is a direct trade-off between robustness and undetectability.

Another dimension to the trade-off involves capacity — the amount of useful information that a watermark can contain. Obviously, more space is needed to contain more useful information. Various PhD dissertations have been written on how to shoehorn the maximum amount of data into a piece of content while preserving undetectability: Techniques involve the detailed analysis of the psychology of how humans perceive images or audio. Some watermarking schemes actually measure the amount of data that they can include in a particular file and only include up to that much.

If watermarks are used to embed copyright information in files, it's important that they be secure in addition to robust. The information contained in watermarks is often encrypted in such a way as to be tamperproof (as discussed earlier). They can also be created in a way that makes it impossible to remove the watermark without visibly or audibly marring the file.

Finally, it's especially important for watermark extraction to be efficient if watermarks are being used for DRM. You'll see shortly how this works.

Watermarks and DRM

Some people get confused about how watermarks relate to encryption and other elements of digital rights management — because vendors are out there making various claims for the two technologies that need to be put into perspective.

Here are some principles to bear in mind:

- ◆ Encryption requires a special application to decrypt and render content; watermarking does not.
- ◆ Watermarking binds metadata (semi)permanently with content; encryption does not.

Because watermarking does not require a special application to render content, it is fundamentally not suited to the task of copy protection by itself.

Figure 5-5 shows the difference between encryption and watermarking with a music example, although it would apply equally to image or video. With encryption (A), you need to decrypt the content (with a DRM controller, as explained in the section “A DRM Reference Architecture”) before it can be played. Furthermore, the metadata exists separately from the content; after you decrypt the content, the metadata is gone. Contrast this with watermarking (B): After you insert the watermark, the metadata is inextricably bound together with the content. The watermark extractor can pull out the metadata and use it separately, but it’s still there, even when the music is played. (Look closely at the figure, and you’ll see it.)

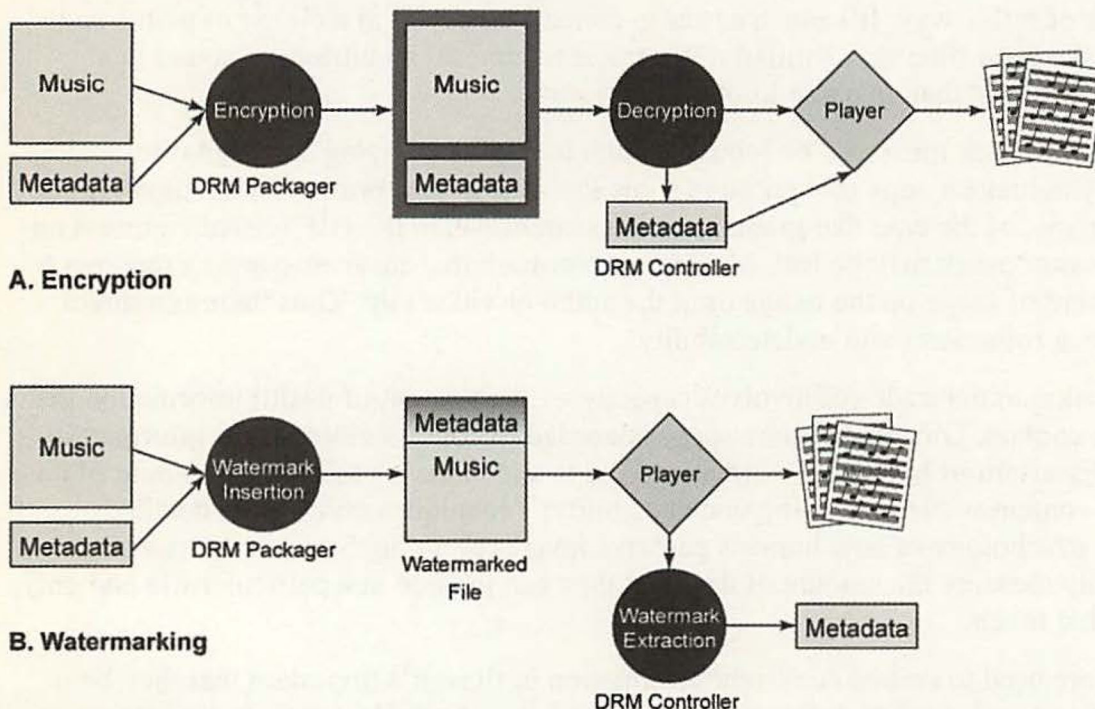


Figure 5-5: Encryption versus watermarking. Encryption requires a special application to decrypt the content for rendering, whereas watermarking requires a special application to extract metadata from the content.

Visible source identification

The most basic use of watermarks in DRM is akin to the original definition of watermarking: The watermark simply — and often visually or audibly — identifies the source of the image. For example, various stock image sites on the Web (such as PhotoDisc) display images for browsing purposes; these have logos plainly visible on them to prevent people from copying

them from their Web browsers and using them unscrupulously. Clearly these watermarks are meant to disrupt images, and they would be difficult to remove without damaging the image. Some don't consider this scenario true watermarking because the marks are not hidden.

Invisible source identification

The second use of watermarks is to establish the ownership of the file but keep the watermark undetectable. The idea is to leave information about the file's ownership, and any other relevant metadata, embedded permanently and invisibly (or inaudibly) as an "audit trail" so that pirated copies can be discovered and, presumably, copyright infringement claimed.

This is the classic application of watermarking in DRM. It is clearly an improvement on things such as plain-text copyright notices, which can be removed. However, it's generally not possible to "discover" pirated copies, especially if they have been printed to hard copy or are sitting on random hard drives. Digimarc, the leading vendor of image watermarking technology, created a program called MarcSpider that purports to search the *entire Internet* for watermarked images and report usage to the owner. This is not an effective way of tracking usage. This second use of watermarks has largely been abandoned.

A more appropriate use of putting ownership information in content watermarks is usage tracking. An application on the client side can read identification and ownership metadata from a watermark and report it back to a server that collects such data. In Chapter 7, we discuss the use of watermarking in broadcast monitoring, which is an example of usage tracking in radio and television. Using watermarks to track content usage is also becoming popular for digital content distributed on the Internet; for example, Minneapolis-based Cognicity, Inc. (www.cognicity.com) makes technology that does this for digital music.

Metadata conveyance

A third use of watermarks is one in which the watermark is visible or audible but does not disturb the image or sound quality and is just there as a convenient way of conveying metadata. Think of a bar code on a jar of jam in the supermarket: The bar code is plainly visible, but it doesn't detract from the jam company's logo, nor from the jam itself. However, it does allow the cashier to scan the price at checkout time.

Xerox' Glyph technology, in which tiny dots sprayed on paper convey lots of meaning in very little space, fits this category. Similar technology is used by the U.S. Postal Service for a handy way of encoding information on bulk mailings.

Encrypted watermarks

Nowadays, watermarking often works in conjunction with encryption as part of end-to-end secure content distribution solutions. In this case, watermarking serves as both a metadata vehicle and as a barrier to rendering.

There are two ways of combining watermarking with encryption. The first is to encrypt the watermark, but not the file. There are two reasons for doing this. The first is to make the watermark tamperproof, as described earlier in this chapter. Watermarks used in usage-tracking schemes, as previously described, contain data about content ownership that must be protected against alteration. A variation on this theme is tamperproof watermarks that contain timestamps, which indicate when the document was created. It's desirable in certain cases to be able to definitively establish the creation time of a document. For example, if two parties created similar intellectual property and are disputing claim to a patent, the winner will be the one who

can say that he or she created the intellectual property first. Surety, Inc.'s Digital Notary technology (www.surety.com) does this.

The other reason for encrypting watermarks is to keep the metadata in the watermark secure, even if the content is not. Digimarc's scheme for discovering pirated files depends on this.

Encrypted files with watermarks

The other way of combining watermarks with encryption is to watermark a file and then encrypt it, as shown in Figure 5-6. This is the general type of approach taken in Phase II of SDMI (see Chapter 6) and used in products by vendors such as Liquid Audio. Assuming that the encryption is effective, this is really the ultimate in secure content distribution because it not only protects copy but also binds content with metadata — which can include rights metadata.

The result is the best of both worlds. Even if someone were to crack the encryption, the metadata would remain with the content — presumably in a tamperproof form — allowing rights and ownership information to be permanently trackable. This allows the DRM controller and the rendering application to be tightly integrated: The controller can decrypt and feed to the player only as much content as necessary to decode the watermark and get the metadata. Then the player can extract the metadata from the watermark and pass it back to the DRM controller for processing — for authentication, payment processing, usage metering, and so on. If the DRM controller doesn't get the feedback that it wants from the player (that is, the player isn't sending back data from the watermark), the controller can suspect a bogus player and shut it off. Thus, would-be crackers have to decode watermarks in addition to breaking the encryption.

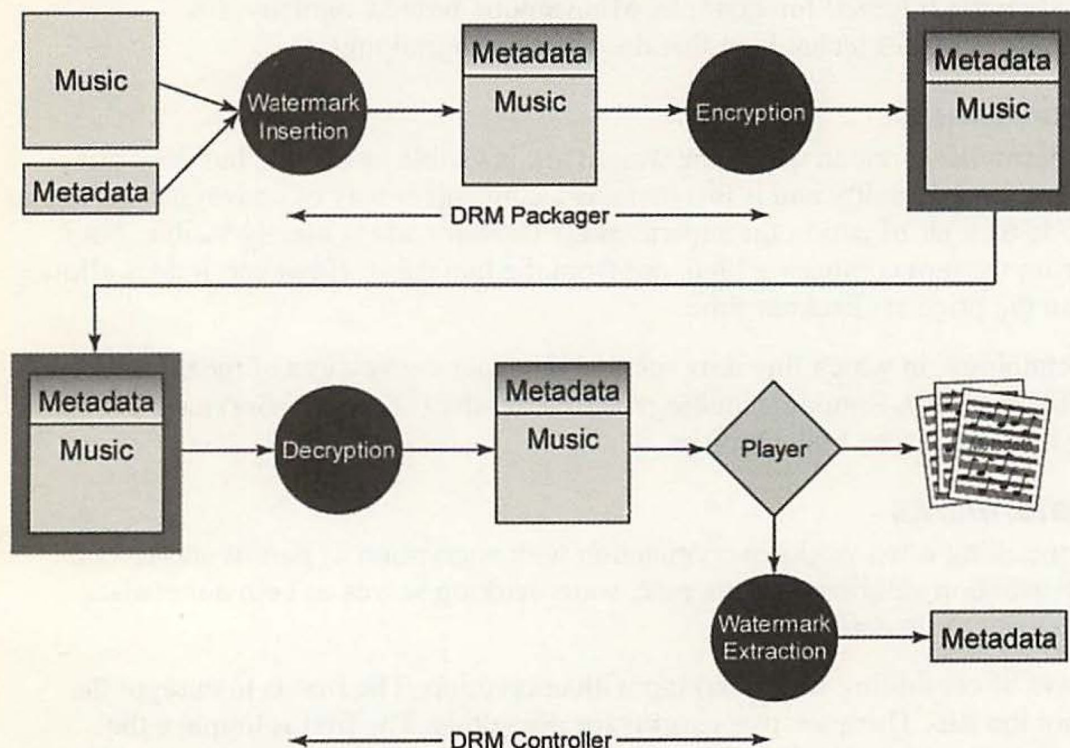


Figure 5-6: Watermarking combined with encryption.

A scheme that incorporates both encryption and watermarking is not foolproof, but (all else being equal) it's the best DRM scheme available. Watermarking applies only to image, audio, and video data types — not to text. But it's likely that the most successful DRM solutions in the years to come will combine encryption and watermarking in this way.