

# **The Electronic Book Exchange System (EBX) Version 0.8**

The complete technical specifications for the Electronic Book Exchange (EBX) system for interoperable applications and devices that use public-key cryptography for copyright protection and distribution of electronic books.

The EBX system is being developed by the EBX Working Group, whose members are Adobe Systems Incorporated, the American Library Association, Audible, ContentGuard, DigitalOwl.com, Glassbook, GlobalMentor, Nokia, RightsMarket.com, SoftLock.com, Thomson Consumer Electronics, Versaware, and Yankee Rights Management.

**July 2000 Draft**

Copyright © 2000 Book Industry Study Group, Inc. All rights reserved.

Editorial contact: Glassbook, Inc., 1601 Trapelo Rd., Waltham, MA 02451  
781-434-2000 [ebx-editor@glassbook.com](mailto:ebx-editor@glassbook.com)

<b>1</b>	<b>AN OVERVIEW OF THE EBX SYSTEM.....</b>	<b>6</b>
1.1	TERMS.....	6
1.2	ROLES .....	7
1.3	SYSTEM PRINCIPLES .....	8
1.3.1	<i>Consumer Needs.....</i>	8
1.3.2	<i>Bookseller and Distributor Needs.....</i>	10
1.3.3	<i>Publisher Needs.....</i>	10
1.3.4	<i>Author Needs.....</i>	11
1.3.5	<i>Library Needs.....</i>	11
1.4	ASSUMPTIONS .....	12
1.5	BASIC DESIGN .....	12
1.6	VOUCHERS .....	14
1.7	FUNCTIONAL MODEL .....	15
1.7.1	<i>Publishing.....</i>	15
1.7.2	<i>Distribution to Online Booksellers and/or Distributors .....</i>	16
1.7.3	<i>Delivery to Consumers.....</i>	17
1.7.4	<i>Transfer Between Consumers – “Give/Lend” .....</i>	18
1.7.5	<i>Transfer Between Libraries and Consumers.....</i>	19
<b>2</b>	<b>TRUST MODEL.....</b>	<b>21</b>
2.1	RELIANCE ON TRUST FOR NEGOTIATION OF DETAILS.....	21
2.2	RESULTS AND EFFECTS OF THE TRUST MODEL .....	21
2.3	NEED FOR PRESCRIPTION .....	22
2.4	TRUST MODEL OVERVIEW .....	22
2.4.1	<i>Behavior of Components in a Multivendor Environment.....</i>	22
2.4.2	<i>Future Versions.....</i>	22
2.4.3	<i>Evaluation and Rating of Products from Different Manufacturers .....</i>	23
2.4.4	<i>Trustedness and Trust Services.....</i>	24
2.4.5	<i>List of Trust Services.....</i>	25
2.4.6	<i>Factors Affecting the Value of Content.....</i>	26
2.5	CHARACTERIZING TRUST LEVELS .....	28
2.5.1	<i>Scope of Particular Failures.....</i>	28
2.5.2	<i>Skill, Tools and Expense Required to Cause Failures .....</i>	29
2.5.3	<i>Trust Levels and Certification.....</i>	30
2.5.4	<i>Perspectives on the Six Levels of Trust.....</i>	32
2.5.5	<i>Definitions of Trust Levels .....</i>	32
<b>3</b>	<b>FOUNDATION TRUST SERVICES .....</b>	<b>38</b>
3.1	PKI: FOUNDATION MECHANISM FOR AUTHENTICATION .....	38
3.1.1	<i>Requirements.....</i>	38
3.1.2	<i>Overall EBX Certificate Authority Architecture .....</i>	39
3.1.3	<i>Vendor Certificate Authorities .....</i>	41
3.2	COMPONENT TRUST LEVEL CERTIFICATION.....	43
3.2.1	<i>Certification Criteria.....</i>	44
3.2.2	<i>Certification Methods.....</i>	45
3.2.3	<i>Certification Entities .....</i>	46

3.2.4	<i>Reviews</i> .....	46
<b>4</b>	<b>TRANSFER PROTOCOL</b> .....	<b>47</b>
4.1	TRANSFER PROTOCOL AND TRANSPORT PROTOCOLS .....	48
4.2	DOMAINS OF TRUST .....	48
4.3	EXAMPLE OPERATION – E-BOOK PURCHASE.....	49
4.4	NOTATIONAL CONVENTIONS AND GENERIC GRAMMAR.....	50
4.5	HTTP IMPLEMENTATION NOTE: EBX HTTP REQUEST .....	50
4.6	HTTP IMPLEMENTATION NOTE: EBX HTTP HEADER EXTENSIONS .....	51
4.6.1	<i>EBX-Action Header</i> .....	51
4.6.2	<i>EBX-Version Header</i> .....	51
4.6.3	<i>EBX Challenge-Response Headers</i> .....	52
4.7	RECEIVING FULFILLMENT INSTRUCTIONS (HANDOFF).....	53
4.7.1	<i>Handoff Request to Voucher Server</i> .....	53
4.7.2	<i>Voucher Server Processing of Handoff Request</i> .....	53
4.7.3	<i>Handoff Response from Voucher Server (XML fulfillment instructions)</i> .....	54
4.7.4	<i>Client Processing of Handoff Response from Voucher Server (XML fulfillment instructions)</i> .....	56
4.8	GETTING VOUCHER(S) .....	56
4.8.1	<i>Voucher Request to Voucher Server (purchase, borrow)</i> .....	56
4.8.2	<i>Voucher Server Processing of Voucher Request (purchase, borrow)</i> .....	58
4.8.3	<i>Response from Voucher Server (purchase, borrow)</i> .....	59
4.8.4	<i>Client Processing of Response from Voucher Server for Voucher Request (purchase, borrow)</i> 60	
4.8.5	<i>Acknowledgment Request to Voucher Server (ACK)</i> .....	61
4.8.6	<i>Voucher Server Processing of Acknowledgment Request (ACK)</i> .....	62
4.8.7	<i>Acknowledgment Response from Voucher Server (ACK)</i> .....	62
4.8.8	<i>Client Processing of Acknowledgment Response from Voucher Server (ACK)</i> .....	63
4.9	GETTING ENCRYPTED E-BOOK(S).....	63
4.9.1	<i>Content Request to Content Server (purchase, borrow)</i> .....	63
4.9.2	<i>Content Response from Content Server (purchase, borrow)</i> .....	63
4.10	GIVING OR LENDING A BOOK (CONSUMER TO CONSUMER).....	64
4.10.1	<i>Transfer Request from Owner to Receiver (give, lend)</i> .....	64
4.10.2	<i>Transfer Response from Receiver to Owner (give, lend)</i> .....	65
4.10.3	<i>Owner Processing of Transfer Response from Receiver (give, lend)</i> .....	67
4.10.4	<i>Voucher Transmission Request from Owner to Receiver (give, lend)</i> .....	68
4.10.5	<i>Receiver Processing of Voucher Transmission Request (give, lend)</i> .....	69
4.10.6	<i>Voucher Transmission Response from Receiver to Owner (give, lend)</i> .....	69
4.10.7	<i>Owner Processing of Voucher Transmission Response from Receiver (give, lend)</i> 70	
4.10.8	<i>Content Transmission Request from Owner to Receiver (give, lend)</i> .....	70
4.10.9	<i>Content Transmission Response from Receiver to Owner (give, lend)</i> .....	71
4.11	ELECTRONIC MAIL TRANSFER.....	71
4.12	EBX ERROR HANDLING AND FLOW.....	71
4.12.1	<i>HTTP Implementation Note</i> .....	71
4.12.2	<i>Successful Requests to Voucher Servers</i> .....	71
4.12.3	<i>Failed Requests to Voucher Servers</i> .....	72

4.12.4	<i>Discussion of Voucher Fulfillment and ACK</i> .....	73
<b>5</b>	<b>VOUCHER ENGINE MODEL</b> .....	<b>75</b>
5.1	VOUCHER ENGINE PROCESSING .....	75
5.2	VOUCHER ENGINE RULES .....	75
5.3	VOUCHER ENGINE INTERFACES.....	76
5.3.1	<i>Create Voucher</i> .....	76
5.3.2	<i>Issue Voucher</i> .....	77
5.3.3	<i>Revoke Voucher</i> .....	77
5.3.4	<i>Import Voucher</i> .....	78
5.3.5	<i>Delete Voucher</i> .....	78
5.3.6	<i>Issue Nonce</i> .....	78
5.3.7	<i>Issue Credentials</i> .....	79
5.3.8	<i>Encrypt Content</i> .....	79
5.3.9	<i>Decrypt Content</i> .....	79
5.4	VOUCHER ENGINE PROCESSING OF EBX RIGHTS.....	80
5.4.1	<i>Writing EBX Rights in a Voucher</i> .....	80
5.4.2	<i>Enforcing EBX Rights in a Voucher</i> .....	81
<b>6</b>	<b>METADATA FORMAT</b> .....	<b>82</b>
6.1	METADATA ELEMENT .....	82
6.1.1	<i>Identifier Element</i> .....	82
6.1.2	<i>Format Element</i> .....	83
6.1.3	<i>Metadata Example</i> .....	83
<b>7</b>	<b>VOUCHER FORMAT</b> .....	<b>85</b>
7.1	RIGHTS SPECIFICATION .....	85
7.1.1	<i>Rights Overview</i> .....	85
7.1.2	<i>Licensee</i> .....	86
7.1.3	<i>Transfer Rights</i> .....	86
7.1.4	<i>Usage Rights</i> .....	87
7.1.5	<i>Authorization context</i> .....	87
7.1.6	<i>Consideration</i> .....	87
7.1.7	<i>Portion</i> .....	88
7.1.8	<i>Target</i> .....	88
7.1.9	<i>Scope</i> .....	89
7.1.10	<i>Consumer's knowledge of rights</i> .....	89
7.2	VOUCHER OBJECT.....	89
7.2.1	<i>EBX-Voucher – Voucher start-tag</i> .....	90
7.2.2	<i>ID – ISBN, DOI, or URN element</i> .....	90
7.2.3	<i>ContentKey – Content decryption key element</i> .....	91
7.2.4	<i>CopyCount – Count of authorized copies element</i> .....	91
7.2.5	<i>Rights – Basic permissions element</i> .....	92
7.2.6	<i>Lending – Lending Timeout and Status Element</i> .....	92
7.2.7	<i>PersonalUse – Personal use element</i> .....	93
7.2.8	<i>MAC – Message Authentication Code element</i> .....	93
<b>8</b>	<b>FORMAT OF OTHER OBJECTS</b> .....	<b>95</b>

8.1	CREDENTIALS OBJECT FORMAT .....	95
8.1.1	<i>Credentials – Credentials start-tag</i> .....	95
8.1.2	<i>AuthenticationScheme</i> .....	96
8.1.3	<i>Nonce</i> .....	96
8.1.4	<i>SignedData</i> .....	97
<b>9</b>	<b>APPENDIX A: CERTIFICATION POLICIES AND PROCEDURES .....</b>	<b>98</b>
<b>10</b>	<b>APPENDIX B: APPLICABLE LAWS.....</b>	<b>99</b>
10.1	U.S. EXPORT LAWS.....	99
10.2	(U.S.) DIGITAL MILLENNIUM COPYRIGHT ACT .....	100
<b>11</b>	<b>APPENDIX C - CONTENT FORMAT REQUIREMENTS.....</b>	<b>101</b>
<b>12</b>	<b>APPENDIX D - CONTENT FORMAT USAGE GUIDELINES .....</b>	<b>102</b>
12.1	OPEN EBOOK FORMAT .....	102
12.1.1	<i>Container File Format</i> .....	102
12.1.2	<i>Encryption</i> .....	103
12.1.3	<i>Metadata</i> .....	103
12.1.4	<i>Display Properties</i> .....	103
12.1.5	<i>Font Embedding</i> .....	103
12.1.6	<i>Open eBook Book Design Guidelines</i> .....	103
<b>13</b>	<b>APPENDIX E - REFERENCES .....</b>	<b>105</b>
<b>14</b>	<b>APPENDIX F - EDIT HISTORY .....</b>	<b>107</b>
14.1	VERSION 0.1.....	107
14.2	VERSION 0.2.....	107
14.3	VERSION 0.3.....	107
14.4	VERSION 0.4.....	107
14.5	VERSION 0.5.....	108
14.6	VERSION 0.6.....	108
14.7	VERSION 0.7.....	108
14.8	VERSION 0.8.....	108

## 1 An Overview of the EBX System

This document describes the Electronic Book Exchange (**EBX**) system. The EBX system defines the way in which electronic books (e-books) are distributed from publishers to booksellers and distributors, from booksellers to consumers, between consumers and between consumers and libraries. It describes the basic requirements of electronic book reading devices and the electronic books themselves.

It also describes how these “trusted” components interact to form a comprehensive copyright protection system that both protects the intellectual property of authors and publishers as well as describes the capabilities required by consumers. In addition, the model describes in general how products and revenue for those products are generated and managed.

The EBX system does not define a specific “content” file format (e.g., PostScript, PDF, HTML, XML). However, it does assume a minimum set of capabilities and features in the content and these are described in an appendix to this document.

### 1.1 Terms

The following terms are part of the EBX system.

- **E-book** – A digital object that is an electronic representation of a book. While an e-book can consist of a single page, it is normally thought of as an electronic analog of a multi-page hardcover or paperback book. An e-book is the so-called "content" in the EBX model. An e-book may come in a variety of formats, including, but not limited to, PDF, Open eBook Publication Structure 1.0, and various other textual and multimedia formats.
- **E-book Reading Device** – Typically a hand-held electronic device that is capable of displaying one or more e-books. Non-dedicated devices such as notebook and desktop PCs and PDAs can also be used as e-book reading devices.
- **E-book Reading System** – The combination of an e-book reading device with software or hardware that enforces copyrights and permissions. The system contains a voucher engine.
- **Voucher** – A digital object that describes an e-book’s transfer and usage permissions and copyrights. These are also called intellectual property rights. A voucher can be passed from one entity in the system to another entity. For example, a publisher can use a voucher to pass the permission to sell multiple copies of an e-book to a bookseller, or a consumer can use a voucher to pass the permission to use a copy of an e-book for a specified period of time (i.e., lend the e-book).
- **Voucher Engine** – The software or hardware that creates, imports, modifies, and reads vouchers. The voucher engine enforces copyrights and transfer and usage permissions.
- **Protocol Engine** – The software or hardware that generates, transmits, and receives network messages using the transfer protocol defined in this specification.

- **Voucher Server** – The computer system that is responsible for delivery of vouchers from one place to another. This system operates a voucher engine.
- **Content Server** – The computer system that is responsible for delivery of e-book content from one place to another.
- **Title** – A copyrighted work prepared by a publisher.
- **Copy** – An individual instance of a title, sold by a bookstore, lent by a library, etc.

## 1.2 Roles

The EBX system includes roles that interact with each other. A role is a function within the EBX system, not an entity (an individual or organization). An entity can assume more than one role, and a role can be distributed across more than one entity. An entity can also delegate a role to another entity.

The EBX system includes the following roles:

- **Publisher** – Creates e-books and their associated vouchers and sells them to booksellers and libraries (either directly or through distributors).

The publisher is the root source for initial encryption of e-books and creation of the associated vouchers for those books. The permissions for further modification, transfer, or sale of the vouchers are specified in the initial vouchers.

The publisher operates a voucher server to create e-books and vouchers and to transfer them to others.

- **Voucher Distributor** – Obtains vouchers from publishers and distributes them to booksellers, consumers, or libraries.

The voucher distributor operates a voucher server.

- **Content Distributor** – Obtains e-books from publishers and distributes them to booksellers, consumers, or libraries.

The content distributor operates a content server.

- **Bookseller** – Sells e-books to consumers or libraries.

This role represents where value (if any) is exchanged to acquire e-books and the vouchers for those e-books. The bookseller is the role that authorizes the consumer or library to purchase an e-book.

- **Library** – Purchases e-books from publishers, booksellers and distributors and lends them to consumers.

This role is similar to bookseller, except that value is not typically exchanged, and the e-books that the consumer obtains have vouchers that expire after a period of time (the loan period).

- **Consumer** – Purchases e-books from booksellers or borrows e-books from libraries.

The consumer's e-book reading system is activated when the consumer completes a purchase transaction with a bookseller or a loan transaction with a library. The e-book reading system negotiates with the voucher server and/or content server specified by the bookseller or library to transfer e-books and vouchers from the voucher and content servers to the e-book reading system.

A consumer may also give or lend an e-book to another consumer. The two consumers' e-book reading systems negotiate with each other to transfer e-books and vouchers from one e-book reading system to the other.

Following are some examples of assumption and delegation of roles by entities:

- A publisher can delegate its publisher role to a distributor.
- A bookseller can delegate its bookseller role to a distributor.
- A bookseller can assume the voucher distributor or content distributor role for the books it sells.
- A library can assume the voucher distributor or content distributor role for the books it lends.

### 1.3 System Principles

[**Editorial note.** This section has not yet been edited for Version 0.8. It might more appropriately appear in an introduction that is not part of the specification.]

The EBX system is designed around a number of basic principles that reflect the needs of the people who will use it. These principles are specific to the copyright and distribution requirements, not the content format.

#### 1.3.1 Consumer Needs

These needs represent the requirements of the people who matter most in an electronic book system: the people who actually read the books. The system must satisfy the consumer or else there is no market at all for electronic books. The paper book has undergone centuries of evolution, and consumers have grown accustomed to its current capabilities. A viable e-book system must at least provide an equivalent to paper book capabilities, and in most cases it must enhance those capabilities. To put it simply: to consumers, electronic books must be a clear *improvement* over paper books.

- **Interoperability** – Consumers need to be able to read any book, from any publisher on any device. They do not want to be in a position where the reading device or software that they own is incapable of reading a book that they desire.
- **Ease of Use** – This is an obvious consumer requirement, but it bears repeating. If the system is not extremely easy to use, consumers will simply continue to buy paper books.



Obtaining a book electronically must be easier than going to a bookstore to buy a paper book, easier than going to a library to borrow a book, easier than packing a book in a box to mail to a friend, easier than lugging a stack of books on a trip or to school and easier than filling a home with bookcases to create a family library.

- **Transparent Performance** – One of the compelling advantages of e-books is “instant gratification.” In other words, when a consumer wants a book, he/she can get the book immediately. Therefore, the system must perform its necessary tasks very quickly: downloading, re-encrypting, transferring, displaying, etc. In fact, all of these operations should be so quick that they become transparent.
- **Giving and Lending (First Sale)** – Consumers know that once they buy a copy of a paper book, within the limits of the rights of the copyright laws, the copy is their possession, and they can dispose of the copy as they wish. They can give their copy of the book to another consumer, they can lend their copy for a limited time to another consumer, they can give the copy to a public library for lending to other consumers, and they can sell a book to a used bookstore. A recent survey showed that about 60% of consumers in the U.S. borrow books from their public libraries. Consumers also expect that a copyright holder is paid for a book only the first time it is sold. In the U.S. this is known as the “First Sale” clause of the Copyright Act.
- **Copyright Law Help (Fair Use or Personal Use)** – Copyright laws are complicated and seemingly ever changing. While most consumers are law-abiding, they get frustrated by constantly being told what they can and cannot do with electronic media. The fine print in so-called “shrink-wrap” software license agreements and “FBI Warnings” on videotapes and disks can be difficult to understand and lessen the enjoyment of the entertainment. Electronic books may be able to avoid the fine print and warnings by making automatic what you can and cannot do with a copyrighted work.
- **Backup** – Consumers have experience with the unreliable nature of computer storage devices and will require a way to either make backup copies of their e-books or entrust a third-party *guardian* service to keep backups of their e-books.
- **Backward Compatibility** – Consumers want to buy e-books with the confidence that the e-book titles they buy today will be readable on the devices of tomorrow.
- **Privacy** – Consumers expect that their e-book purchases, borrowing, and subsequent transfer records will remain private and not be made available to third-parties. Some consumers even demand the ability to perform e-book transactions anonymously, as they do with paper book purchases today (e.g., with e-cash).
- **Added Value** – For consumers to make the leap to e-books, there has to be additional value in e-books over and above paper books. For example, immediate delivery, searching, categorizing, word lookup, and e-mail lending provide additional value to e-books.

- **Authenticity of Content**- Consumers need assurance that a particular copy of an e-book is exactly as published by the publisher and has not been tampered with or modified.
- **Preview** – Consumers would like to be able to preview e-books before they buy them.
- **Refund** – Consumers need a way to get a refund if a particular e-book is not what they wanted. The exact refund policy will be at the discretion of the bookseller.

### 1.3.2 Bookseller and Distributor Needs

These needs represent the requirements of booksellers and distributors, whether they are online or brick-and-mortar stores.

- **Copyright Protection** – Booksellers and distributors have a permission that will be granted by publishers: to make copies of e-books for sale to consumers, other booksellers and libraries. Booksellers and distributors require that the e-book system transparently enforce the rights of the copyright owners.
- **Scalability** – A large online bookseller may have a Web site that sells to millions of customers. An online book distributor may have a site that distributes to thousands of booksellers. On the other hand, a small bookshop on Main Street may have a kiosk that sells to a hundred local consumers. The e-book system must be extremely scalable to span these different needs.
- **Liability Security** – Booksellers need the copyright protection system to protect e-book titles on bookseller servers to limit their liability by ensuring e-book titles cannot be stolen from their servers.
- **Publisher Distrust** – Booksellers do not implicitly trust publishers and the system needs to take that into consideration.

### 1.3.3 Publisher Needs

These needs represent the requirements of book publishers.

- **Copyright and Revenue Protection** – The system must enforce the principle that there exists exactly one usable copy of an e-book per purchase.
- **Ease of Production** – Publishers must find it very easy to produce e-books from existing paper book production systems. An ideal solution would be one that simply uses existing production tools and pre-press files.
- **Elimination of Physical Manufacturing** – Electronic books have the potential to reduce the production costs of books and increase the profitability of publishing. However, if an e-book system requires manufacturing of a physical object (e.g., CD-ROM, ROM card), then it will be difficult to reduce costs. Basically, it just moves costs from paper to plastic. Publishers would like option of eliminating physical manufacturing, but some may find it desirable in some cases.

- **Scalability** – A large publisher or imprint may publish hundreds of books each month. It may distribute those books to hundreds of booksellers and distributors.
- **Accountability and Auditability** – Booksellers and distributors provide to publishers an accounting of the e-books that were sold. Publishers should also be able to audit booksellers and distributors to ensure that the numbers are accurate.
- **Bookseller Distrust** – Publishers do not implicitly trust booksellers and the system needs to take that into consideration.
- **Marketing Data** – Publishers would like to find out more about who buys their books. This goal may conflict with the consumer’s privacy requirement, so they must be balanced appropriately.
- **Versioning** – Some books (e.g., computer books) are updated on a very regular basis and publishers need a way to distinguish new versions of e-books and make them available to consumers.
- **Super Distribution** – To provide the widest distribution possible, some publishers would like the capability for consumers to re-distribute e-books, but instead of the consumer losing his/her copy, the new consumer would pay for an additional copy (without the added step of manually going to a bookseller or back to the publisher).
- **Selling** – Some publishers would like to be able to sell e-books directly to consumers, without a distributor or bookseller. The system should therefore allow a publisher to also be a bookseller.

#### 1.3.4 Author Needs

These needs represent the requirements of book authors.

- **Copyright and Revenue Protection** – Authors depend upon the copyright laws to ensure that they are paid for each copy of their book that is sold. In fact, most authors that have been asked for their opinions of electronic books have answered: “how do I get paid?”
- **Self Publishing** – Some authors would like the ability to publish their own works, either because they cannot find a publisher willing to publish their work, or because they think they can do a better job, or because they would like to make more money per sale than they get from a publisher.
- **Interactive Content** – There are some sophisticated authors and their publishers (e.g., academic textbook, reference book authors) who would like to be able to add interactive content like sound, animation, and video to their books.

#### 1.3.5 Library Needs

These needs represent the requirements of public, academic and corporate libraries.

- **Lending and Borrowing** – It is critical to their very existence that libraries have the ability to buy electronic books and then lend them to their patrons.
- **Fair Use** – It is an established legal doctrine that library patrons can excerpt information from books for their own use. Libraries are rightly concerned that electronic copyright protection systems, if poorly implemented, could eliminate fair use. Since fair use is not a simple issue, no technical solution can completely handle it. However, libraries require that e-book systems at least recognize fair use and provide some capability for it.
- **Archiving** – Many libraries need to be able to provide an perpetual archive for their e-books. Therefore, the e-book system must provide more than time-limited licenses.
- **Platform Independence** – While most of the participants in the e-book system have the need for the system to run on almost any hardware platform, the need is even more acute for libraries. They cannot afford to maintain (forever) separate hardware reading devices for e-books in their archives.

#### **1.4 Assumptions**

[**Editorial note.** This section has not yet been edited for Version 0.8.]

The EBX design makes the following assumptions:

- Consumers will, for the most part, prefer to read books off-line. It is still expensive and inconvenient to have a continuous wired or wireless connection to the Internet. The system assumes that most of the time consumers will not be connected while they are reading a book.
- Books will nearly always be exchanged electronically over wired or wireless connections. It seems a step backward to add a physical media-manufacturing step to an e-book system.
- The Internet and World Wide Web is the preferred infrastructure for publishers, distributors, booksellers, libraries and consumers to exchange e-books. Most organizations will want the e-book system to fit into their existing Web servers. Most consumers will want to use their existing Internet Service Providers and e-mail tools.

The information to be protected is not secret. Books are public information. Although a copyright system must be secure against property rights attack, it calls for very different cryptographic techniques than are used to protect confidential information.

#### **1.5 Basic Design**

[**Editorial note.** This section has not yet been edited for Version 0.8.]

The EBX system meets the requirements and assumptions listed above using the following time-tested network and cryptographic technologies.

- Internet Web Servers and Browsers (TCP/IP, SSL, HTTP).
- Public key and symmetric key cryptography (RSA, RC4, DES, SHA, PKCS/X509).

The basic design for publishing and transferring e-books with EBX between components (e.g., publisher to distributor, distributor to bookseller, bookseller to consumer or library, consumer to consumer or library) follows:

- A publisher creates a single content container file (e.g., ZIP file) encapsulating the text and graphics of an e-book.
- The publisher also randomly generates a symmetric content-encryption key (e.g., DES) and uses it to encrypt the text and graphics in the content file. Since a symmetric algorithm is being used, the encryption key is also the decryption key.
- Typically, a commercial organization like a publisher, distributor, bookseller or library puts up a Web site to post the availability of e-books and initiate transfers between organizations. Consumers typically transfer e-books among themselves either by e-mail or via infrared “beaming.”
- Each time an encrypted e-book file is transferred to a client from a server:
  - The client opens a TCP/IP port at the server via an SSL session. SSL is used to ensure privacy during the transaction.
  - The server sends a very large, randomly generated number called a *nonce* to the client.
  - The client’s reading system encrypts the nonce with its private key. It also generates a non-repeating serial number, which, along with a five minute timer for garbage collection purposes, is saved in a record in the reading system’s voucher file. The reading system is personalized at the factory with an EBX-certified public/private key pair. Among other things, the certificate proves that the reading system adheres to the rules of this specification.
  - The client reading system sends an object called *credentials* which contains its certified public key, the encrypted nonce, and the serial number, to the server’s software. The server’s software decrypts the challenge with the public key in the client’s certificate, validates the certificate and, if valid, uses the client’s public key to encrypt the content-decryption key and the serial number, creating a *voucher*.
  - The server then sends the encrypted content file and voucher to the client e-book reading system. The reading system decrypts the voucher containing the content-decryption key and serial number with its private key. Assuming the serial number’s record has not been timed-out by the reading system, the reading system stores the voucher, including the content-decryption key, in its voucher file.
  - When a commercial entity like a publisher, distributor or bookseller uses EBX to transfer an e-book to another entity, its EBX software retains a copy of the voucher and

decrements the copy count in the voucher. Commercial EBX software must also create an audit file, which is used for revenue accounting back to the previous commercial entity in the chain, to account for the number of copies of each book that has been sold.

- When a consumer *gives* an e-book to another consumer or to a library, the consumer's e-book reading system deletes the corresponding voucher from its voucher file. When a consumer or library *lends* an e-book to another consumer or to another library, the consumer's reading system or library's server puts a time limit on the new recipient's voucher and disables its own voucher for the same period of time.
- When a consumer wishes to read an e-book, the e-book reading system uses the content-decryption key in the corresponding voucher to decrypt each encrypted page, or in the case of Open eBook, each chapter.

## 1.6 Vouchers

[**Editorial note.** This section has not yet been edited for Version 0.8. It may be redundant with the voucher format description.]

EBX is primarily concerned with the creation and transfer of digital objects called *vouchers*. A voucher is an electronic description of e-book permissions transferred from one book owner in the network to another book owner. EBX vouchers are encoded in XML.

A book voucher contains the following information:

- ID – ISBN or Digital Object Identifier (DOI) of the e-book content object.
- ContentKey - Content-decryption key (e.g., 56-bit DES).
- CopyCount - Number of copies of the content object the holder of the voucher is allowed to view/lend/give/sell.
- Permissions - Various permissions that the holder of the voucher has:
  - Lendable – Whether the holder can lend the voucher.
  - Givable – Whether the holder can give the voucher to another entity.
  - Sellable – Whether the holder can sell the voucher to another entity.
  - LendingTimeout – Amount of time the holder is allowed to borrow the voucher.
  - PersonalUseCopies – Maximum number of personal use copies per PersonalUseTime.
  - PersonalUseTime – Day, Week, Month, Year.
  - PersonalUseCopySize - Paragraph, page, chapter, whole.

The ContentKey in a voucher is encrypted using the public key of the owner of the voucher, and can only be decrypted with the owner's corresponding private key, which is not divulged, even to its owner.

[This is a crucial difference between the use of cryptography in a rights protection system like EBX and the use of cryptography in a typical secret information system like S/MIME e-mail. In a rights system the private key is *not* made accessible or known to the person who is the owner of the key. This may seem odd at first glance, but the whole point of an electronic rights system is that the person is not trusted to perform only legal operations on the protected content. Protecting the content from the person using the content enables the system to ensure that the person uses the content in legal ways.]

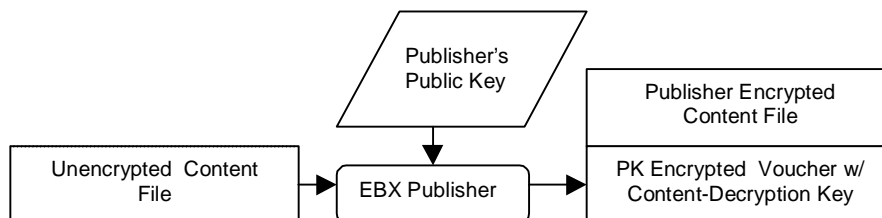
## 1.7 Functional Model

[**Editorial note.** This section has not yet been edited for Version 0.8.]

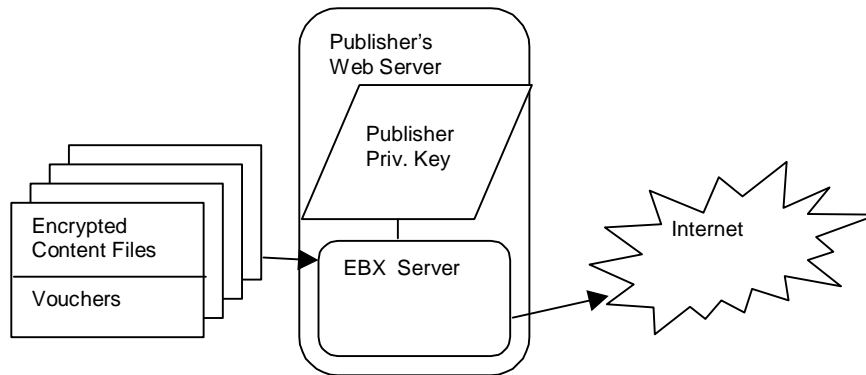
### 1.7.1 Publishing

For a publisher to create an EBX formatted e-book, the following steps are performed.

- The publisher licenses, from a certified EBX software vendor, an **EBX Publisher** software utility and an **EBX Server**, which is typically a Web server add-in. Both packages contain an EBX certified public/private key pair.
- The publisher formats each e-book title into a content container file (e.g., ZIP file) using standard software (e.g., Adobe PageMaker, Adobe FrameMaker, QuarkXPress, Microsoft Word or Publisher, PKZIP, WinZIP).
- The publisher uses the EBX Publisher utility to encrypt the content file. The EBX Publisher uses a single randomly generated content-encryption key to encrypt the content file for the specific title. The EBX Publisher then creates a Voucher *template* for the content file by encrypting the content-encryption key with the publisher's public key.



- The publisher maintains an Internet Web site for use by authorized booksellers and distributors. The publisher installs the EBX Server add-in software to its Web server. The publisher adds all the encrypted content files and the corresponding vouchers to its Web server.

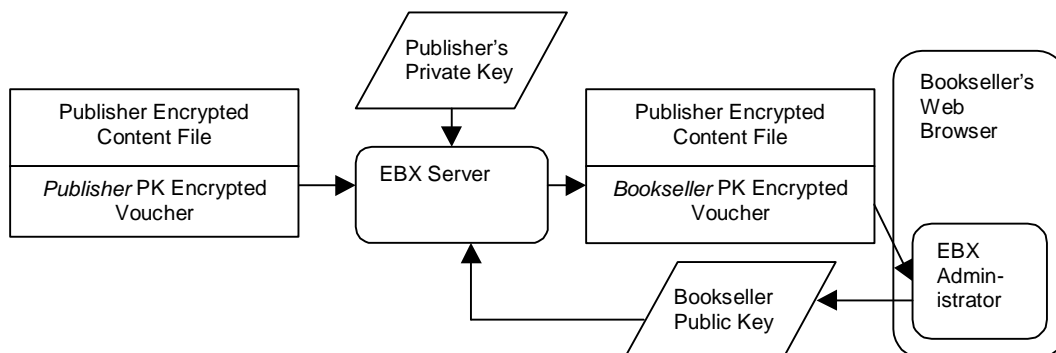


### 1.7.2 Distribution to Online Booksellers and/or Distributors

To transfer a book from a publisher directly to a bookseller or from a publisher to a distributor and then to a bookseller, the following steps are performed.

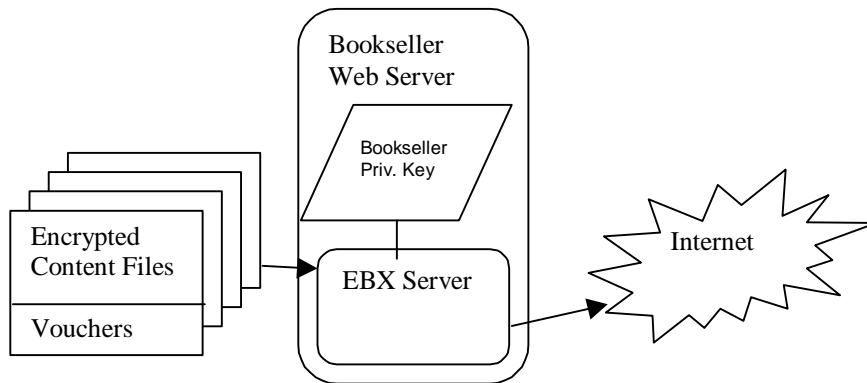
- The bookseller licenses an **EBX Server** Web server add-in from a certified EBX software vendor. An administrator at the bookseller also installs an **EBX Server Administrator** Web browser add-in to his/her browser.
- For each e-book that the distributor/bookseller wants to offer for sale on the distributor/bookseller's Web site, the administrator uses an EBX Server Administrator-equipped browser to download the appropriate EBX encrypted content file from the publisher's Web site. The publisher's EBX Server performs the actual download.

The publisher's EBX Server decrypts the desired voucher using the publisher's private key, sets the bookseller's permissions and copy count, and re-encrypts the voucher using the bookseller's public key. The EBX Server looks-up the bookseller's appropriate redistribution permissions and credit allowance in a database to set the copy count and permissions in the new voucher. The encrypted voucher is transferred to the bookseller. The distributor/bookseller then adds the encrypted content file to its own on-line Internet bookstore Web site.



- The distributor/bookseller installs the EBX Server to its on-line Internet Web server bookstore.



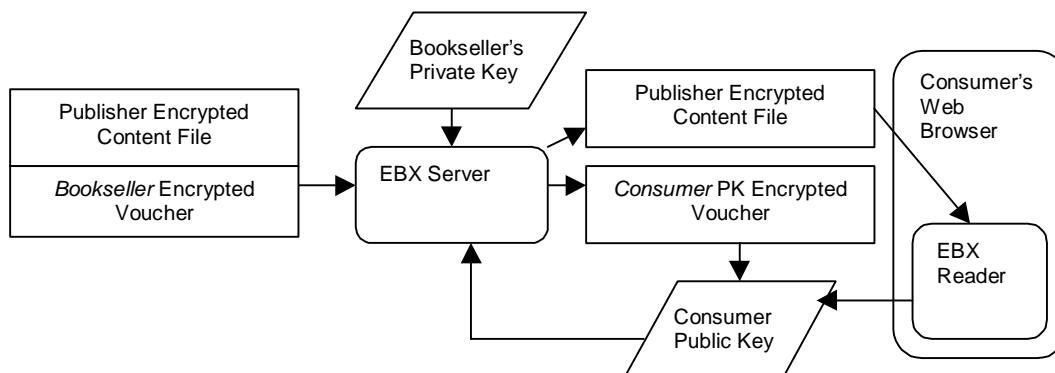


### 1.7.3 Delivery to Consumers

To deliver a book to a consumer from a bookseller the following steps are performed.

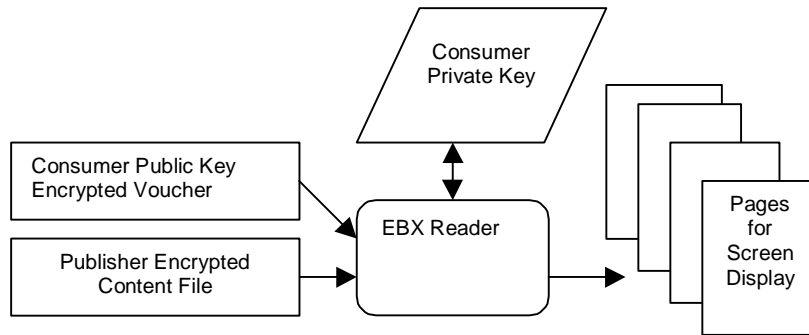
- The consumer purchases, from a certified EBX e-book reading system vendor, an EBX reading system. The EBX reading system is also installed as an add-in to the consumer's Web browser.
- For each e-book that the consumer wants to purchase from the bookseller's Web site, the consumer uses his/her EBX reading system to download the appropriate EBX encrypted content file and voucher from the bookseller's Web site. The bookseller's EBX Server performs the actual download. The bookseller's EBX Server decrypts the file's corresponding voucher using the bookseller's private key. The EBX Server creates a new voucher from the bookseller's voucher combined with the applicable permissions for a consumer (e.g., 1 copy, lending and giving allowed, printing not allowed). The EBX Server then encrypts the new voucher with the consumer's public key.

The consumer's EBX reading system can then use its private key to decrypt the voucher. The new voucher is stored in the consumer's voucher file on the consumer's reading device.



- Vouchers containing content-decryption keys are always stored encrypted using the EBX reading system's public key and are not otherwise revealed. Therefore, to decrypt the e-

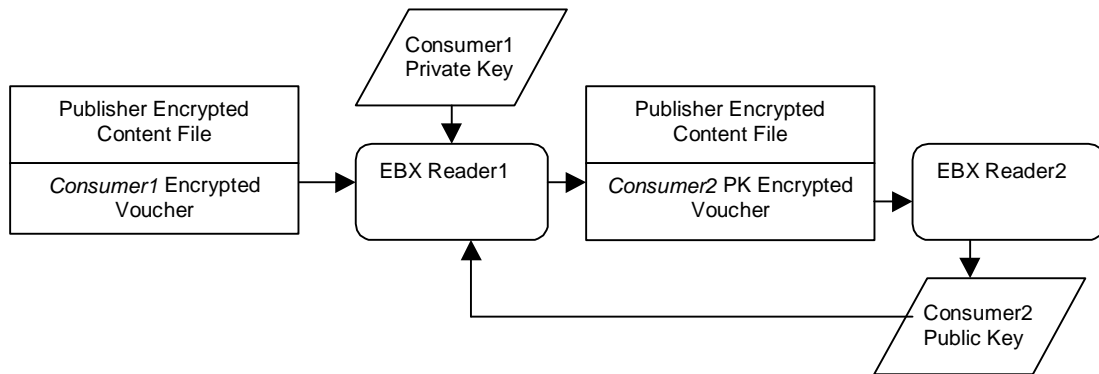
book, the EBX reading system using the corresponding voucher explicitly decrypts each encrypted page.



#### 1.7.4 Transfer Between Consumers – “Give/Lend”

Consumers can transfer books among themselves in two different ways. They can either “give” a book to another consumer, in which case the giver is giving up all rights to the book forever. Consumers can also “lend” a book to another consumer, in which case the lender is giving up all rights to the book for a specified period of time (e.g., 2 weeks). At no time is it allowable for two or more consumers to have the ability to decrypt the same copy of a book. To transfer a book between two consumers in either case, the following steps are performed:

- Both consumers purchase e-book reading systems from a certified EBX e-book reading system vendor.
- When a consumer wants to give or lend a book to another consumer, the receiver/borrower’s EBX reading system generates a random serial number and sends it with its certified public key to the giver/lender’s EBX reading system. If the certificate is valid, then the giver/lender’s EBX reading system creates a new copy of the appropriate voucher and encrypts it with the receiver/borrower’s public key. If the operation is a “give” then the giver/lender’s EBX reading system deletes its own copy of the voucher. If the operation is a “lend” then the giver/lender’s EBX reading system includes a timeout value (e.g., 2 weeks) with the voucher and sets the “lent” state and the same timeout in the voucher in its own EBX reading system.

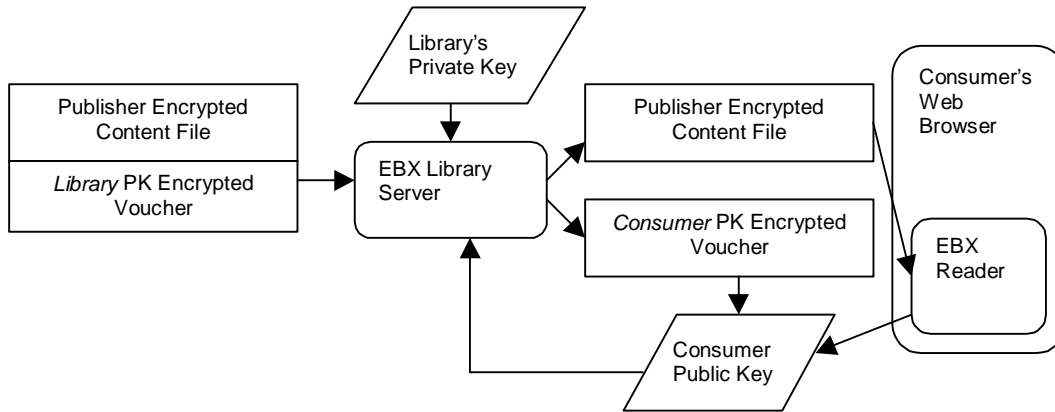


- The giver/lender's EBX reading system software then sends the encrypted voucher, the serial number and the encrypted content file to the receiver/borrower. The receiver/borrower's EBX reading system receives the encrypted voucher and the serial number. The EBX reading system decrypts the voucher and stores it in its voucher file. When the consumer wants to read the book, the EBX reading system uses the content-decryption key in the voucher to decrypt the pages of the book.

### 1.7.5 Transfer Between Libraries and Consumers

Libraries can also lend books to consumers. Transferring a book between a library and a consumer is very similar to give/lend between consumers and the following steps are performed:

- The consumer purchases, from a certified EBX e-book reading system vendor, an e-book reading system. The Library purchases an **EBX Library Server** Web server add-in for their Web server.
- The library purchases e-books in much the same way as consumers. Libraries may typically purchase multiple copies of popular books (i.e, their vouchers contain CopyCounts > 1).
- When a consumer wants to borrow a book from a library, the borrower's EBX reading system generates a random serial number and sends it with its certified public key to the library's EBX Library Server. If the certificate is valid, then the Library Server decrypts the appropriate book's voucher using its own private key. The Library Server creates a new voucher from the original voucher to include a timeout value (e.g., 2 weeks) in the new voucher, sets the same timeout in its own corresponding voucher, and then encrypts the new voucher with the borrower's public key.



- The EBX Library Server then sends the encrypted voucher and the encrypted content file to the receiver/borrower. The consumer's EBX reading system receives the encrypted voucher and stores it in its voucher file. When the consumer wants to read the book, the EBX reading system decrypts the voucher and uses the content-decryption key in it to decrypt the pages of the book.

## 2 Trust Model

The *trust model* in the EBX specification is concerned with computer security principles, with the particular needs of copyright holders, and with “ratings” that define the level of security (and thus the trustworthiness) of a particular vendor’s EBX-compliant product.

### 2.1 *Reliance on Trust for Negotiation of Details*

Eventually, EBX reading systems and servers will constitute an “EBX network” made up of several manufacturers’ products. Where two components share all the other necessary attributes (for instance, if both are based on PDF files or both are based on OEB files of some type), they will actually exchange content. For instance, a single vendor’s server could serve e-books to several vendors’ reading systems.

In order for two components, such as a reading system and a voucher server, to interoperate, they must communicate with each other to find out what they have in common. They can ask each other (or “negotiate”) such questions as what algorithm is used to encrypt the content file, what key length is used, and the type of the content (Open eBook file, PDF file, etc.).

Such a negotiation is part of the standard EBX protocol, so an EBX-compliant server is *required* to answer unsolicited queries of this kind from an EBX client. However, in order to ensure the integrity of the overall network, that is, to avoid giving away this information to potentially hostile clients, we need a means of establishing the server’s trust in the client.

This is analogous to a conversation between two people: If someone confronts you and insists that you answer sensitive questions, it is not enough to know that you are in a “secure” room and can’t be overheard or to be assured that your answers will be kept confidential. You also have to trust the person asking you, either implicitly, because you know him personally, or else because he has been positively identified to you in some way that conveys his authority to ask the questions.

### 2.2 *Results and Effects of the Trust Model*

The trust model itself defines, in prose, the conceptual framework and assumptions associated with protecting rights across the distributed system architecture. It explains the requirements for and responsibilities of the various security mechanisms employed in the system architecture, enabling distributed EBX-conformant components to enforce digital rights to a specified level of assurance.

The related Foundation Trust Services specify the services that each component must perform correctly to both manage the digital rights of protected content and preserve the integrity of the system. The trust services are much more specific than the general trust model. The main “deliverable” of the trust services is a system for the mutual authentication of clients and servers. By virtue of this system, it makes sense to say that the server in the previous example must answer all the questions about its implementation choices, because the questions are coming from a trusted member of the EBX network.

### **2.3 Need for Prescription**

The authentication system is one place in the EBX specification where it is necessary to prescribe certain implementation properties. Once trust is established between two components, many other details can be negotiated on the fly, and not all vendors need to use the same technology. (An obvious example, at this time in history, is that e-book vendors use several different file formats for content.) However, to establish trust in a rigorous manner requires that the client and server authenticate each other and authenticate each other in specific, standard ways, and therefore we require the use of certain technologies, if not for all time then at least for particular generations of this specification.

### **2.4 Trust Model Overview**

Publishers and other copyright holders want to know the level of protection provided to their digital content when it is distributed electronically. Furthermore, recognizing that higher levels of protection often come at the expense of other desirable properties, they want the ability to set different levels of protection for different titles.

The technology provides some protections, but social processes including deterrence, detection, and legal enforcement as well as insurance-based compensation for parties that fall victim to cracks in the system are integral to the overall protection of digital rights. In balancing the contributions achieved through these various approaches, EBX provides assurance to all parties—publishers, authors, insurers and law enforcement—that adequate precautions have been taken to protect the intellectual rights of authors and publishers.

#### **2.4.1 Behavior of Components in a Multivendor Environment**

In the open EBX architecture, a number of distributed components including servers and end-user readers from multiple vendors cooperate to enforce digital rights. These components provide persistent copyright protection via secure authentication, secure transfer and controlled exposure of the vended content according to both the rules defined for the system as a whole and the rights associated with each particular digital title. *The fundamental rule enforced by every component is to deny any right unless explicitly granted.* Consequently, associated with each digital content title is a set of permissions that grant particular rights under specified conditions. Included in the specified rights are rules defining the assurances required before control over a particular digital title can be transferred from one component to another.

#### **2.4.2 Future Versions**

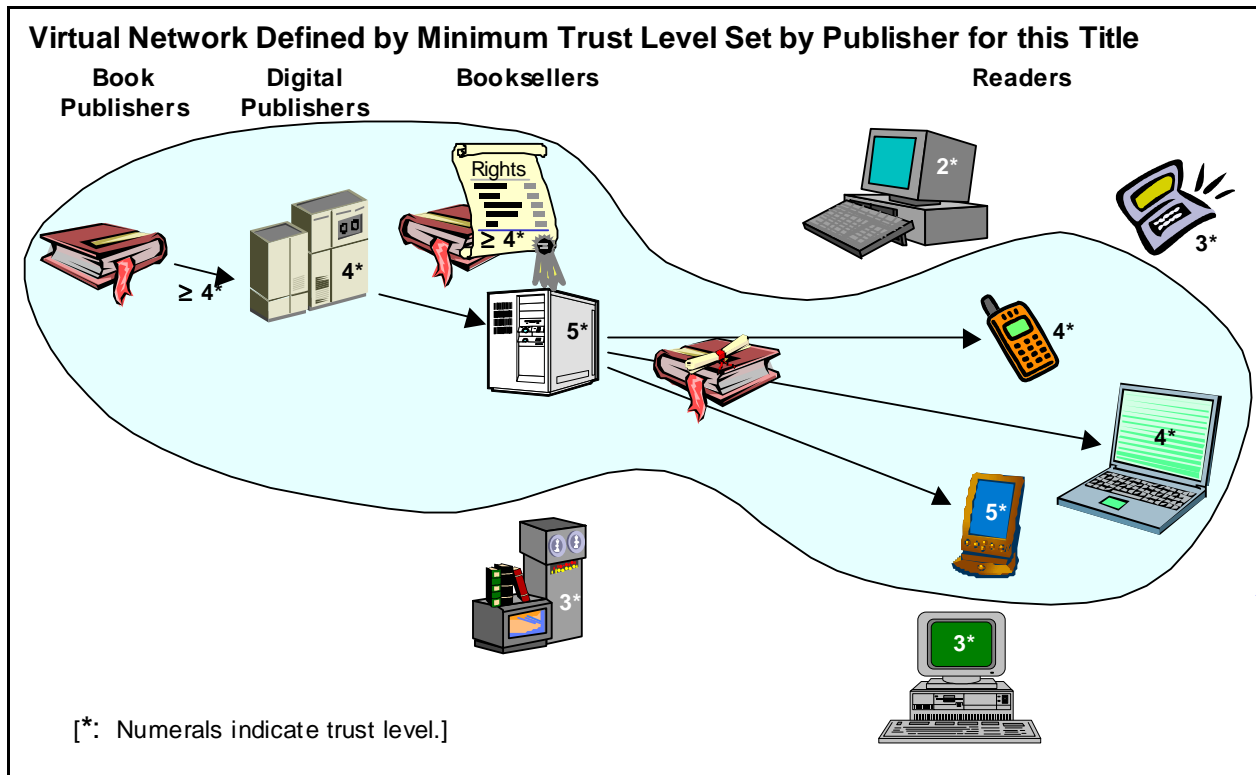
Anticipating that the functional requirements placed on the trust foundation are likely to change over time, the Trust Model mandates mechanisms in the EBX specified protocols to allow forward and backward interoperability between different generations of the trust foundation components. This enables enhancements in the trust foundation over time without requiring updating or retiring of earlier-generation components.

### 2.4.3 Evaluation and Rating of Products from Different Manufacturers

In developing the trust model, we anticipate that some of these components will provide greater protection for digital content than other components. As any system can at best be only as effective in providing these protections as the weakest participating component in the system, the Trust Model defines the concept of trust level associated with each component in the network of participating content-protecting components.

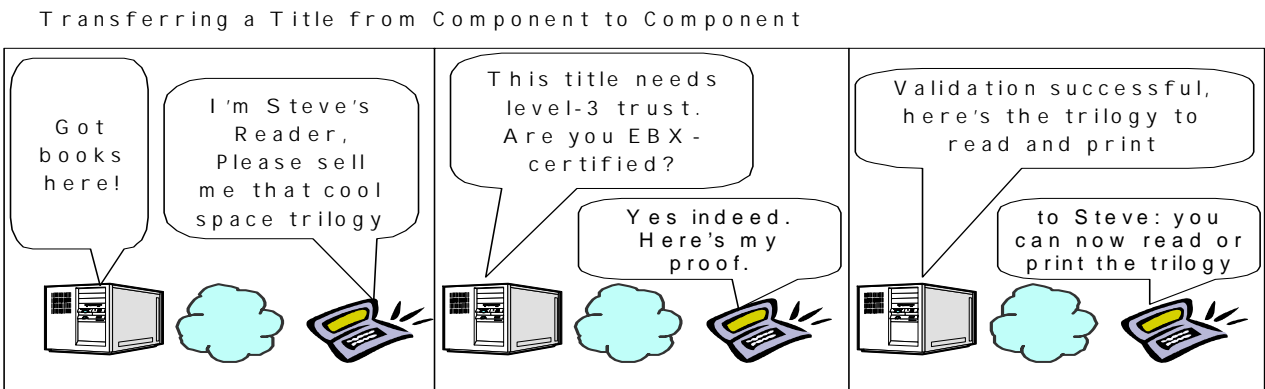
A managed evaluation process certifies each component with respect to the level of protection it provides to the digital titles it processes and its ability to identify itself securely, prove its authorizations and validate the identify and authorizations of other components in the system with which it cooperates. This enables the publisher (or copyright holder) to define a subset of the full network of components that he or she trusts to provide the level of protection for each digital title appropriate to its value and other characteristics, as shown in Figure 1.

Publishers can then make tradeoffs, title by title if they choose, between the level of protection and other properties of components such as cost, the number of units in service or the demographics of users of those units. At the same time, the trust model allows vendors to offer products that provide different levels of protection and that can all participate and interoperate in a single digital content protection and distribution system.



**Figure 1: Subset of components that can handle a "trust level-4" eBook.**

The role of “publisher” in this specification means the originator of the content in the EBX network. Although the publisher may not own the copyright for the book, the publisher is acting on behalf of the copyright holder and is responsible for the content’s safe handling. Once the publisher has introduced the content into the EBX network, responsibility shifts to EBX. Each component that processes a transaction involving the content must correctly enforce the rights associated with that content for that transaction. Before a component passes control of the content to another component it first must establish that the destination component is a proper EBX-certified component authorized to receive and operate on the content consistently with the rules and rights associated with the content, as shown in Figure 2.

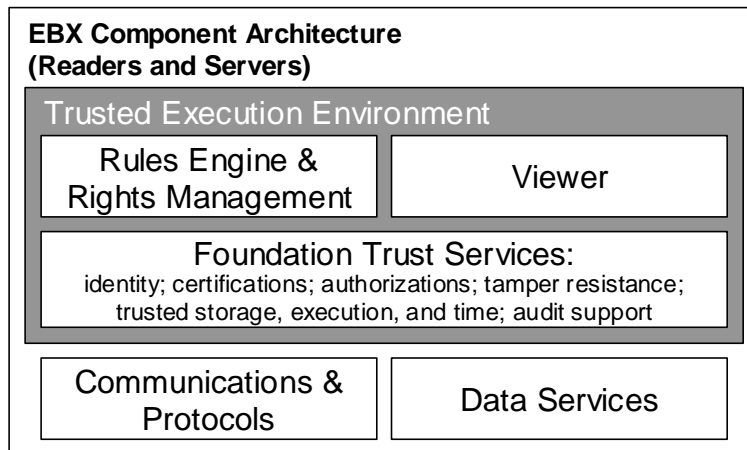


**Figure 2: An upstream component confirms authorization of a downstream component before transferring control of a protected title.**

#### 2.4.4 Trustedness and Trust Services

The capability of the system to enforce the rights specified by a copyright holder for a protected title depends on a basic foundation of trust associated with the system. In turn, trust in the system depends upon trust in each participating component. We characterize trust features in a certified component with respect to the ability of that component to perform a series of functions correctly and resist efforts by an attacker to defeat the correct performance of those functions. Figure 3 below shows one possible architecture.





**Figure 3: Possible Component Architecture**

### 2.4.5 List of Trust Services

The following list identifies the key functions required to establish this foundation of trust. This list also provides an important input to defining the criteria for certifying the trust level associated with each component.

- **Authenticates Self** – A component is able to communicate and prove its own identity to other EBX components with which it communicates and exchanges EBX-protected content and vouchers.
- **Proves Authorization** – A component is able to communicate its capabilities and the services that it is authorized to perform in the EBX system in a manner that can be verified by communicating partner.
- **Protects Own Credentials** – A component protects its own identity and authorization parameters and associated credentials (e.g. name, ID, component type and cryptographic keys used to prove its credentials to other components.)
- **Protects EBX Credentials** – A component stores and protects the EBX system credentials necessary to validate the assertions made to it by other components. (E.g. initializes and securely stores and retrieves the public keys (or public key certificates) of all necessary EBX Root Certifying Authorities.
- **Authenticates Other Components** – A component determines and is able to validate the identity of each other EBX component with which it communicates.
- **Validates Authorization and Trust** – A component determines and is able to validate the capabilities and authority of each other EBX component with which it communicates and the trust level at which it is certified to perform those functions.
- **Enforces Rules** – A component enforces the rules defined both by its own defined functional specification in conformance with the EBX Standard and the particular

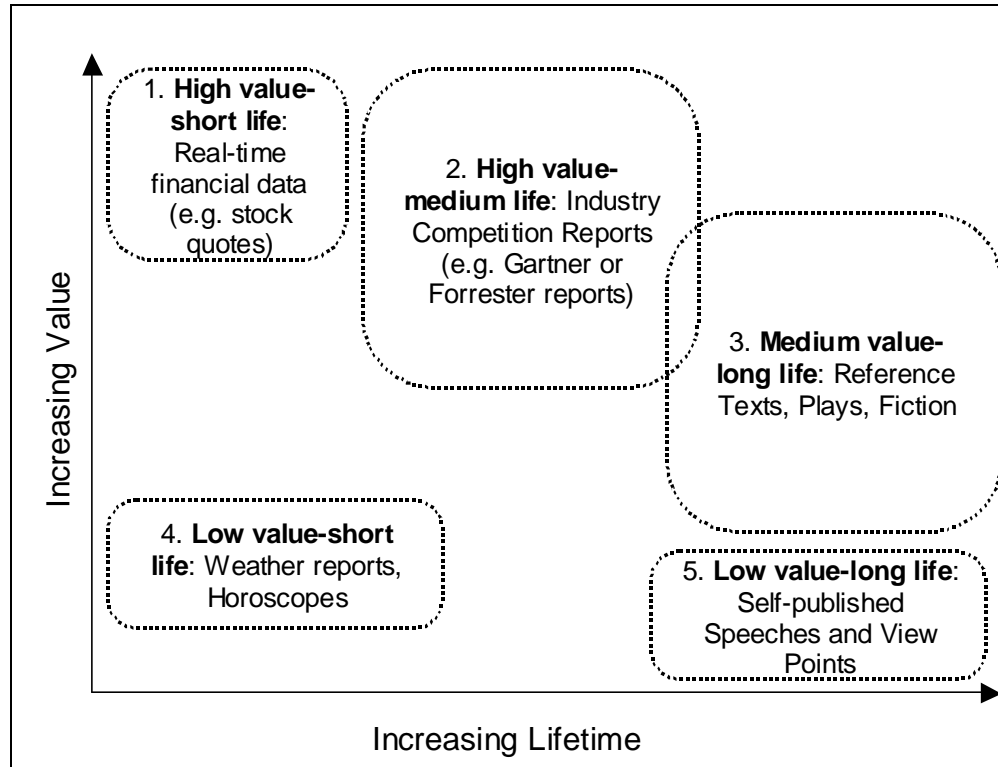
requirements defined for the particular EBX-protected content on which it is operating. A component also secures the EBX content and associated encryption keys so that they cannot be extracted and employed to pirate EBX-protected content.

- **Protects Execution** – A component executes its logic and cryptographic algorithms in a protected environment to ensure correct operations. This also applies to protecting the logging and reporting functions.
- **Detects Invalid EBX Components and Vouchers** – While properly a subset of the functions listed above, an important capability of each EBX component is its ability to identify invalid EBX vouchers and EBX components seeking service and to respond appropriately. (E.g., refusing to deliver content or vouchers to components whose credentials are invalid or have been revoked prevents bogus components from subverting EBX distribution.)
- **Logging and Reporting** – Each component maintains a correct and protected log of transactions processed, exceptions experienced and other significant events. Logged events are reported as defined by specified rules and support response to protection failure incidents, audit, and system investigations.

#### 2.4.6 Factors Affecting the Value of Content

Works are created, produced, sold and distributed for the value derived, including profit, prestige and education, to name but a few motives. Works are ‘stolen’ by illegally copying and / or fraudulently misrepresenting ownership or authorship for similar motives. The effect of time on the value of different titles varies widely. Time must also be considered when we are protecting works since the protection must persist into the future. Attack methods, tools and techniques are expected to improve dramatically over time while little will change with a given security implementation. For this reason, works that must be protected for the life of their copyright present greater challenges than works that lose value after a few days of their publishing such as yesterday’s weather forecast.

Figure 1 shows the relationship between time and value and suggests five example categories. This matrix is shown for illustration. In the real world there is substantially more overlap among these categories.



**Figure 4 Time Value of Information – Example Categories**

The sample categories include:

- 1. High value-short life:** this includes stock quotes and other financial information that has very high value but a lifetime of only a couple dozen of minutes. The analysis of this information aggregated over time continues to hold value of course, but that is a function of its presentation rather than the content and value life.
- 2. High value-medium life:** examples include reports on highly dynamic, competitive industries that are used for business intelligence and strategy.
- 3. Medium value-long life:** some reference works and all fictional works will retain their value for the lifetime of the copyright.
- 4. Low value-short life:** some types of information are given away for next to nothing. They also have a very short value life. Examples include weather reports and horoscopes that are available for free and are only valuable for a few days.
- 5. Low Value-long life:** this category of information covers authors that publish in order to share their viewpoints. Copying and wide dissemination is encouraged as long as the content is not changed and the author receives the appropriate credit for the work. Value of the works

extends to the lifetime of the copyright (see footnote #1). Examples include Amnesty International reports and political position papers.



## **2.5 Characterizing Trust Levels**

The ability of the system to preserve system integrity and protect content derives from the ability of the participating components to provide the trust services identified above with a known level of assurance. Because current technology cannot provide perfect security without incurring other undesirable characteristics (e.g. high cost, difficulty of use, inflexibility, etc.) we approach this problem by permitting components with different trust capabilities to interoperate, but enable the copyright holder to specify a minimum trust assurance for the handling of a particular protected title.

Four factors rank high in assessing and certifying the trust level provided by a component. First is the scope of a particular failure. If a single feasible failure has more serious consequences in one device than another, then the former ranks at a lower level of trust. Second is the skill level required to defeat the protecting mechanisms. The higher the level of skill necessary to wage a successful attack, the more trusted is a component. Third is the sophistication and availability of the tools necessary to wage an attack. The more widely available or accessible are the tools required to defeat a particular security mechanism, the lower the trust level of the component. Fourth and probably most important, the overall level of resources, including manpower, time, computer processing capacity, etc. factors into the trust level for a component. An encryption algorithm that succumbs to a Pentium-day exhaustive key search attack offers a lower level of trust to an algorithm that requires multiple Pentium-years to crack by key search.

### **2.5.1 Scope of Particular Failures**

We consider attacks directed against the content itself as well as attacks aimed at compromising a component in the EBX system, characterizing the resulting failure roughly in terms of the number of protected titles potentially compromised by the successful attack.

Increasing cost to product vendors 				
	Attacks Waged at Protected Content	Attacks Waged Against a Single Component	Attacks Waged Against the Infrastructure Integrity	
Increasing cost to copyright holders 	<b>Leak</b>	Enables a single unlicensed copy of a single title	Enables an unlicensed copy of a single title on one device	
	<b>Spill</b>	Enables multiple unlicensed copies of a single title, or enables a single device to view all titles without licensing them	Enables unlicensed copies of a single title on multiple devices, or unlicensed access on a single device to all titles	Failure of a single component's basic trust capabilities
	<b>Flood</b>	Enables unlicensed access to a title on multiple devices or strips a title of content protection	Strips titles of content protection or enables unlicensed access to all titles	Trust foundation failure in multiple components in a single certified release requiring revocation of a family of devices
			Trust foundation failure in set of components requiring revocation of more than one family of devices	

**Figure 5: Scope of attacks waged against components.**

Clearly, floods are to be avoided regardless of the locus of attack. For electronic books, as usually defined (a digital issue of a well known novel or other book), the difference between a leak and a spill may not be meaningful to a publisher: if a leak produces a high quality copy that is stripped of its protection, that copy can now be distributed ubiquitously without having to attack other reading systems.

### 2.5.2 Skill, Tools and Expense Required to Cause Failures

The following table characterizes in terms of three levels the skills, tools, and total effort necessary to subvert a component's integrity and its ability to protect content.

	Skills Necessary to Effect Failures <sup>1</sup>	Tools Required to Effect Failures	Total Resources to Effect Failures
<b>High</b>	Expert - Requires the dedicated effort of a professional-caliber cracker to attack the multiple layers of tamper-resisting technology, or medium skills to attack the cryptography by brute force.	<ul style="list-style-type: none"> <li>Specialized – Professional tools and equipment such as in-circuit emulators, logic analyzers, custom software applications, specialized decryption engines, etc.</li> </ul>	<ul style="list-style-type: none"> <li>High – Factoring in forecasted improvements in computing, the work effort 10 years hence to break the protective mechanism will require more than \$300,000 in (possibly specialized) equipment dedicated to the attack for over 1 year.</li> </ul>
<b>Medium</b>	Developer - Requires an intermediate skill set such as that of an experienced software developer familiar with the basic technologies employed in the component	Professional – Typical software development tools including debuggers, decompilers, memory reading/writing tools, etc.	<ul style="list-style-type: none"> <li>Medium – Requires \$100,000 worth of today’s equipment operating one month or more.</li> </ul>
<b>Low</b>	<ul style="list-style-type: none"> <li>Novice - Requires a beginner-intermediate skills such as those of an experienced user of computer desktop applications</li> </ul>	Common – General purpose tools used widely and available at modest cost	<ul style="list-style-type: none"> <li>Low – Current standard desktop PC processing for several weeks or less.</li> </ul>

**Figure 6: Skill, Tools, and Expenses Associated with Subverting a Component.**

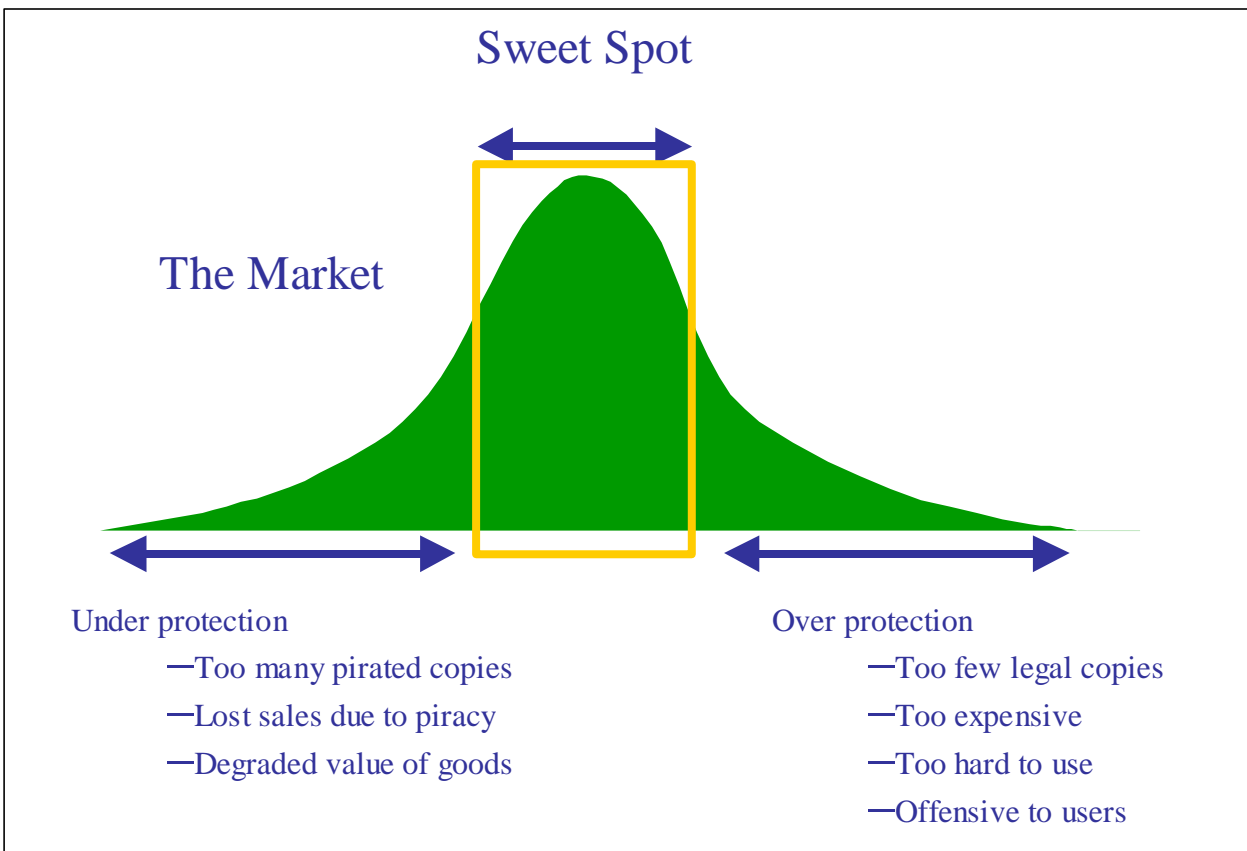
### 2.5.3 Trust Levels and Certification

The EBX system will coexist with societal norms, which include legal recourse and insurance-based compensation for parties that fall victim to cracks in the system. Therefore, EBX provides trust levels that are necessary and sufficient to provide reasonable assurance to all parties—publishers, authors, insurers and law enforcement—that adequate precautions have been taken to protect the intellectual rights of authors and publishers.

---

<sup>1</sup> It is important to keep in mind that someone with the skills shown may be able to create easy-to-use attack tools that someone with lower skills may employ to wage the attack successfully. It is also worth noting that, at least in the United States, Federal law makes it illegal to traffic in such tools.

The trust rating requirements allow publishers to make a time value of information estimation of their work and then set a corresponding trust requirement of the components allowed to access their work. In this way high value, long life works may be protected by highly trusted systems while lower-value, short life works need not be burdened by these requirements. The intention is to provide flexibility in the system such that the needs of all parties will be met. In particular, it allows the publisher to select the right level of protection to maximize the value of a title, avoiding losing sales to excessive piracy at one extreme and losing sales to difficult protection mechanisms or offensive enforcement at the other end of the spectrum.



**Figure 7: Picking the right level of protection.**

It is assumed, at a minimum, that all publishers, regardless of content they produce, will want to ensure the data integrity and the authenticity of their authorship. Therefore, adequately long key lengths shall always be used for the purposes of digitally signing their content.

EBX aims to provide sufficient technical protections against theft and uncontrolled replication of protected content to satisfy the needs of the publishing and distribution business. Inherent to all security systems is the property that they can be defeated by the application of enough resources. Even at the highest level of trust, the EBX systems do not claim to provide security adequate for content that must remain confidential for years into the future. Successful attacks on the EBX protection mechanisms are expected and need to be anticipated in the larger operational assumptions and plans.

The trust model initially defines six levels of trust, 0 (lowest) to 5 (highest) each progressively more trusted than the next lower level and providing all of the features of the lower levels plus the enhanced features introduced at its level. The potential for defining even higher levels is anticipated so the scale is defined to be open ended.

#### 2.5.4 Perspectives on the Six Levels of Trust

We expect this document to be read by people coming from diverse backgrounds including: publishing, writing, legal, technology development, operations, certification, and security. In what follows below, we provide descriptions of each trust level from different perspectives ranging from the technical mechanisms likely to be employed to achieve the particular level of protection to analogies to common processes familiar in the world of business such as insurance. The goal is to make these security levels understandable to the cross section of people with interest in protecting and distributing digital content. However, the association of these assessments derived from different perspectives is inherently difficult and in some cases unknown.

The following table characterizes the several perspectives from which to evaluate the assurance afforded by a component at each level of trust in the hierarchy.

Attack Resources and Expenses: Describes the skill level, tools, and expenses necessary to wage a successful attack against the system to produce a flood-type event or tool to defeat protections on multiple titles.
Class of Titles Appropriately Protected: Proposes typical kinds of content that might be adequately protected by this level of component.
Example Technical Defenses: Describes example technical mechanisms from today's technology that might be employed in an implementation to provide the kind of content protection and defenses appropriate for this level.

#### 2.5.5 Definitions of Trust Levels

The EBX system defines components operating at different levels of trust and provides a mechanism for publishers to specify the minimum level trusted component that may operate on a particular EBX-protected title. The components themselves are certified to perform their services at these specified levels of trust. The rules defined by the EBX specification combined with the authentication and authorization validation mechanisms ensure that a particular EBX-protected title is distributed only through EBX system components that have been certified and are validated at the minimum trust level associated with the content. However, the overall security afforded by a system built from secure components is typically lower than the minimum security of each of the participating components. This section aims to describe in lay terms reasonable expectations for the performance of the system as a whole in protecting EBX content being handled at each of the specified trust levels.



### 2.5.5.1 Level-0 – (lowest) No protection

This level serves those who desire to create content with desktop publishing tools and distribute it for viewing on the same reader platforms used for protecting higher-value content.

- No Protection for Titles
- No Proof of Origin
- No Assurance Regarding Integrity of Infrastructure
- Copying allowed
- Unregistered reader (lacks a unique ID)

Attack<sup>2</sup> Resources and Expenses: \$10 May be as small as none as components are not required to provide any defenses.

Class of Titles Appropriately Protected: Titles of negligible monetary value

Example Technical Defenses: None. Components execute communication protocols, but make no promise to enforce any rules or rights. Note that any title that does not have a verifiable origin and verifiable encoding of rights, but is otherwise displayable, may be displayed. Whatever signal that is employed at Level-1 and higher to indicate to the user that the origin and copyright have been validated shall not be displayed for any Level-0 titles. Revocation of components is not supported.

### 2.5.5.2 Level-1 – Signed by Author/Publisher

- Cryptographically signed by Author/Publisher binding in:
  - Attributes, Rights, Integrity, Origin
- Copying allowed (Works are sent in the clear)
- Registering reader is optional (lacks a unique ID)

---

<sup>2</sup> Successful attack means developing a tool that enables perpetrator to create an unprotected text version of any title protected by EBX assuming the perpetrator purchases one copy of each title to be pirated.

Attack Resources and Expenses: **\$10** Requirements to duplicate the title may be as small as none as content is not encrypted and components are not required to provide any defenses. Stripping off the bound in rights, copyright, etc. in its entirety and converting the title to Level-0 is relatively easy as is republishing parts of or the entire title under some other registered identity's authorship.

**\$300,000+** Altering or forging the copyright, authorship, rights, etc. while preserving the appearance of being a validly produced title by the original publisher or any other registered publisher (not under control of the pirate) requires Developer skills, Professional tools, and medium-to-high expenditure of resources.

Class of Titles Appropriately Protected: Low value-short life (Category 4)

Example Technical Defenses: Sound protection design, but crypto, credential storage, rules, and authentication running in unprotected software. No encryption is provided for the title, but source signature is properly validated and presented to the user. Revocation is not supported. The reader shall employ a distinctive means of indicating to the user when a title's origin and copyright have been validated.

#### 2.5.5.3 Level-2 – Personalize book to purchaser – Honor system

- Cryptographically signed by Author/Publisher binding in:
  - Attributes, Rights, Integrity, Origin
- Purchaser identity cryptographically bound to individual copies
  - Note that this feature is not protected under the Digital Millennium Copyright Act; that law specifically allows a user to circumvent or undo that part of a copyright management system that contains “personally identifying information.”
- Copying subject to rights, but not technically prevented
- Registering reader is optional (lacks a unique ID)

Attack Resources and Expenses: **\$10** Requirements to duplicate the title may be as small as none because the content is not encrypted and components are not required to provide any defenses. Stripping off the bound in rights, copyright, etc. in its entirety and converting the title to Level-0 is relatively easy as is republishing parts of or the entire title under some other registered identity's authorship under one's control.

**\$300,000+** Altering or forging the copyright, authorship, rights, and identity of the purchaser of that copy, etc. while preserving the appearance of being a validly produced title by the original publisher or any other registered publisher (not under control of the pirate) requires Developer skills, Professional tools, and medium-to-high expenditure of resources.

Class of Titles Appropriately Protected: Low Value-long life (Category 5)

Example Technical Defenses: Sound protection design, but crypto, credential storage, rules, and authentication executing in software with limited anti-tamper mechanisms. No encryption is provided for the title, but source signature and the identity of the purchaser of each copy is cryptographically bound into the title. The reader shall employ a distinctive means of indicating to the user when a title's origin and copyright have been validated. Revocation is not supported.

#### 2.5.5.4 Level-3 – Software DRM

- Range of rights specified and enforced
- Strong encryption protecting content
- Implemented in unprotected software
- Individually registered
- Voucher servers individually revocable

Attack Resources and Expenses: \$1,000-\$10,000 Developers skills, professional tools, 1-30 days to reveal the protected content.

\$300,000+ Altering or forging the copyright, authorship, rights, and identity of the purchaser of that copy, etc. while preserving the appearance of being a validly produced title by the original publisher or any other registered publisher (not under control of the pirate) requires Developer skills, Professional tools, and medium-to-high expenditure of resources.

Class of Titles Appropriately Protected: High value-short life (Category 1) & lower-value/shorter-life categories 2 & 3

Example Technical Defenses: Business caliber cryptographic algorithms and key lengths, but crypto, credential storage, rules, and authentication executing in software with limited anti-tamper mechanisms such as those based on obscurity. The reader shall employ a distinctive means of indicating to the user when a title's origin and copyright have been validated. Each unit is initialized uniquely; and the infrastructure supports revocation.

#### 2.5.5.5 Level-4 – High Value

- Resistant to the well equipped lone hacker
- Component's private key and other credentials contained in and obfuscated by special software or hardware techniques that resist debugging/sniffing tools and defend against virus and Trojan horse attacks;
- Decryption of content occurs in a secure, trusted environment;
- Revocation procedures in place for rogue devices and content servers.

Attack Resources and Expenses: **\$30,000-\$100,000** Requires expert skills at reverse engineering obfuscated or other tamper resistant technology and specialized and/or custom tools. 10-60 person-day effort.

**\$300,000+** Altering or forging the copyright, authorship, rights, and identity of the purchaser of that copy, etc. while preserving the appearance of being a validly produced title by the original publisher or any other registered publisher (not under control of the pirate) requires Developer skills, Professional tools, and medium-to-high expenditure of resources.

Class of Titles Appropriately Protected: Higher-value/longer lifespan categories 2 & 3

Example Technical Defenses: Business caliber cryptographic algorithms and key lengths bolstered by credential storage, rules, and authentication executing in a tamper resistant engine achieved by techniques such as cryptographically obfuscated software or tamper resistant hardware. The reader shall employ a distinctive means of indicating to the user when a title's origin and copyright have been validated. Each unit is initialized uniquely; and the infrastructure supports revocation.

#### 2.5.5.6 Level-5 (highest) – Industrial Level Hacking / Cryptanalysis Required

- Coercion more serious threat (insider influence)
- Hardware solutions
- Hardware must have tamper detection and appropriate shutdown procedures;
- Password guessing shall be detected and the component shall have appropriate shutdown procedures;
- Hardware must have provisions to protect secrets in all modes of operation (startup and shutdown included);
- Intended to protect all category 2, 3 and most of category 5 content from all but intelligence agencies;

Attack Resources and Expenses: **\$300,000-1,000,000** To attack either the content protection or signed metadata requires expert skills at reverse engineering obfuscated and tamper resistant technology implemented in both hardware and software employing specialized techniques and significant quantities of specialized tools or a brute force attack on the cryptography employing significant computing resources. 100+ person days effort. (Need also to consider elapsed time.)

Class of Titles Appropriately Protected: Highest-value/longer lifespan categories 2 & 3

Example Technical Defenses: State of the art crypto credential storage benefiting from both software and hardware mechanisms, rules, strong authentication, authentication executing in highly tamper resistant hardware, each unit is initialized uniquely; and the infrastructure supports revocation. The reader shall employ a distinctive means of indicating to the user when a title's origin and copyright have been validated.

### 3 Foundation Trust Services

This section describes technical approaches mandated for the Release 1.0 trust foundation services. It also includes analysis of some of the choices made and discussion on topics related to implementing the trust foundation for an EBX-certifiable component.

#### 3.1 PKI: Foundation Mechanism for Authentication

PKI is employed by EBX to perform the identification, certification, and authorization functions required by the trust model. PKI was selected in favor of other authentication alternatives because it affords the following benefits in the EBX context.

- PKI offers sufficient mechanisms to provide the identification, certification, and authorization functions required by the trust model;
- A simple hierarchy of CAs and RAs enables the distributed control and operations EBX needs much more flexibly than central server-based solutions;
- PKI readily enables revocation of EBX server components, that are relatively few in number, as well as providing the mechanisms to support revocation of individual reading-only systems, although the latter is optional under the EBX specification;
- Multiple existing providers offer standard PKI products and services so that both EBX and vendors building EBX components have ample choices in how to realize the necessary mechanisms.

##### 3.1.1 Requirements

The requirements that must be satisfied by the EBX PKI are:

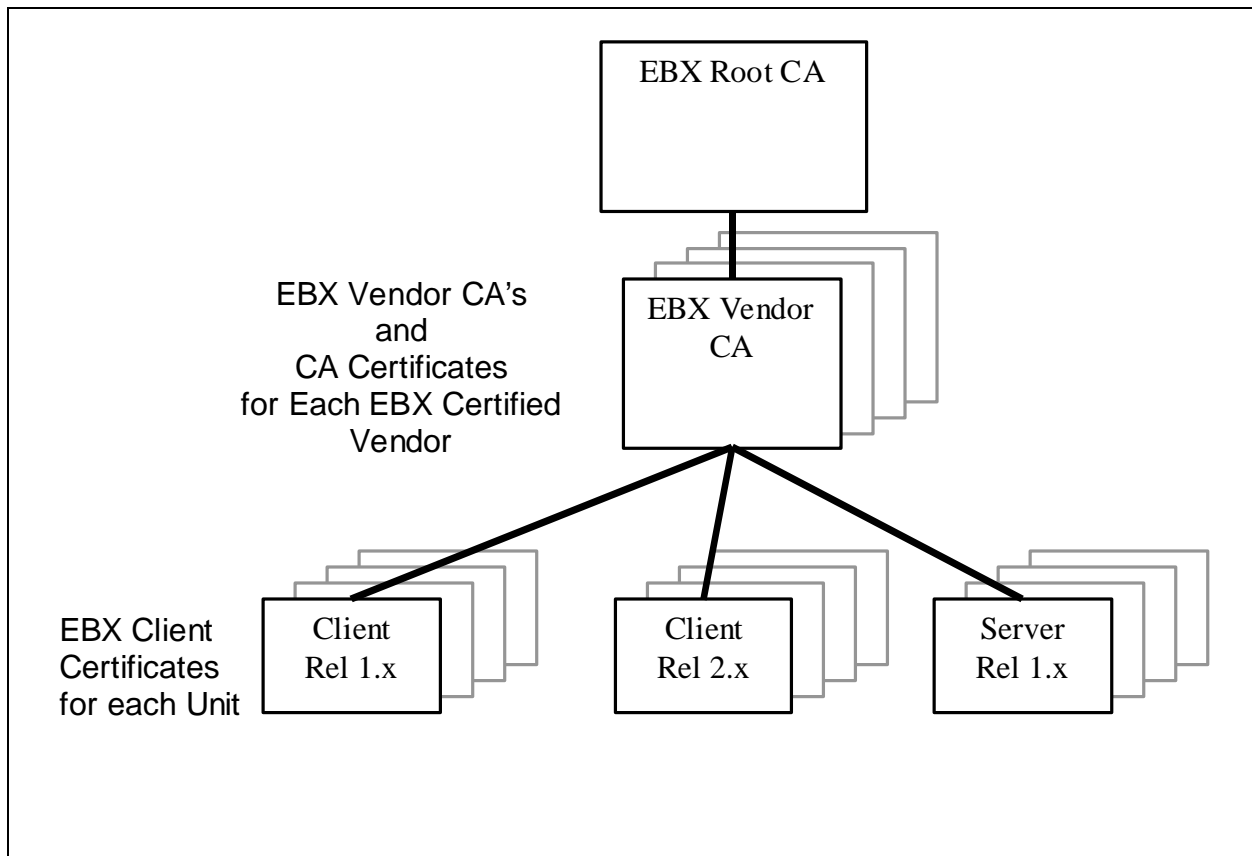
1. Issue unique identifiers for each instance of an EBX client and server.
2. Authenticate that a particular client instance is an instance of an approved EBX-compliant implementation.
3. Allow interoperability among different EBX client and server implementations, so that a server implemented by one vendor can authenticate a client implemented by a second vendor.
4. Identify the EBX component of a particular client or server instance.
5. Identify the role { **Publisher** | **Distributor** | **Client** } an EBX component is authorized to perform.
6. Identify the maximum trust level for which an EBX component instance is approved.
7. Revoke certification of a client or server implementation.
8. Revoke certification of individual server instances.

9. Revoke certification of individual client instances (optional).
10. Check the revocation status of implementations or instances.
11. Allow for authentication of clients from other domains that already have an established PKI (e.g., wireless phones.)
12. Allow for extensibility of the hierarchy of trust.

Each requirement is met by one or more components of the PKI, as explained in the following section.

### 3.1.2 Overall EBX Certificate Authority Architecture

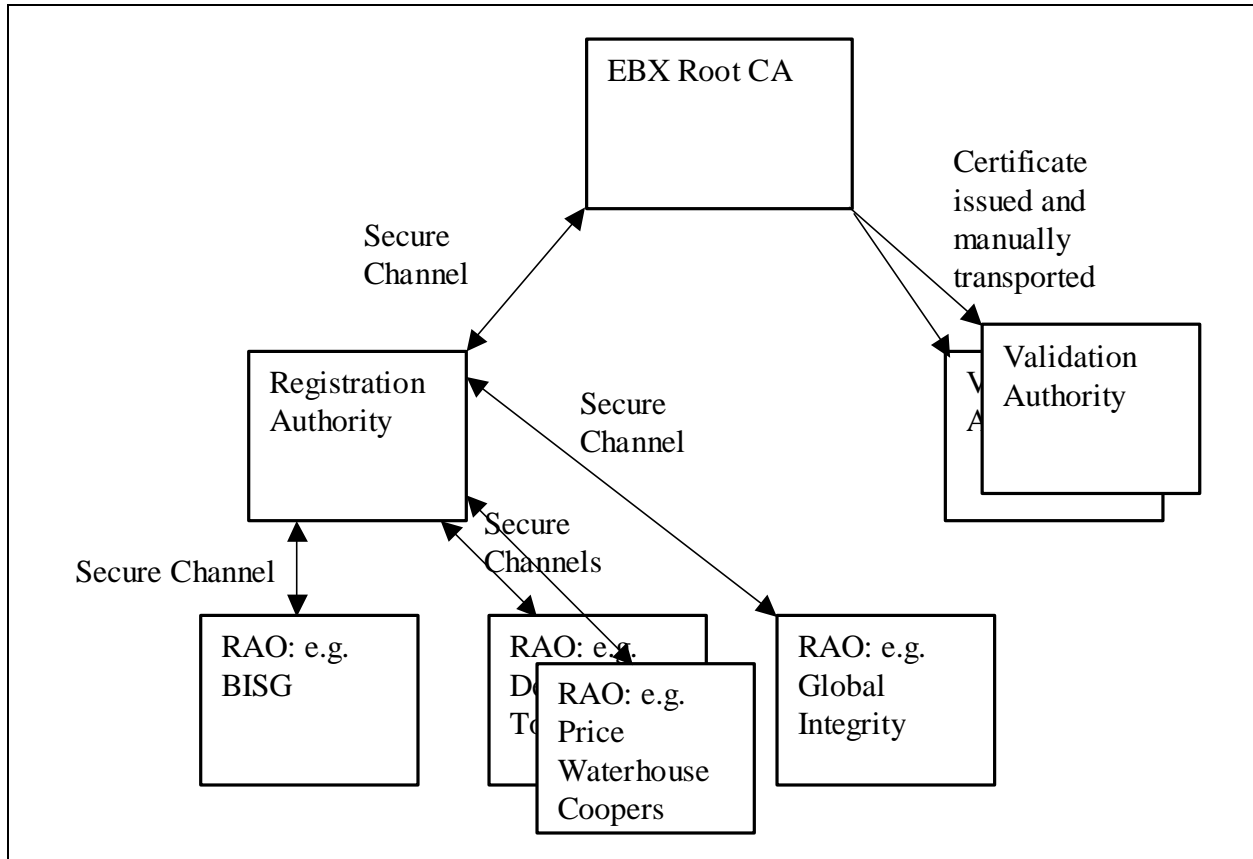
The public key infrastructure consists of several components: the EBX certification authority (CA) and related software and systems, the organization(s) chosen to operate the EBX CA and its related systems, vendor CAs and related systems, the organizations chosen to operate vendor CAs and systems, and finally, certificate processing services implemented by vendors.



#### 3.1.2.1 EBX Root Certificate Authority

The EBX CA consists of a Certification Authority, hosted by a CA Service Provider, a Registration Authority (RA), hosted by the CA Service Provider, one or more Registration

Authority Operators (RAOs), and one or more Validation Authorities.



The EBX Root CA issues X509 v3 certificates to vendor CAs, which in turn issue certificates to client and server instances. The EBX Root CA only issues certificates when it receives a certificate request from the Registration Authority. We expect the Registration Authority and the EBX root CA would be hosted by a single service provider. However, the architecture allows them to be hosted by two different service providers. Trust between the RA and the CA is established manually by the service provider(s).

When the EBX Root CA issues a vendor CA certificate, it is delivered via email to the email address provided by the vendor. It is then the vendor's responsibility to install the certificate into its own CA.

### 3.1.2.2 Registration Authority and Registration Authority Operators (RAOs)

The Registration Authority accepts certificate requests in PKCS#10 format from vendors either via email or via an online protocol. It is the vendor's responsibility to ensure that the certificate request is in the correct format and contains all of the correct fields. The RA then waits for authorizations from the RAOs. It then forwards the certificate requests to the EBX Root CA after receiving authorization from one RAO in each category.



Each category of RAO handles a different aspect of the EBX implementation certification process.

- The BISG verifies that the vendor requesting a certificate is a member of the EBX working group.
- The auditor RAO verifies that the vendor is a legitimate business and is not out to steal money from publishers, booksellers, or consumers.
- The implementation RAO verifies conformance to the EBX spec and establishes the maximum trust level for the implementation. Each RAO operates under guidelines set out in the EBX Certification Policies and Procedures document.

Each RAO must submit its policies and procedures to the EBX Working Group, and its policies and procedures must be approved before it can begin approving certificate requests. The EBX Working Group will ask for reports, from time to time, which will be used to refine the policies and procedures used by all RAOs of the same category. RAO reports should be detailed but will not be confidential: The EBX Working Group will use findings from these reports to refine the EBX specification and its associated public documents.

Once the RA receives authorizations from one RAO in each category, it forwards the certificate request to the EBX Root CA. The RA sends additional information, such as the maximum trust level, to the EBX Root CA, which then adds the appropriate extension fields to the certificate. The communication mechanism for this additional information is not specified, as it depends on the CA service provider. The CA service provider must ensure (to the satisfaction of the EBX Working Group) that the integrity of the certificate request data and additional field data is not compromised in transmission between the RA and the EBX Root CA.

### 3.1.2.3 Validation Authorities

The Validation Authority may be hosted by the same organization as the CA service provider, or by a separate organization. The Validation Authority uses Online Certificate Status Protocol (OCSP – RFC 2560) to provide timely information on the revocation status of certificates. The EBX Root CA must issue a certificate to the Validation Authority with the extendedKeyUsage field set to indicate that the Validation Authority is authorized to perform validation on certificates issued by the EBX Root CA. Each vendor CA certificate issued by the EBX Root CA must conform to the certificate content requirements in RFC2560, section 3.1, so that the OCSP client (an EBX server or client) can determine where and how to check the revocation status of the certificate.

### 3.1.3 Vendor Certificate Authorities

Each vendor CA issues X.509 v3 certificates to “end entities,” which are instances of an EBX client or server implementation. A vendor may choose to set up its CA using any combination of CA service providers, RAs or RAOs. It may choose to use manual or automated processes to provide unique identification to each client and server instance. The vendor’s choice of policies and procedures is examined as part of the EBX certification process. Guidelines for vendors in creating their policies and procedures are included in the EBX Certification document.

The key requirements of the vendor CA are that it issue a unique certificate per client and server instance (Requirement 1), and that the certificates conform to the format in the EBX Specification, including all required fields and extensions (Requirement 3, Requirement 4). Additionally, a vendor CA must only issue certificates indicating a trust level at or below the maximum trust level approved for the client or server implementation (Requirement 6.)

A vendor CA may issue certificates that contain the OCSP extensions, so that EBX servers can check the revocation status of individual client or server instances, and so that the vendor can revoke certificates of individual users. In this case, the vendor must set up the appropriate Validation Authority to perform the certificate validations. The Validation Authority associated with the EBX Root CA only validates the revocation status of certificates issued directly by the EBX Root CA.

On the other hand, a vendor may choose not to implement a Validation Authority, and so does not need the OCSP extensions in its certificates. The result of this choice is that a revocation causes an entire class of clients or servers to be revoked, and there is no revocation of individual users' certificates.

A vendor CA may issue certificates containing additional extensions, but none of the additional extensions may be marked as critical.

### 3.1.3.1 Vendor Certificate Services

Each conforming implementation of an EBX client must:

- Have a mechanism to install or associate a private key/certificate pair. The mechanism may be part of the client implementation, or a separate procedure.
- Be able to access its private key in order to decrypt content keys stored in vouchers.
- Be able to build a PKCS#7 certificate chain from its end-entity certificate up to the EBX Root CA.
- Send the PKCS#7 certificate chain along with a signed nonce during the authentication step of the EBX protocol (See section \*\*\* for the format.) This implies that the private key corresponding to the end-entity certificate is available to the client software at the time of the EBX transaction.

Each conforming implementation of an EBX server must:

- Accept and decompose PKCS#7 certificate chains sent by EBX clients during the authentication stop of the EBX protocol (See section \*\*\* for the format.)
- Examine each certificate in the chain, and determine:
  - Whether the certificate is a valid X.509 v3 certificate with all extensions required by EBX.

- Whether the constraints imposed by EBX on each certificate type are met.
- Whether the usage constraints imposed by the certificate issuer of each certificate are met.
- Whether the CA that issued the certificate has an associated Validation Authority.
- For each certificate that does have an associated Validation Authority, contact the Validation Authority using OCSP for a determination of the revocation status of the certificate.
- For each certificate that does not have an associated Validation Authority, determine whether the EBX Root CA issued the certificate. If the EBX Root CA issued the certificate, the server must return an error.
- Ignore any non-critical extensions that it cannot handle.
- Return an error if it encounters a critical extension that it cannot handle.
- Extract the end-entity public key from the authenticated end-entity certificate, and encrypt the content key in the voucher with the end-entity public key.

### **3.2 Component Trust Level Certification**

Every software or hardware component of the EBX system that enforces digital rights in a document, including encryption and decryption of the document, must be certified. These components include voucher servers and e-book reading systems.

Components that merely store or transmit encrypted content do not have to be certified. For instance, encrypted book files can be safely stored on and distributed from ordinary file servers, because the necessary decryption keys can be obtained only from EBX voucher servers.

If you envision an “EBX network” made up of nodes that speak the EBX protocols and handle vouchers, those are the components that are certified. A valid EBX certificate not only connotes that the component is authentic and can participate in the EBX network but also conveys the *trust level* of the component. The certificates of these components are checked automatically (“validated”) by software at key points.

Certificates are granted by an impartial organization. An impartial process can also revoke certificates. For example, an investigation might discover that a component (for example, a particular model of reading system) is no longer performing at its designated trust level. In both cases, “impartial” means that the technology vendors do not control the certifying or revoking organization. EBX components *authenticate* one another by requesting and validating certificates. At the higher trust levels, they also check certificates against a *revocation list*, to ensure they have not been revoked.

### 3.2.1 Certification Criteria

A component's certification is valid for a specific trust level. Certification attests that the component protects content according to the requirements of the specified trust level. In general, a certified component meets all the following criteria:

- It correctly interprets general rules of EBX for handling digital rights, which are independent of trust level;
- It implements all content protection mechanisms required by the definition of the specific trust level;
- It correctly interprets and responds to EBX communication protocol;
- The *bona fides* of the manufacturer have been verified by the certifying organization. These would include the manufacturer's compliance with applicable laws, including U.S. export laws.
- The manufacturer's policies and procedures in certain areas, such as certificate granting, have been examined and found satisfactory.

Depending on the trust level, certification may take into account such factors as these:

- Algorithms for such operations as encryption, hashing, and random number generation;
- Length of encryption keys;
- Security of persistent key storage;
- Protection of keys in software, such as by obfuscation;
- Protection of keys in hardware;
- Use of software and hardware for authentication;
- Enforcement of revocation of authorization.

Certification criteria must also take into account the environments in which components operate. We assume that e-book reading systems, because they are mass-market products, live in a hostile environment. That is, the most malicious attacker could have ready access to any e-book reading system. In this case, trust rests entirely on the security of the hardware and software of the reading system.

In contrast, voucher servers (excluding client-based voucher servers) usually live in a friendlier environment. That is, malicious attackers are assumed to have limited access to servers. Trust thus rests partly on the provision of a secure environment for a server. Certification for a server can rest on the manufacturer's support of and recommendation of:

- Use of industry-standard procedures for protecting servers on the Internet;

- Control of network access to the server within the organization;
- Control of physical access to the server;
- Implementation of tamper-resistant and/or secure storage for critical data.

Certification is valid for a specific version of the component. Significant revisions of the hardware or software may require recertification. The manufacturer determines the need for recertification. The manufacturer bears the costs of certification, as well as the costs of revocations and recalls, so we feel the manufacturer has sufficient incentive to recertify when appropriate.

### **3.2.2 Certification Methods**

An entity that performs certification must use industry-standard methods of testing software and hardware components to verify that a component meet requirements for certification. Specific testing methods depend on the requirements of the trust level for which the component is being certified.

Methods can include, among others:

- Interviews with the vendor and inspection of design documentation;
  - Manufacturers may require the certifier to sign nondisclosure agreements, to protect trade secrets and other intellectual property.
  - Refusal to provide design documentation or information even under nondisclosure would be grounds for denial of a particular trust level or revocation of a certificate.
- Inspection of source code, logic diagrams, and other source material;
- Functional and operational tests of the component;
- Disassembly of hardware or software;
- Logic analysis and other tests of hardware;
- Attempted attacks on the component.

An entity certifying an e-book reading system must test a representative production sample of the hardware or software. An entity certifying a voucher server may, depending on the requirements of the trust level, test the documented installation recommendations and requirements of the server in addition to its hardware or software. Methods for testing the environment can include, among others:

- Inspection of documentation for policy and personnel procedures;

- Inspection of the server's hardware, software, and network configuration.

The certifier is required to write a report, without compromising any nondisclosure agreements, that summarizes the findings and methods used in the process. This report will be available to the EBX membership in order to guide future specification and certification efforts.

### **3.2.3 Certification Entities**

The Executive Committee authorizes specific entities to certify EBX components. Certifying entities in general must be competent to employ the certification methods discussed in this section.

Certification costs will be the responsibility of product manufacturers. However, certification organizations should be independent of these manufacturers and worthy of trust by copyright holders. “Big 5” accounting/consulting firms are likely candidates. More specialized consulting organizations such as SAIC (Global Integrity) would also be likely candidates.

### **3.2.4 Reviews**

Manufacturers will be reviewed annually by the certifying organization, to provide feedback to the EBX Working Group.

## 4 Transfer Protocol

It is critical that the distribution and transfer protocols and the format of Vouchers and Credentials be standardized to ensure interoperability between publishers, distributors, booksellers, libraries, and consumers.

Table 1 shows the possible transfer relationships between roles in the EBX system. The table summarizes the status of transfer relationships, using the following notation:

- **In scope** – The protocol for the transfer is within the scope of this version of this specification.
- **Deferred** – The protocol for the transfer may be within the scope of this specification, but consideration of possible specification is deferred to a future version.
- **Out of scope** – The protocol for the transfer is outside the scope of this specification.

All relationships are potentially bilateral. For example, a consumer can potentially both receive a voucher from a voucher distributor and transfer a voucher to a voucher distributor. In this version of the specification, only transfer relationships as shown in Table 1 have been considered. In general, the following kinds of transfers are within the scope of this version:

- Transfers of vouchers and e-book content from publishers to distributors to consumers.
- Transfers of vouchers and e-book content from one consumer to another.

**Table 1.** Specification levels of e-book transfers between roles.

To --> ^^	Publisher	Voucher Distributor	Content Distributor	Bookseller	Library	Consumer
<b>Publisher</b>		In scope	Out of scope			
<b>Voucher Distributor</b>				Deferred	Deferred	In scope
<b>Content Distributor</b>				Out of scope	Deferred	In scope
<b>Bookseller</b>					Deferred	Out of scope
<b>Library</b>						Deferred
<b>Consumer</b>						In scope

#### **4.1 Transfer Protocol and Transport Protocols**

EBX defines a protocol for transferring e-books from one entity to another. This transfer protocol contains requirements for the sequencing and content of data transfer between entities. It does not specify an underlying transport protocol, but instead demands that the transport protocol reliably exchange data as required by the transfer protocol.

When used over the Internet, EBX is commonly implemented using an extended form of HTTP 1.1 / RFC2068 and RFC2069 [HTTP1.1]. HTTP is a good solution for transporting e-books because:

- It passes through network firewalls without trouble (a new protocol / port would probably require firewall configuration changes everywhere).
- There are many high-level APIs already available to program it.
- It makes adding EBX service to an existing Web server easier (such as by using servlets or Active Server Pages).

EBX over HTTP can also enable two consumers to give or lend e-books using a short-range infrared (IR) link. In other words, one consumer can cordlessly *beam* an e-book from his/her reading device to another consumer's e-book reading device.

EBX/HTTP over an IR link can use an IrDA connection [IRDA95]. EBX/HTTP can be used over an IR link between consumer e-book reading devices and e-book personal library servers (e.g., home PCs). An IR link can also be used by in-store e-book kiosks to sell books to consumers.

When used to transfer e-books to or from wireless devices, EBX can use the Wireless Application Protocol (WAP).

The description of transactions in the transfer protocol below specifies the requirements for sequencing and content of data transfer. Implementation notes describe requirements and recommendations for using EBX over HTTP. In this context, *client* means the HTTP client, and *server* means the HTTP server.

#### **4.2 Domains of Trust**

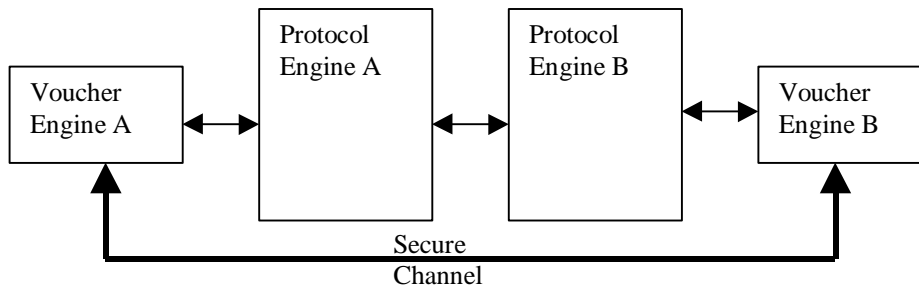
[**Editorial note.** This section has not yet been edited for Version 0.8. It may be obsolete.]

The transaction model in EBXTP involves two distinct domains of trust: protocol engine and voucher engine. The protocol engine runs on the unprotected central processor of client and server computers, sends and receives EBXTP commands, parses the commands, and performs the commands. The voucher engine runs on a, possibly, physically protected processor of the client and server computers, authenticates other voucher engines, sends and receives EBX vouchers and credentials, stores vouchers in protected memory, and performs voucher and content key operations.



In a consumer computer like an e-book reading device, the protocol engine runs on the main processor and memory and the entire voucher engine runs on a smart card. In a commercial computer like a publisher, bookseller or library Web server, the main processor and memory are assumed to be physically protected from consumers and therefore, most of the voucher engine runs on the main processor. Only the actual private key operations of the voucher engine in a commercial server are performed by a smart card.

Protocol engines, while generally considered “trustworthy”, are not sufficiently trusted to handle voucher operations. From the perspective of a voucher engine, protocol engines exist simply to transfer vouchers between voucher engines. (Of course, from the perspective of the consumer, protocol engines primarily exist to transfer e-book content and the voucher exchange is just some copyright “stuff”.)



These separate domains of trust are important concepts when analyzing and implementing EBX/HTTP. Put bluntly, since the protocol engine runs on an unprotected processor, it cannot be trusted to do anything with encrypted vouchers except transfer them. The voucher engine should always be coded to assume the protocol engine is vulnerable to malicious failures.

### 4.3 Example Operation – E-book Purchase

This section describes how a consumer experiences the purchase and download process involving the Protocol Engine. The process description up to the point where the consumer’s e-book reading system becomes involved is for illustration purposes only. Those parts of the process are not explicitly specified herein and are beyond the scope of this specification.

**Step 1: Consumer accesses Web site.** A consumer accesses the publisher/bookseller Web site via a browser. The consumer selects one or more e-books to purchase.

**Step 2: Purchase info submitted.** Regardless of the specific design of the site, at some point the consumer will have accumulated one or more content identifiers for the e-books the consumer is purchasing. The consumer must also (somewhere along the way) supply identifying information (e.g., name) and payment information (e.g., credit card number), plus any additional market research info the seller wants to collect. This information is transmitted to the Web server (usually as an HTML form, but it could be done other ways too).

**Step 3: Payment authorization.** Before beginning the EBX download process, the bookseller Web site server switches to an SSL session in order to validate the consumer's payment information. Presumably the credit card information is submitted to an authorization facility over the Internet, and an authorization code is returned.

**Step 4: Create purchase record.** At this point, the server enters a purchase order record in an order database at the Voucher Distributor's Web site. At a minimum the purchase order record should include the following: unique order ID, list of e-book content identifiers, and a fulfillment status. The initial value of the fulfillment status field is AUTHORIZED.

**Step 5: Return fulfillment URL.** After the purchase record has been created at the Voucher Distributor, the Voucher Distributor server generates a URL that may be used to fulfill the purchase record just created.

**Step 6: Present fulfillment URL link in browser.** The result page of the Submit operation (Step 2) is for the bookseller Web site to present the "Order Confirmation" or "Thank You" page to the consumer. Typically this page displays a message like "Purchase authorized", and it presents the fulfillment URL as a link labeled "Download E-books".

Everything to this point is outside the scope of the Protocol Engine itself.

**Step 7: User clicks fulfillment URL, fulfillment instructions are returned, vouchers are downloaded, e-books are downloaded.** This is the domain of the Protocol Engine, and is the subject of the rest of this chapter.

#### **4.4 Notational Conventions and Generic Grammar**

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by HTTP 1.1 / RFC2068 [HTTP1.1]. Implementers need to be familiar with RFC2068, RFC2069, and the notation in order to understand this specification.

#### **4.5 HTTP Implementation Note: EBX HTTP Request**

[**Editorial note.** This section needs to specify query values for GET requests.]

In RFC2068, the generalized HTTP request syntax is defined as:

```

Request      = Request-Line
              * ( General-Header
                | Request-Header
                | Entity-Header )
              CRLF
              [ Entity-Body ]
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
Method      = "OPTIONS"
            | "GET"
            | "HEAD"
            | "POST"
            | "PUT"

```

```

"DELETE"
"TRACE"
Request-URI = "*" | absoluteURI | abs_path
(See RFC2068 for further syntax.)

```

The Request-URI in RFC2068 can include a `rel_path` which includes a query defined as follows:

```

rel_path   = [ path ] [ ";" params ] [ "?" query ]
query      = * ( uchar | reserved )

```

In EBX, the query for GET HTTP requests is defined as follows:

```

query      = "action=" get_action * ( "&" query_param)
get_action = "purchase" | "borrow" | "handoff"
query_param = query_key "=" query_value
query_key   = "orderid" | "ack" | "bookid"

```

In EBX, the query for POST HTTP requests is defined as follows:

```

query      = "action=" post_action * ( "&" *query_param)
post_action = "return" | "lend" | "give"

```

All requests must also contain the following HTTP header field:

**Host** – Required by HTTP 1.1.

## 4.6 HTTP Implementation Note: EBX HTTP Header Extensions

EBX uses several extensions to the HTTP headers for various requests and responses. These are described below.

### 4.6.1 EBX-Action Header

TBS

### 4.6.2 EBX-Version Header

[**Editorial note.** This section has not yet been edited for Version 0.8. It may be obsolete.]

The EBX-Version header is provided so that clients and servers can identify the level to which their implementation complies with this specification.

```

EBX-Version = "x-EBX-Version" ":" 1*DIGIT "." 1*DIGIT

```

The actual header is as follows:

```

x-EBX-Version: 0.8

```

### 4.6.3 EBX Challenge-Response Headers

[**Editorial note.** This section has not yet been edited for Version 0.8. It may be obsolete.]

EBX uses an abbreviated implementation of Section 11, “Access Authentication”, from RFC2068. This section in RFC 2068 describes a simple challenge-response authentication mechanism that may be used between clients and servers. EBX uses this scheme to allow voucher engines on each side of a transaction to authenticate themselves in order to move vouchers from one voucher engine to another.

The generalized HTTP syntax from RFC 2068 Section 11 for “Access Authentication” used by EBX is:

```
auth-scheme    = token
auth-param     = token "=" quoted-string
challenge      = auth-scheme 1*SP auth-param
credentials    = auth-scheme #auth-param
```

#### 4.6.3.1 EBX-Authenticate Header

The response used by an origin server to challenge the authorization of a user agent includes an Authenticate header field containing at least one challenge applicable to the requested resource.

These are the specific EBX settings to the generalized syntax:

```
auth-scheme    = EBX
auth-param     = nonce="base64_encoded_NONCE"
challenge      = EBX nonce="base64_encoded_NONCE"
```

EBX HTTP Header (Authenticate header with challenge):

```
x-EBX-Authenticate: EBX nonce="base64_encoded_NONCE"
```

#### 4.6.3.1.1 Implied EBX-Authenticate Header

In EBX purchase and borrow scenarios, the nonce is contained in the fulfillment instructions. Therefore the EBX-Authenticate header is not transmitted to clients when receiving fulfillment instructions.

#### 4.6.3.2 EBX-Authorization Header

Continuing, using RFC 2068 Section 11 for reference, a user agent that wishes to authenticate itself with a server includes an Authorization header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent.

These are the specific EBX settings to the generalized syntax:

```
auth-scheme    = EBX
auth-param     = credentials="base64_encoded_credentials"
credentials    = EBX credentials="base64_encoded_credentials"
```

EBX HTTP Header (Authorization header with credentials):

```
x-EBX-Authorization: EBX credentials="base64_encoded_credentials"
```

#### 4.6.3.3 EBX-Authenticationinfo Header

In EBX, after credentials have been issued, vouchers are returned. The header defined for the voucher is:

EBX HTTP Header (Authenticationinfo header with voucher):

```
x-EBX-Authenticationinfo: voucher="base64_encoded_voucher"
```

### 4.7 Receiving Fulfillment Instructions (handoff)

The process of transferring vouchers and e-books from a voucher distributor to an EBX e-book reading system begins with the reading system's receiving fulfillment instructions from the voucher server that has been authorized to fulfill the order for the consumer. Typically, this is the result of an HTTPS request from a browser against the fulfillment URL presented at the end of a purchase transaction by the consumer with a bookseller's Web site.

#### 4.7.1 Handoff Request to Voucher Server

The handoff request is issued by a Web browser client to the voucher server and specifies the order ID to be handed off from the voucher server to the EBX e-book reading system. The order ID must reliably identify to the voucher server the e-books to be transferred but is otherwise unspecified. The voucher server responds with the XML fulfillment instructions for the protocol engine in an EBX reading system to use to obtain the vouchers for the order.

##### 4.7.1.1 HTTP Implementation Note

The handoff request can use either a GET or a POST method. The request must supply the **action** parameter, whose value is **handoff**, and the **orderid** parameter, whose value is a string. For example:

```
GET /EBX?action=handoff&orderid=6789346 HTTP/1.1
Host: www.acme.com
```

#### 4.7.2 Voucher Server Processing of Handoff Request

Handoff requests are typically executed from client Web browsers. In a successful response, the Content-Type header is set to **application/x-ebx** to activate an EBX e-book reading system. In an error response, the Content-Type header is set to **text/html** and the user typically sees the error response in the browser window. The EBX e-book reading system is not activated on error responses.

- If the order is marked as ACKNOWLEDGED, return HTTP 410 (Gone, Order delivered and ACKed)
- If the order is marked as FULFILLED, return HTTP 410 (Gone, Order delivered)

- If other errors occur while processing the request, return one of the other HTTP error codes as specified earlier in this section.
- If the order is marked as AUTHORIZED, proceed. The voucher server uses the voucher engine to generate and record the NONCE for the response.

#### 4.7.3 Handoff Response from Voucher Server (XML fulfillment instructions)

The voucher server responds to the handoff request by returning fulfillment instructions that the protocol engine in the EBX reading system uses to obtain the vouchers and books for the order. As a result of this response, the reading system must receive the fulfillment instructions.

Fulfillment instructions are in XML format as specified in the [TBS] section. Following is an example:

```
<?xml version='1.0'?>
<ebx:transferData ebx:version="0.8">
  <nonce>lbJkMjTeUz1OL9tXaYhNSnR42iIq</nonce>
  <baseURL>http://localhost/fulfill/ebx</baseURL>
  <availableAuthenticationSchemes>
    <ebx:authenticationScheme
      ebx:encryptionAlgorithmID="tueh87yRHiuy975hfh4rd"
      ebx:encryptionAlgorithmName="RSA"
      ebx:signatureAlgorithmID="n457skYh29H8erg"
      ebx:signatureAlgorithmName="RSA-SHA1">pki-rsa-sha1
    </ebx:authenticationScheme>
    <ebx:authenticationScheme>
      ebx:encryptionAlgorithm="tueh87yRHiuy975hfh4rd"
      ebx:encryptionAlgorithmName="RSA"
      ebx:signatureAlgorithm="n457skYh29H8erg"
      ebx:signatureAlgorithmName="RSA-MD5">pki-rsa-md5
    </ebx:authenticationScheme>
  </availableAuthenticationSchemes>
  <request>
    <action>purchase</action>
    <orderID>114816072033825</orderID>
  </request>
  <entry>
    <bookID>ISBN:444444444</bookID>
    <availableTypes>
      <dc:format
        dcq:scheme="IMT"
        ebx:encryptionMethod="pdfcrypt"
        ebx:encryptionAlgorithmID="57YTH28vq7Uuedh3"
        ebx:encryptionAlgorithmName="DES"
        ebx:encryptionKeyLength="56">
        application/pdf
      </dc:format>
      <dc:format
        dcq:scheme="IMT"
        ebx:encryptionMethod="zipencode"
        ebx:encryptionAlgorithmID="57YTH28vq7Uuedh3"
        ebx:encryptionAlgorithmName="DES"
        ebx:encryptionKeyLength="56">
```

```

                text/x-oeb1-document
            </dc:format>
        </availableTypes>
    </entry>
</ebx:transferData>

```

The **baseURL** element is the URL that the protocol engine uses to get the vouchers (and, subsequently, e-books) for this order. The **nonce** element is given to the voucher engine in the EBX reading system to create the credentials for the voucher request.

#### 4.7.3.1 HTTP Implementation Note

The Content-Type header value in the HTTP response is set to **application/x-ebx**. The Content-Disposition header is set to **inline; filename="ebx.etd"**. This assures that the file is saved according to the MIME registration settings for application/x-ebx.

An EBX reading system running on Microsoft Windows platforms should register itself as the MIME handler for application/x-ebx so that it will be activated by the browser client on receipt of this HTTP response.

In this scenario, the browser typically saves the data from the response (the XML fulfillment instructions) in a temporary file on the hard disk. The application that is registered as the MIME handler is then activated with the path to the file on the command line.

For example:

```

HTTP/1.1 200 OK
Content-Type: application/x-ebx
Content-Disposition: inline; filename="ebx.etd"
Content-Length: 789

[789 bytes of XML fulfillment instructions]

```

#### 4.7.3.2 Windows Registration Implementation Note

These are the Windows Registry settings required to map a reading system to handle the content type of application/x-ebx.

WINDOWS REGISTRY KEY	Value Name	Value Data
<b>HKEY_CLASSES_ROOT</b>		
.etd	(Default)	EBXTransfer
	Content Type	application/x-ebx
MIME\Database\Content Type\application/x-ebx	(Default)	(value not set)
	Extension	.etd
EBXTransfer	(Default)	EBX Transfer Data File
	EditFlags	00 00 01 00
EBXTransfer\shell\open	(Default)	(value not set)

	EditFlags	01 00 00 00
EBXTransfer\shell\open\command	(Default)	"readerapp.exe" "%1"

#### 4.7.4 Client Processing of Handoff Response from Voucher Server (XML fulfillment instructions)

The EBX client, after receiving fulfillment instructions may use them to obtain vouchers and content. The client should store the instructions in a safe place so that they may be replayed if there are errors while obtaining vouchers. Once the fulfillment instructions have been successfully executed, they can be discarded.

### 4.8 Getting Voucher(s)

Transfer of a book from the server to the client begins when the client sends the voucher server a voucher request using the URL provided in the **baseURL** element of the fulfillment instructions.

The voucher server responds with the voucher(s) for the order requested. The client responds with an “ACK” to the voucher server to signify that it has received the voucher(s), then it gets the e-book(s) by examining the voucher(s) it receives.

#### 4.8.1 Voucher Request to Voucher Server (purchase, borrow)

The Issue voucher request is typically used when an EBX e-book reading system wishes to retrieve a book from a voucher server after acquiring the rights to a book from a bookseller or a library. It is also used by a distributor, bookseller, or library to purchase one or more books from a publisher, distributor, or bookseller. The action for the request is either “purchase” or “borrow”.

The additional parameter “orderid” is the purchase order identifier to be used by the voucher server (typically as a database key) to retrieve the vouchers for the order.

The reading system makes a request for vouchers to the voucher server identified by the **baseURL** element in the fulfillment instructions. The request contains data in XML format containing two objects: a Credentials object, issued by the reading system’s voucher engine, and a Voucher Request object. Following is an example:

```
<?xml version='1.0'?>
<ebx:credentials ebx:version="0.8">
  <ebx:authenticationScheme
    ebx:encryptionAlgorithmID="tueh87yRHiuy975hfh4rd"
    ebx:encryptionAlgorithmName="RSA"
    ebx:signatureAlgorithmID="n457skYh29H8erg"
    ebx:signatureAlgorithmName="RSA-SHA1">pki-rsa-sha1
  </ebx:authenticationScheme>
  <nonce>lbJkMjTeUz1OL9tXaYhNSnr42iIq</nonce>
  <signedData>MIIDwQYJKoZIhvcNAQcCoIIDsjCCA64CAQExCzAJBgUrDgMCGGUAM
AsGCSqGSIB3DQEHAaCCAnQwggJwMIIB2aADAgECAgEAMA0GCSqGSIB3DQEBBQUAMGoxaDAJ
BgnVBAYTA1VTMBQGA1UECBMNTWFzc2FjaHVzZXR0czAhBgNVBAsTG1RFU1QgQ0VSVE1GSUN
BVEUGQVVUSE9SSVRZMCIGA1UEChMbr2xhc3Nib29rLCBJbmMuIC0gVEVTVCBPTkxZMB4XDT
AwMDCwOTIwMjYzOFoXDTEwMDCwODIwMjYzOFowaJFoMAkGA1UEBhMCVVMwFAYDVoQIIEw1NY
```



```

XNzYWNodXNldHRzMCEGA1UECxmAVEVTVCBDRVJUSUZJQ0FURSBVVVRIT1JVVfkwIgyYDVQOK
ExtHbGFzc2Jvb2ssIEluYy4gLSBURVNUIE9OTfkwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIG
JAoGBANPWjIW6aw8yE9PwLD+IevFzQ4aONTzI8ptNUSlauKzNWhh8Q1YXYp5eiSgQsS6y19
zvXodl7tgQxw5Wc3uvRs jTlnU5LRqIALj+6z3BHoVVbUoUX6QUoVWFfZz575XXHZ5c8DxeC
wVQJWNWSZ9qJA4HgW6BvNUczVC/aYQaS6mDAGMBAAGjJjAkMBIGA1UdEwEB/wQIMAYBAf8C
AQAwDgYDVR0iAQH/BAQwAjaAMA0GCSqGSIb3DQEBAQUAA4GBAHpAL2BuMHbPRgfsPg/SZDU
sN81qnxAZLtgzkd5hzYw68rCJfL8MZWGCBaTm7M60Cza95HYoeiFT68Sg85ZehcmBjNebZy
pt4tYCh5A4uqjOuUIBzXzZcbCA8vouCPkUPcTD1jJ0hrJc/SZZUe5wke2GWjn1t87PM0Z5J
bJzCzQxMYIBFTCCARECAQEwbzBqMWgwCQYDVQQGEwJVUzAUBgNVBAGTDU1hc3NhY2h1c2V0
dHMwIQYDVQQLExpURVNUIEFUF1RJRk1DQVRFIEFVVEhPUklUWTAiBgNVBAoTG0dsYXNzYm9
vaywgSW5jLiAtIFRFU1QgT05MWQIBADAJBgUrDgMCGGUAMA0GCSqGSIb3DQEBAQUABIGaky
w6vEziW8aPiWJ/BPFCIAxRnFnYRy4xXYruM5myTgeygAff//CbaFN92pXOE/48Mf987PRbE
COvJxQ1kY2M6bHQq7r7CDEx8VmYEXjf6v6IlCaEmESmTyPzUwbKzhtuBM8VHG6x1Rm/GgPc
dUDuAol0///lcujXJQ6+U/w/s9w=
  </signedData>
</ebx:credentials>
<ebx:voucherRequest ebx:version="0.8">
  <action>purchase</action>
  <orderID>114816072033825</orderID>
  <preferredTypes>
    <dc:format
      dcq:scheme="IMT"
      ebx:encryptionMethod="zipencode"
      ebx:encryptionAlgorithmName="DES"
      ebx:encryptionKeyLength="56">
      text/x-oeb1-document
    </dc:format>
    <dc:format
      dcq:scheme="IMT"
      ebx:encryptionMethod="pdfcrypt"
      ebx:encryptionAlgorithmID="57YTH28vq7Uuedh3"
      ebx:encryptionAlgorithmName="DES"
      ebx:encryptionKeyLength="56">
      application/pdf
    </dc:format>
  </preferredTypes>
  <entry>
    <bookID>ISBN:444444444</bookID>
    <preferredTypes>
      <dc:format
        dcq:scheme="IMT"
        ebx:encryptionMethod="zipencode"
        ebx:encryptionAlgorithmID="57YTH28vq7Uuedh3"
        ebx:encryptionAlgorithmName="DES"
        ebx:encryptionKeyLength="56">
        text/x-oeb1-document
      </dc:format>
    </preferredTypes>
  </entry>
</ebx:voucherRequest>

```

The **orderID** in the **voucherRequest** is the order ID identified by the **orderID** element in the fulfillment instructions. The high-level **preferredTypes** element is required and tells what content types the reading system accepts, in order of preference. The **entry** elements of the **voucherRequest** are optional. It is used when the reading system wishes to inform the voucher

server of the preferred content types for a specific work. If present, each **entry** in the **voucherRequest** corresponds to an **entry** in the fulfillment instructions. The **preferredTypes** element for each entry contains a list of content types that the reading system can accept for the specific work, in order of preference.

#### 4.8.1.1 HTTP Implementation Note

The x-EBX-Action header is used in the HTTP POST request.

Two examples, one with the x-EBX-Action header set empty, and one using the x-EBX-Action header.

```
POST /EBX?action=purchase&orderid=114816072033825 HTTP/1.1
Host: www.acme.com
x-EBX-Action: ""
Content-Length: 789
```

[789 bytes of XML Request Object]

```
POST /EBX HTTP/1.1
Host: www.acme.com
x-EBX-Action: "purchase"
```

```
Content-Length: 789
```

[789 bytes of XML Request Object]

#### 4.8.2 Voucher Server Processing of Voucher Request (purchase, borrow)

For purchase or borrow, the voucher server should perform these validations:

- Verify that the EBX version supported by the client is supported by the server. If not, return statusCode of 412 (Precondition Failed, Unsupported EBX Version).
- Verify that the EBX authentication scheme supported by the client is supported by the server. If not, return statusCode of 412 (Precondition Failed, Unsupported EBX authentication scheme.)
- Validate the purchase order number against its local database. If not valid, return statusCode of 404 (Not found, Order number not found).
- If the order is marked as ACKNOWLEDGED, return statusCode of 410 (Gone, Order Delivered and Acknowledged)
- If the order is marked as FULFILLED, mark this as possible abuse of the EBX protocol. This could also be a subsequent request from a client if the initial response sent by the voucher server never arrived at the client, or the client encountered errors saving the vouchers to persistent storage. If the voucher server has rules in place to enforce a limit on the number of times an order or voucher may be fulfilled before rejecting a request, and this request exceeds

the limit, then return statusCode of 403 (Forbidden, Multiple fulfillment count exceeded), otherwise proceed.

- If the order is marked as AUTHORIZED, proceed.
- **For each** book in the order, perform the following operations:

The **<preferredTypes>** of the **<voucherRequest>** must be examined for each book in the order. The correct **<preferredTypes>** is obtained from the **<entry>** element of the voucherRequest, if it exists, or from the high-level default **<preferredTypes>** element. If the voucher server cannot deliver any of the formats specified for the specific book in the order, then statusCode 406 (Not Acceptable) is returned in the **<entry>** element in the response. The voucher is not issued.

If other errors occur while processing the request, set statusCode in the **<entry>** element in the response to an appropriate value. The voucher is not issued.

If the voucher is issued, the fulfillment status field in the database for the voucher is set to FULFILLED.

- If all of the vouchers are issued, the value in the statusCode of the response is set to 200 (OK). The entry tags are not required.
- If none of the vouchers could be issued, return statusCode of 204 (No Content, No vouchers could be issued). There should be an entry tag for every voucher that could not be issued with the detailed statusCode for the failure.
- If some, but not all of the vouchers could be issued, return statusCode of 206 (Partial Content, Some vouchers could not be issued). There should be an entry tag for every voucher that could not be issued with the detailed statusCode for the failure.

### 4.8.3 Response from Voucher Server (purchase, borrow)

The voucher server responds with an XML object containing the status of the operation and, if successful, the voucher(s) for the particular order. The returned voucher(s) have the content-key encrypted with the public key of the client, which was obtained from the credentials sent on the request. Metadata for the particular books, including the URL for the encrypted content files, is contained in the voucher(s) that are received.

The XML status object is specified in the [TBS] section.

The XML format of vouchers is specified in the *Voucher Format* section on page 85. Following is an example:

```
<?xml version="1.0" encoding='UTF-8' ?>
<ebx:status ebx:version="0.8">
  <statusCode>...</statusCode>
  <statusMessage>...</statusMessage>
  <statusMessage>...</statusMessage>
  <entry>
    <bookID>ISBN:4444444444</bookID>
```

```

        <statusCode>...</statusCode>
        <statusMessage>...</statusMessage>
        <statusMessage>...</statusMessage>
    </entry>
</ebx:status>
<ebx:vouchers ebx:version="0.8">
    <ebx:voucher ebx:version="0.8">
        Please see Voucher section for details
    </ebx:voucher>
    <ebx:voucher ebx:version="0.8">
        Please see Voucher section for details
    </ebx:voucher>
</ebx:vouchers>

```

#### 4.8.3.1 HTTP Implementation Note

The body of the message is the XML object.

Example:

```

HTTP/1.1 200 OK
Content-Type: text/XML
Content-Length: 1234

[1234 bytes of XML voucher file]

```

#### 4.8.4 Client Processing of Response from Voucher Server for Voucher Request (purchase, borrow)

- If the statusCode of the response is 200 (OK), the vouchers are extracted from the response, and saved to persistent storage, then the client issues the ACK request. Additionally, if the EBX e-book reading system should delete the fulfillment instructions XML file if it had possession or ownership of the file.
- If the statusCode of the response is 410 (Gone), then the voucher(s) for this order have already been delivered and acknowledged. This is not a “hard failure”, and the EBX e-book reading system should delete the fulfillment instructions XML file if it had possession or ownership of the file. The fulfillment instructions file has already been executed and acknowledged.
- If the statusCode of the response is 206 (Partial Content), some, but not all the vouchers were delivered. The delivered vouchers are extracted from the response, and saved to persistent storage, then the client issues the ACK request, with the exact set of vouchers that were obtained. The statusCode of the failed entries may be examined to present feedback to the user.
- If the statusCode of the response is 204 (No Content), none of the vouchers were delivered. The statusCode of the failed entries may be examined to present feedback to the user.

- If the statusCode in the response is some error other than above, then none of the vouchers are delivered, and the EBX e-book reading system should preserve the fulfillment instructions XML file on disk so that it may be replayed at a future time.
- If errors occur within the EBX e-book reading system while processing a response with status code of 200 OK, and prior to saving the voucher(s) from the response in persistent storage, the EBX e-book reading system should preserve the fulfillment instructions XML file on disk so that it may be replayed at a future time. Additionally, the error condition may be presented to the user and/or logged.

Detailed error messages may be obtained from the <statusMessage> </statusMessage> tags of the response for presentation to the user or for logging.

Successfully received vouchers are passed along to the voucher engine. The voucher engine must commit the voucher(s) to persistent storage at this point. The protocol engine and voucher engine must be integrated in such a way that the protocol engine is able to obtain relevant metadata from the vouchers for each voucher that is received. At a minimum, the protocol engine needs to be able to obtain the URL for the encrypted content file from the voucher.

A client may replay a fulfillment request if it does not receive a successful statusCode in the response or if it fails while committing the voucher(s) to persistent storage.

#### 4.8.5 Acknowledgment Request to Voucher Server (ACK)

For fail-safe error recovery, the client must make a request to the voucher server to acknowledge that it received the vouchers successfully and added them to persistent storage. The client must make an acknowledgment request if it has successfully received and stored any vouchers. The request contains data in XML format as specified in the [TBS] section. The XML data includes the same action and orderID elements as the previous voucherRequest object that was sent to the voucher server. The entry element for vouchers received is not required. If the entry element is omitted, it is assumed that all vouchers from the previous request were received. If the client wishes to add every voucher received in the entry tag, it may. However, if only some of the vouchers could be stored, then in this case the successfully stored vouchers are passed as entry elements. Following is an example:

```
<voucherAck ebx:version="0.8">
  <action>purchase</action>
  <orderID>114816072033825</orderID>
  <entry>
    <bookID>ISBN:4444444444</bookID>
  </entry>
</voucherAck>
```

##### 4.8.5.1 HTTP Implementation Note

The x-EBX-Action header is used in the HTTP POST request.

Two examples, one with the x-EBX-Action header set empty, and one using the x-EBX-Action header. Note that to form the URI for the request with the empty x-EBX-Action header, the

client adds the query parameter “ack=true” to the original request that was issued to obtain the voucher(s).

```
POST /EBX?action=purchase&orderid=114816072033825&ack=true HTTP/1.1
Host: www.acme.com
x-EBX-Action: ""
Content-Length: 789
```

[789 bytes of XML Request Object]

```
POST /EBX HTTP/1.1
Host: www.acme.com
x-EBX-Action: "purchase"
```

Content-Length: 789

[789 bytes of XML Request Object]

#### 4.8.6 Voucher Server Processing of Acknowledgment Request (ACK)

When the voucher server receives the ACK request, it performs the following action:

- If the request contains entries, each database entry corresponding to the entry in the request is changed from FULFILLED to ACKNOWLEDGED.
- If there is no entry in the request, all database entries corresponding to the order in the request are changed from FULFILLED to ACKNOWLEDGED.
- If any of the entries are already set to ACKNOWLEDGED in the database, no action is needed.
- If errors occur while processing the request, or if any of the entries are marked AUTHORIZED, Set the statusCode element of the response to an appropriate value.

#### 4.8.7 Acknowledgment Response from Voucher Server (ACK)

The voucher server returns a successful response after receiving an acknowledgment request.

```
<?xml version="1.0">
<ebx:status ebx:version="0.8">
  <statusCode>200</statusCode>
</ebx:status>
```

##### 4.8.7.1 HTTP Implementation Note

The voucher server returns success after receiving the ACK.

Example:

```
HTTP/1.1 200 OK
Content-Type: text/XML
Content-Length: 75
```

[75 bytes of XML data]

#### **4.8.8 Client Processing of Acknowledgment Response from Voucher Server (ACK)**

The successful GET response for ACK is the statusCode of 200 (OK) being returned by the voucher server. Failed responses may be safely ignored by the EBX e-book reading system

- If the statusCode in the response is 200 (OK), the voucher server has received and processed the ACK request. The client does not need to take any further action.
- If the statusCode in the response is some code other than 200 (OK), the voucher server may not have received and processed the ACK request. The client does not need to take any further action. It is not required to receive a successful response to the ACK request. Clients must not take advantage of the fact that ACK was not delivered successfully, and retransmit a request to fulfill the order that was not acknowledged.

#### **4.9 Getting Encrypted E-book(s)**

In order to obtain an encrypted e-book, a reading system must use its voucher engine to extract the content location from the e-book's voucher, and it must then get the e-book content from that location.

E-books may have a variety of locations, including Web sites, CD-ROMs or DVD-ROMs, and network file systems. How the reading system obtains e-book content is outside the scope of this specification. This section describes a possible request and response for obtaining content using HTTP.

Once it has obtained the e-book content, the reading system maintains a correspondence between the voucher and the local copy of the e-book, if any. The mechanism of this correspondence is outside the scope of this specification.

##### **4.9.1 Content Request to Content Server (purchase, borrow)**

After the ACK for voucher receipt has been completed, the client's protocol engine uses the content file URL from within each voucher that it obtained to get the encrypted content files. Often, these are simple HTTP or FILE URLs. The encrypted content files typically reside on high-bandwidth file servers on the Internet.

###### **4.9.1.1 HTTP Implementation Note**

Example:

```
GET www.bookserver.com/ebooks/enc_file.pdf HTTP/1.1
Host: www.acme.com
```

##### **4.9.2 Content Response from Content Server (purchase, borrow)**

The content server issues the file.

#### 4.9.2.1 HTTP Implementation Note

Example:

```
HTTP/1.1 200 OK
Content-Type: application/pdf
Content-Length: 123456

[123456 bytes encrypted content file]
```

### 4.10 Giving or Lending a Book (consumer to consumer)

This section details the protocol for a peer-to-peer transfer of an e-book from one e-book reading system to another. This protocol is used when one reading system gives or lends an e-book to another reading system. It is approximately the mirror image of the protocol for purchasing an e-book. In peer-to-peer transfer, the reading system that owns the e-book acts as the client and initiates all requests. However, the data that the e-book owner transmits is similar to the data that the server transmits for the purchase protocol. The receiver of the e-book acts as the server and responds to requests. However, the data that the receiver transmits is similar to the data that the client transmits for the purchase protocol. Furthermore, even though it acts as the client in initiating requests, the owner of the e-book must authenticate the receiver.

#### 4.10.1 Transfer Request from Owner to Receiver (give, lend)

The owner of the e-book initiates the transfer by issuing a request to the receiver. Before issuing the request, the owner generates a nonce. The request contains XML data that is similar to fulfillment instructions in the purchase protocol, with the following differences:

- The **baseURL** in the transfer data object is absent.
- The **action** in the **request** object is **give** or **lend**.
- The **orderID** in the request object is absent.

Example:

```
<?xml version='1.0'?>
<ebx:transferData ebx:version="0.8">
  <nonce>lbJkMjTeUz1OL9tXaYhNSnR42iIq</nonce>
  <availableAuthenticationSchemes>
    <ebx:authenticationScheme
      ebx:encryptionAlgorithmID="tueh87yRHiuy975hfh4rd"
      ebx:encryptionAlgorithmName="RSA"
      ebx:signatureAlgorithmID="n457skYh29H8erg"
      ebx:signatureAlgorithmName="RSA-SHA1">pki-rsa-sha1
    </ebx:authenticationScheme>
    <ebx:authenticationScheme>
      ebx:encryptionAlgorithm="tueh87yRHiuy975hfh4rd"
      ebx:encryptionAlgorithmName="RSA"
      ebx:signatureAlgorithm="n457skYh29H8erg"
      ebx:signatureAlgorithmName="RSA-MD5">pki-rsa-md5
    </ebx:authenticationScheme>
  </availableAuthenticationSchemes>
  <request>
```



```

        <action>give</action>
    </request>
    <entry>
        <bookID>ISBN:4444444444</bookID>
        <availableTypes>
            <dc:format
                dcq:scheme="IMT"
                ebx:encryptionMethod="pdfcrypt"
                ebx:encryptionAlgorithmName="DES"
                ebx:encryptionAlgorithmExtra="none"
                ebx:encryptionKeyLength="56">
                    application/pdf
            </dc:format>
        </availableTypes>
    </entry>
</ebx:transferData>

```

#### 4.10.1.1 HTTP Implementation Note

The x-EBX-Action header is used in the HTTP POST request.

Two examples, one with the x-EBX-Action header set empty, and one using the x-EBX-Action header.

```

POST /EBX?action=give HTTP/1.1
Host: www.acme.com
x-EBX-Action: ""
Content-Length: 789

[789 bytes of XML Request Object]

```

```

POST /EBX HTTP/1.1
Host: www.acme.com
x-EBX-Action: "give"

Content-Length: 789

[789 bytes of XML Transfer Data Object]

```

#### 4.10.2 Transfer Response from Receiver to Owner (give, lend)

The response by the receiver to the transfer request contains data in XML format containing two objects: a Credentials object, issued by the reading system's voucher engine, and a Voucher Request object. Following is an example:

```

<?xml version='1.0'?>
<ebx:credentials ebx:version="0.8">
    <ebx:authenticationScheme
        ebx:encryptionAlgorithmID="tueh87yRHiuy975hfh4rd"
        ebx:encryptionAlgorithmName="RSA"
    >

```

```

    ebx:signatureAlgorithmID="n457skYh29H8erg"
    ebx:signatureAlgorithmName="RSA-SHA1">pki-rsa-sha1
  </ebx:authenticationScheme>
  <nonce>lbJkMjTeUz1OL9tXaYhNSnr42iIq</nonce>
  <signedData>MIIDwQYJKoZIhvcNAQcCoIIDsjCCA64CAQExCzAJBgUrDgMCGGUAM
AsGCSqGSIb3DQEHAaCCAnQwggJwMIIB2aADAgECAgEAMA0GCSqGSIb3DQEBBQUAMGoxaDAJ
BgNVBAYTAlVTMBQGA1UECBMNTWFzc2FjaHVzZXRX0czAhBgNVBAstGlRFU1QgQ0VSVElGSUN
BVEUgQVVUSE9SSVRZMCIGA1UEChMbr2xhc3Nib29rLCBjbmMuIC0gVEVTVCBPTkxZMB4XDT
AwMDcwOTIwMjYzOFoXDTEwMDcwODIwMjYzOFowaJFoMAkGA1UEBhMCMVVMwFAYDVQQIEw1NY
XNzYWNodXNldHRzMCEGA1UECXMaveVTVCBDRVJUSUZJQ0FURSBVVRIT1JJVfkwIgyYDVQK
ExtHbGFzc2Jvb2ssIEluYy4gLSBURVNUIE90TFkkgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIG
JAoGBANPWjIW6aw8yE9PwLd+IevFzQ4aONTzI8ptNUSlauKzNWhh8QlYXYp5eiSgQsS6y19
zvXodl7tgQxw5Wc3uvRsjtlnU5LRqIALj+6z3BHoVvUoUX6QUoVWffZz575XXHZ5c8DxeC
wVQJWNWSZ9qJA4HgW6BvNUczVC/aYQaS6mDAGMBAAGjJjAKMBIGA1UdEwEB/wQIMAYBAf8C
AQAwDgYDVR0iAQH/BAQwAJAAMA0GCSqGSIb3DQEBBQUAA4GBAHpAL2BuMHbPRgfsPg/SZDU
sN81qnxAZLtgzkd5hzYw68rCjfl8MZWGCBATm7M60Cza95HYoeiFT68Sg85ZehcmBjNebZy
pt4tYCh5A4uqjOuUIBzXzZcbCA8vouCPkUPcTD1jJ0hrJc/SZZUe5wke2GWjnl1t87PM0Z5J
bJzcZQxMYIBFTCCARECAQEwbzBqMWgwCQYDVQQGEwJVUzAUBGNVBAGTUDU1hc3NhY2h1c2V0
dHMwIQYDVQQLExpURVNUIEFUFURJk1DQVRFIEFVVEhPuklUWTAiBgNVBAoTG0dsYXNzYm9
vaywSW5jLiAtIFRFU1QgT05MWQIBADAJBgUrDgMCGGUAMA0GCSqGSIb3DQEBAQUABIGaky
w6vEziW8aPiWJ/BPFCIAxRnFnYRy4xXYruM5myTgeygAff//CbaFN92pXOE/48Mf987PRbE
COvJxQ1kY2M6bHQq7r7CDEx8VmYEXjf6v6IlCaEmESmTyPzUwbKzhtuBM8VHG6xlRm/GgPc
dUDuAol0///lcujXJQ6+U/w/s9w=
  </signedData>
</ebx:credentials>
<ebx:voucherRequest ebx:version="0.8">
  <action>give</action>
  <preferredTypes>
    <dc:format
      dcq:scheme="IMT"
      ebx:encryptionMethod="zipencode"
      ebx:encryptionAlgorithmName="DES"
      ebx:encryptionAlgorithmExtra="none"
      ebx:encryptionKeyLength="56">
      text/x-oeb1-document
    </dc:format>
    <dc:format
      dcq:scheme="IMT"
      ebx:encryptionMethod="pdfcrypt"
      ebx:encryptionAlgorithmName="DES"
      ebx:encryptionAlgorithmExtra="none"
      ebx:encryptionKeyLength="56">
      application/pdf
    </dc:format>
  </preferredTypes>
  <entry>
    <bookID>ISBN:444444444</bookID>
    <preferredTypes>
      <dc:format
        dcq:scheme="IMT"
        ebx:encryptionMethod="zipencode"
        ebx:encryptionAlgorithmName="DES"
        ebx:encryptionAlgorithmExtra="none"
        ebx:encryptionKeyLength="56">
        text/x-oeb1-document
      </dc:format>
    </preferredTypes>
  </entry>

```

```

    </entry>
</ebx:voucherRequest>

```

The **voucherRequest** does not contain an **orderID** element. The high-level **preferredTypes** element is required and tells what content types the reading system accepts, in order of preference. The **entry** elements of the **voucherRequest** are optional. It is used when the reading system wishes to inform the voucher server of the preferred content types for a specific work. If present, each **entry** in the **voucherRequest** corresponds to an **entry** in the fulfillment instructions. The **preferredTypes** element for each entry contains a list of content types that the reading system can accept for the specific work, in order of preference.

#### 4.10.2.1 HTTP Implementation Note

The body of the message is the XML object.

Example:

```

HTTP/1.1 200 OK
Content-Type: text/XML
Content-Length: 1234

```

```
[1234 bytes of XML Credentials and Voucher Request]
```

#### 4.10.3 Owner Processing of Transfer Response from Receiver (give, lend)

After receiving the transfer response, the owner's voucher server should perform these validations:

- Verify that the EBX version supported by the receiver is supported by the owner. If not, return statusCode of 412 (Precondition Failed, Unsupported EBX Version).
- **For each** book in the order, perform the following operations:

The **<preferredTypes>** of the **<voucherRequest>** must be examined for each book in the order. The correct **<preferredTypes>** is obtained from the **<entry>** element of the **voucherRequest**, if it exists, or from the high-level default **<preferredTypes>** element. If the owner cannot deliver any of the formats specified for the specific book in the voucher request, then statusCode 406 (Not Acceptable) is returned in the **<entry>** element in the next request. The voucher is not issued.

If other errors occur while processing the request, set statusCode in the **<entry>** element in the response to an appropriate value. The voucher is not issued.

- If all of the vouchers are issued, the value in the statusCode of the response is set to 200 (OK). The entry tags are not required.
- If none of the vouchers could be issued, return statusCode of 204 (No Content, No vouchers could be issued). There should be an entry tag for every voucher that could not be issued with the detailed statusCode for the failure.

- If some, but not all of the vouchers could be issued, return statusCode of 206 (Partial Content, Some vouchers could not be issued). There should be an entry tag for every voucher that could not be issued with the detailed statusCode for the failure.

#### 4.10.4 Voucher Transmission Request from Owner to Receiver (give, lend)

The owner transmits a request to the receiver with an XML object containing the status of the operation and, if successful, the voucher(s) for the operation. The returned voucher(s) have the content-key encrypted with the public key of the receiver, which was obtained from the credentials sent in the previous response. Metadata for the particular books, including the URL for the encrypted content files, is contained in the voucher(s) that are received.

The XML status object is specified in the [TBS] section.

The XML format of vouchers is specified in the *Voucher Format* section on page 85. Following is an example:

```
<?xml version="1.0" encoding='UTF-8' ?>
<ebx:status ebx:version="0.8">
  <statusCode>...</statusCode>
  <statusMessage>...</statusMessage>
  <statusMessage>...</statusMessage>
  <entry>
    <bookID>ISBN:4444444444</bookID>
    <statusCode>...</statusCode>
    <statusMessage>...</statusMessage>
    <statusMessage>...</statusMessage>
  </entry>
</ebx:status>
<ebx:vouchers ebx:version="0.8">
  <ebx:voucher ebx:version="0.8">
    Please see Voucher section for details
  </ebx:voucher>
  <ebx:voucher ebx:version="0.8">
    Please see Voucher section for details
  </ebx:voucher>
</ebx:vouchers>
```

##### 4.10.4.1 HTTP Implementation Note

The x-EBX-Action header is used in the HTTP POST request.

Two examples, one with the x-EBX-Action header set empty, and one using the x-EBX-Action header.

```
POST /EBX?action=give HTTP/1.1
Host: www.acme.com
x-EBX-Action: ""
Content-Length: 789
```

[789 bytes of XML Status and Voucher Objects]

```
POST /EBX HTTP/1.1
Host: www.acme.com
x-EBX-Action: "give"
```

```
Content-Length: 789
```

```
[789 bytes of XML Status and Voucher Objects]
```

#### 4.10.5 Receiver Processing of Voucher Transmission Request (give, lend)

- If the statusCode of the response is 200 (OK), the vouchers are extracted from the response, and saved to persistent storage, then the receiver issues the response. Additionally, if the EBX e-book reading system should delete the transfer data XML file if it had possession or ownership of the file.
- If the statusCode of the response is 206 (Partial Content), some, but not all the vouchers were delivered. The delivered vouchers are extracted from the response, and saved to persistent storage, then the receiver issues the response, with the exact set of vouchers that were obtained. The statusCode of the failed entries may be examined to present feedback to the user.
- If the statusCode of the response is 204 (No Content), none of the vouchers were delivered. The statusCode of the failed entries may be examined to present feedback to the user.
- If the statusCode in the response is some error other than above, then none of the vouchers are delivered.
- If errors occur within the EBX e-book reading system while processing a response with status code of 200 OK, and prior to saving the voucher(s) from the response in persistent storage, the error condition may be presented to the user and/or logged.

Detailed error messages may be obtained from the <statusMessage> </statusMessage> tags of the response for presentation to the user or for logging.

Successfully received vouchers are passed along to the voucher engine. The voucher engine must commit the voucher(s) to persistent storage at this point. The protocol engine and voucher engine must be integrated in such a way that the protocol engine is able to obtain relevant metadata from the vouchers for each voucher that is received. At a minimum, the protocol engine needs to be able to obtain the URL for the encrypted content file from the voucher.

#### 4.10.6 Voucher Transmission Response from Receiver to Owner (give, lend)

The receiver's response to the voucher transmission request contains data in XML format as specified in the [TBS] section. The XML data includes the same action element as the previous voucherRequest object. The entry element for vouchers received is not required. If the entry

element is omitted, it is assumed that all vouchers from the previous request were received. If the receiver wishes to add every voucher received in the entry tag, it may. However, if only some of the vouchers could be stored, then in this case the successfully stored vouchers are passed as entry elements. Following is an example:

```
<voucherAck ebx:version="0.8">
  <action>give</action>
  <orderID>114816072033825</orderID>
  <entry>
    <bookID>ISBN:4444444444</bookID>
  </entry>
</voucherAck>
```

#### 4.10.6.1 HTTP Implementation Note

The body of the message is the XML object.

Example:

```
HTTP/1.1 200 OK
Content-Type: text/XML
Content-Length: 1234

[234 bytes of XML Voucher Ack Object]
```

#### 4.10.7 Owner Processing of Voucher Transmission Response from Receiver (give, lend)

When the owner receives the voucher transmission response, it performs the following action:

- If the request contains entries, the voucher corresponding to each entry is either deleted (for “give”) or marked as “lent” for the appropriate loan period (for “lend”).
- If there is no entry in the request, all vouchers corresponding to the request are either deleted (for “give”) or marked as “lent” for the appropriate loan period (for “lend”).

#### 4.10.8 Content Transmission Request from Owner to Receiver (give, lend)

For each voucher successfully transmitted to the receiver and committed to persistent storage, the owner transmits the corresponding content in a request to the receiver.

##### 4.10.8.1 HTTP Implementation Note

The x-EBX-Action header is used in the HTTP POST request.

Two examples, one with the x-EBX-Action header set empty, and one using the x-EBX-Action header.

```
POST /EBX?action=give HTTP/1.1
Host: www.acme.com
x-EBX-Action: ""
Content-Length: 380123
```

```
[380123 bytes of the encrypted content file]
```

```
POST /EBX HTTP/1.1  
Host: www.acme.com  
x-EBX-Action: "give"
```

```
Content-Length: 380123
```

```
[380123 bytes of the encrypted content file]
```

#### 4.10.9 Content Transmission Response from Receiver to Owner (give, lend)

The receiver responds to each e-book content request.

##### 4.10.9.1 HTTP Implementation Note

Example:

```
HTTP/1.1 200 OK
```

#### 4.11 Electronic Mail Transfer

It is not always feasible to transfer an electronic book using point-to-point protocols like TCP/IP or IrDA. Sometimes, both parties involved in a transfer are not connected simultaneously or do not publish their network addresses. An example of this situation, is when a consumer wishes to give a book to another consumer as a gift. Electronic mail may therefore be useful as a store-and-forward transport for electronic books.

The method by which EBX e-books are transferred via standard Internet e-mail protocols like SMTP and MIME is (T.B.S.)

#### 4.12 EBX Error Handling and Flow

The EBX transfer protocol defines a set of error mechanisms that must be adhered to in order to specify a robust system. The error mechanism uses status codes embedded in the XML response from voucher servers. The XML tags for status are `<statusCode>` and `<statusMessage>`.

##### 4.12.1 HTTP Implementation Note

Voucher servers always return HTTP 200 (OK) to every request. The true status of the operation is *ALWAYS* in the `<statusCode>` XML tag of the response.

##### 4.12.2 Successful Requests to Voucher Servers

Voucher servers must set the `<statusCode>` XML tag to 200 when a successful response is generated to a client request.

### 4.12.3 Failed Requests to Voucher Servers

The following table describes status codes used for error processing within the EBX system. They are initially adopted from HTTP status codes. They are used in the `<statusCode>` XML tag for EBX responses.

Voucher servers must set the `<statusCode>` XML tag to some code other than 200 when a failure occurs while servicing a client request. Whenever any error response is generated, voucher servers should generate human-readable text describing the error condition between the `<statusMessage>`.`</statusMessage>`.tags of the XML response to the failed request. Each `<statusMessage>`.`</statusMessage>`.element set represents a single line of diagnostic error text.

<code>&lt;statusCode&gt;</code>	Meaning; Suggested Message
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Moved Temporarily
303	See Other
304	Not Modified
305	Use Proxy
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
500	Internal Server Error
501	Not Implemented
502	Bad Gateway



503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported

#### 4.12.3.1 General Errors – 400

Most errors encountered by voucher servers while processing a client request will use the status code of 400 (Bad Request) in the response.

#### 4.12.3.2 Server Errors – 500 or 503

When the voucher server itself has failures, either Status code of 500 (Internal Server Error), or 503 (Service Unavailable) should be used in the response.

#### 4.12.3.3 No Acceptable Content – 406 (Not Acceptable)

If the request for vouchers (<voucherRequest>) is accompanied by <entry> tags with the <preferredTypes> elements specified, and the voucher server cannot deliver any of the formats specified in the request, then status code 406 (Not Acceptable) is returned with the response.

#### 4.12.3.4 Order Delivered and ACKed – 410 (Gone)

If an order has already been delivered from the voucher server, and the client has returned with the ACK request, and a subsequent request is made to the voucher server to either handoff the order (action=handoff), or to execute the fulfillment instructions for the order (action=purchase or action=borrow), the voucher server must return status code 410 (Gone) to indicate that this order has already been fulfilled and acknowledged.

### 4.12.4 Discussion of Voucher Fulfillment and ACK

Using the ACK creates a simple, fault-tolerant mechanism for voucher servers to know that clients have received the voucher(s) from a fulfillment request.

Under normal circumstances, the client gets the voucher(s) for an order, adds them to persistent storage, makes the ACK request to the voucher server, and the order is marked ACKNOWLEDGED by the voucher server. The voucher server responds with status code 200 when it receives the ACK from the client.

If the connection breaks during the response to the voucher request to the voucher server for action=purchase or action=borrow, or if errors occur in the client that prevent persistent storage of the voucher, then the client will not make the ACK request to the voucher server, the order will not be marked ACKNOWLEDGED, and the voucher(s) for the order may be obtained by a subsequent request to the voucher server.

If a request is made to a voucher server for an order that has already been acknowledged, then the voucher server will respond to the request with status code 410 (Gone), to indicate that this request was already fulfilled and acknowledged.

If the ACK request from the client to the voucher server fails to reach the server, or if the client does not receive the status code 200 from the voucher server in response to the ACK, the client has two choices: It can ignore the fact that the request was not acknowledged by the voucher server, or it may re-transmit the ACK request.

Voucher servers must not require ACK from a client. Failure to receive ACK simply means that the voucher server may still honor subsequent requests to fulfill an order.

It is the responsibility of the voucher server to perform proper accounting in the case that vouchers are being fulfilled after the first fulfillment request has been honored, but before an ACK has been received for the request.

If a voucher server detects multiple fulfillment requests for an order that has not been acknowledged, the voucher server may flag this condition as possible abuse of the EBX protocol by a "rogue client".

## 5 Voucher Engine Model

### 5.1 Voucher Engine Processing

[**Editorial note.** This section is incomplete in Version 0.8.]

A voucher engine is a processor that responds to requests from the protocol engine. The voucher engine receives input, applies a set of rules, and may generate output, which is typically a new voucher. The input to the voucher engine generally includes the following:

- The request
- Identification of the entity making the request
- A statement of rights

The publisher creates the initial voucher, which contains the initial set of rights for a work. Subsequent statements of rights are obtained from an existing voucher.

### 5.2 Voucher Engine Rules

[**Editorial note.** This section is incomplete in Version 0.8.]

The voucher engine's processing is constrained by a set of rules, including these:

- Rights are denied unless explicitly granted.
- Expansion of rights is disallowed.
- Narrowing of rights is allowed.
- Processing requires valid:
  - Voucher signature
  - Requestor ID certificate
  - Requestor authorization certificate (which may be the same as the ID)
  - Requestor technology certificate
- Output complies with:
  - Publisher-granted rights
  - Authorizations granted the recipient
- Parse the latest general rights you can

- If rights don't parse:
  - If optional, ignore
  - If mandatory, abort the output

### **5.3 Voucher Engine Interfaces**

[**Editorial note.** This section is incomplete in Version 0.8.]

The voucher engine performs a set of abstract operations described in this section.

#### **5.3.1 Create Voucher**

Input:

- Rights set (contains work ID)
- Content format (including encryption method, encryption algorithm, key length, etc.)
- Private key (for use in signing voucher body)
- (Optional) Public key (for encrypting content key).
  - Question: does this public key match the private key used to sign the voucher body, or some other private key? I.e., are different certificates required for signing and for encrypting the content key?

Processing:

- Check the voucher engine's own credentials – only a publisher can create vouchers. This implies that we can identify the publisher role from a certificate.
- (Optional) encrypt content and place encrypted content key in voucher.
- Generate and sign the rights kernel.
- Generate rights for the distributor role.
  - Note: these rights are the same as those in the rights kernel.
- Sign voucher.

Output:

- Voucher
  - Note: this voucher is an object that is created once per work and is stored persistently.

### 5.3.2 Issue Voucher

Input:

- Existing voucher
- Rights set (rights to remove or rights to select? Is this an implementation issue?)
- Credentials of the requester and signed nonce (in a PKCS #7 signed data message)

Processing:

- Check credentials
  - Validate all certificates in the chain.
  - Verify that the trust level of the requester is at least as high as the minimum trust level in the voucher.
- Verify the nonce
  - Validate the signature on the nonce.
  - Ensure that the nonce is the same one you issued to the requester.
- (Optional) encrypt content. [**Editorial note.** The ability to encrypt content during the Issue Voucher operation is intended to accommodate systems that encrypt content differently for each user. The implications for the protocol and trust models has not yet been investigated.]
- Encrypt content key using the public key of the requester (if the encrypted key is not obtained from Encrypt Content).
- Compare the input rights set with the rights in the existing voucher to ensure that the new rights do not expand existing rights.
- Remove specified rights (if any)
- Sign voucher

Output:

- Voucher

### 5.3.3 Revoke Voucher

Input:

- Existing voucher

Processing:

- Invalidates voucher but does not delete it (could happen on server)

Output:

- None

#### **5.3.4 Import Voucher**

Input:

- Voucher

Processing:

- Validate the voucher signature.
- Validate the signature on the rights kernel.
- Transfer voucher to repository
- Maybe merge vouchers?

Output:

- None (or handle to the stored voucher)

#### **5.3.5 Delete Voucher**

Input:

- Voucher

Processing:

- Remove voucher from repository

Output:

- None

#### **5.3.6 Issue Nonce**

Input:

- None

Processing:

- Generate a nonce
- Record the nonce (i.e., record that this nonce is one you issued).

Output:

- Nonce

### **5.3.7 Issue Credentials**

Input:

- Nonce

Processing:

- Generate a PKCS #7 signed data message, using the nonce as data.

Output:

- Credentials (the PKCS #7 signed data message)

### **5.3.8 Encrypt Content**

Input:

- Cleartext content
- Content format, including encryption method, encryption algorithm, key length, etc.
- Public key (for encrypting content key)

Processing:

- Generate content key.
- Encrypt content with the content key.
- Encrypt content key with the public key passed in.

Output:

- Encrypted content.
- Encrypted content key.

### **5.3.9 Decrypt Content**

Input:

- Encrypted content.
- Voucher.

Processing:

- Retrieve the private key.
- Decrypt the content key in the voucher, using the private key.
- Decrypt the content, using the content key and information about the content format, encryption algorithm, method, key length, etc., from the voucher.

Output:

- (Some amount of) cleartext content, depending on trust level. (Keeping all rendering of cleartext content within the voucher engine is potentially more secure than passing any cleartext content outside the voucher engine.)

#### **5.4 Voucher Engine Processing of EBX Rights**

[**Editorial note.** This section is incomplete in Version 0.8.]

As the EBX specification evolves, voucher engines compliant with different versions of the EBX specification need to coexist with vouchers compliant with different versions of the EBX specification. When a voucher engine processes a voucher, two kinds of version mismatch are possible:

- The voucher engine complies with a later version of the specification than the voucher.
- The voucher engine complies with an earlier version of the specification than the voucher.

This section describes the mechanism by which voucher engines write and read EBX rights.

##### **5.4.1 Writing EBX Rights in a Voucher**

A voucher can contain multiple rights descriptions. Among these are rights specified by EBX and possibly rights specified by other descriptions.

A voucher contains at least two EBX rights sections, each of which is identified with an EBX version number:

- A full-function section that evolves relatively freely with each EBX version.
- A basic section that evolves in very limited ways. In particular, elements of basic rights can be changed only in nonrestrictive ways; any restrictive change requires a new element.



When a voucher engine writes a voucher, it must put two EBX rights sections into the voucher:

- The current version of the full-function EBX rights section.
- The current version of the basic EBX rights section.

A voucher engine may (but is not required to) also write earlier versions of the full-function section.

For any possible permission (such as permission to read or print), the basic rights section must not grant that permission if the permission is either restricted or is not granted in the full-function rights section. In writing the basic EBX rights section, a voucher engine must grant a permission only if that permission is granted and is not restricted in the full-function EBX rights section.

For example, suppose that the full-function EBX rights section allows reading of the content at any time before an expiration date specified in the full-function section. The expiration date thus restricts the read permission. Basic EBX rights have no mechanism for restricting a permission by an expiration date. The basic section must therefore grant no read permission in this case.

#### **5.4.2 Enforcing EBX Rights in a Voucher**

In reading and enforcing EBX rights in a voucher, a voucher engine must follow these rules:

- A voucher engine must not use a full-function EBX rights section in the voucher if the full-function rights section has an EBX version that is later than the voucher engine's EBX version. In the usual case, the voucher engine uses the latest-versioned EBX full-function rights set in the voucher that is less than or equal to the voucher engine's EBX version.
- If no full-function EBX rights section in the voucher has an EBX version that is less than or equal to the voucher engine's EBX version, the voucher engine must use the basic EBX rights set in the voucher, regardless of the EBX version of the basic rights set. If the basic rights set has an EBX version that is later than the voucher engine's EBX version, the voucher engine ignores any elements of the basic rights set that do not exist in the voucher engine's EBX version.

## 6 Metadata Format

Information that identifies or describes an e-book is *metadata*. EBX distinguishes between two forms of metadata:

- Extended metadata, used for such purposes as describing an e-book on a bookseller's Web site and cataloging an e-book in a library.
- Concise metadata, used for such purposes as identifying available and preferred content formats, logging a transaction, identifying an e-book in an error message, and displaying the progress of downloading an e-book.

The format and transfer protocol for extended metadata are outside the scope of the EBX specification. A widely accepted standard format for communicating e-book product information among publishers, distributors, and booksellers is ONIX International [ONIX101]. A widely accepted standard format for communicating bibliographic information for libraries is the MARC 21 Format for Bibliographic Data [MARC21].

EBX specifies the representation of concise metadata for use in such objects as vouchers and fulfillment instructions. Concise metadata is defined by the **metadata** XML element.

### 6.1 Metadata Element

The **ebx:metadata** element can contain any of the Dublin Core [DC11] metadata elements, version 1.1 or higher, and any of the Dublin Core metadata qualifiers [DCQ10], version 1.0 or higher. All Dublin Core metadata elements are repeatable within the **ebx:metadata** element. The **ebx:metadata** element, or an element that contains the **ebx:metadata** element, must declare the **dc** namespace (containing Dublin Core metadata elements) and the **dcq** namespace (containing Dublin Core metadata qualifiers), as in the following example:

```
<ebx:metadata
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcq="http://purl.org/dc/qualifiers/1.0/">
```

Each **ebx:metadata** element must contain at least the following Dublin Core metadata elements for the work it describes: **dc:identifier**, **dc:creator**, and **dc:title**.

#### 6.1.1 Identifier Element

The **dc:identifier** element should include the **dcq:scheme** attribute, which names the encoding scheme of the identifier. EBX specifies two values for **dcq:scheme** that are extensions of the Dublin Core metadata qualifiers specification: "ISBN" [ISBN] and "DOI" [DOI].

### 6.1.2 Format Element

The **dc:format** element should include the **dcq:scheme** attribute, which names the encoding scheme of the format of the e-book. The value of the **dcq:scheme** attribute should be “IMT” [IMT].

EBX specifies the following required attributes for **dc:format** that are extensions of the Dublin Core metadata qualifiers specification:

#### **ebx:encryptionMethod**

Encryption method for the e-book content. Valid values include but are not restricted to the following:

- “none”
- “pdfcrypt”
- “pdffilter”
- “zipencode”

#### **ebx:encryptionAlgorithmName**

Algorithm for encrypting the e-book content. Valid values include but are not restricted to the following:

- “none”
- “RC4”
- “DES”
- “3DES”

#### **ebx:encryptionAlgorithmExtra**

Parameter or qualifier for the algorithm for encrypting the e-book content. Valid values include but are not restricted to the following:

- “none”

#### **ebx:encryptionKeyLength**

An integer representing the length in bits of the key used to encrypt the e-book content. Example values:

- “40”
- “64”
- “128”

### 6.1.3 Metadata Example

Following is an example **ebx:metadata** element:

```
<ebx:metadata ebx:version="0.8"
```

```
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcq="http://purl.org/dc/qualifiers/1.0/"
<dc:identifier dcq:scheme="ISBN">9876543210</dc:identifier>
<dc:creator>Jane Smith</dc:creator>
<dc:creator>John Doe</dc:creator>
<dc:title>The Joy of E-Books</dc:title>
<dc:date dcq:scheme="W3CDTF">
  <dcq:issued>2000</dcq:issued>
</dc:date>
<dc:description>
  <dcq:abstract>Describes the process of creating e-
  books.</dcq:abstract>
</dc:description>
<dc:format
  dcq:scheme="IMT"
  ebx:encryptionMethod="pdfcrypt"
  ebx:encryptionAlgorithmName="DES"
  ebx:encryptionAlgorithmExtra="none"
  ebx:encryptionKeyLength="56">
  application/pdf
</dc:format>
</ebx:metadata>
```

## 7 Voucher Format

It is important that the format of Vouchers and Credentials be standardized to ensure interoperability between publishers, booksellers, libraries, and consumers. To allow vouchers and credentials to be readily used by many components, the object format for these objects is XML [XML98]. Since metadata is not stored in vouchers or credentials, the use of XML in EBX is limited to the UTF-8 encoding.

### 7.1 Rights Specification

All rights are assigned at the initial creation of an encrypted e-book and its associated voucher (the “master voucher”) during the publication process in a voucher server by the EBX role of Publisher.

The **publisher** section of the voucher (rights kernel) contains this initial rights description, and is signed so that at any point in the downstream life cycle of the voucher in the value chain, the rights granted to the current licensee can be compared to the rights set forth at the time the initial master voucher was created.

The **licensee** section of the voucher contains the set of rights granted to the current holder of the voucher.

Any time that a voucher is issued, the issuing voucher engine will modify the licensee section of the issued voucher with appropriate rights, depending on the context of the transaction.

#### 7.1.1 Rights Overview

EBX specifies two types of rights. The licensee is granted some or all of these rights.

- **Transfer rights** allow the holder to sell, give, or lend the book, or any other similar operation. These rights are typically applied when the rights holder (licensee) is a voucher distributor [server].
- **Usage rights** allow the holder to display, print, or to copy the book to the clipboard, or any other similar operation. These rights are typically applied when the rights holder (licensee) is a consumer.

Rights may be constrained in various ways.

Rights may often be limited in scope.

The scope of transfer rights could be limited by quantities or counts. The scope of usage rights could be limited by time periods and frequencies. These are only examples, and are not meant to be mutually exclusive or all-inclusive.

Usage rights may be limited to certain portion of the work, limited by a device class, or to certain authorization contexts.

### 7.1.2 Licensee

The rights kernel, or publisher’s rights, are created during the create voucher transaction at the publisher voucher server. The right kernel “locks in” the rights available to all consumers of the voucher as it is issued for downstream consumption in the value chain. These consumers are referred to as the licensees.

When a voucher is issued, the licensee (the entity that the voucher is being issued to) is granted rights as specified in the rights kernel section that applies to the transaction.

Licensee	Description
Voucher Distributor	This section of the rights kernel contains the rights that a voucher distributor has. Obvious examples will be the sell and modify rights. The lend right would be included here to enable the voucher for lending by a voucher server on behalf of a library.
Consumer	This section of the rights kernel contains the rights that a consumer has. All usage rights would be specified here. If the voucher is enabled for consumer-to-consumer lending, some transfer rights would also be specified here as well.

### 7.1.3 Transfer Rights

Some set of transfer rights are always granted to the licensee when the voucher is issued to another voucher distributor. Transfer rights may or may not be granted to the licensee when the voucher is issued to a consumer

A transfer right that is not specifically stated in the voucher is not granted to the licensee.

**Note:** When the master voucher is created, there is an implicit licensee section created that should grant infinite transfer rights. This master voucher is created on behalf of the rights owner (publisher), therefore the rights owner can distribute them forever. This allows all voucher distributors (one of which implicitly operates on behalf of the publisher) to behave generically to follow rules to flow the vouchers downstream, either directly to consumers or to other voucher distributors.

Right	Description
Sell	The licensee may issue vouchers from this voucher as the result of value being exchanged (sale).
Lend	The licensee may issue vouchers from this voucher for the purposes of a loan.
Give	The licensee may issue vouchers from this voucher for the purposes of a gift.
Modify	The licensee may modify (narrow) rights when issuing a voucher

Other possible transfer rights:

Vend, Distribute, Return, Derive, Share

### 7.1.4 Usage Rights

Some set of usage rights are always granted to the licensee when the voucher is issued to a consumer. Usage rights may or may not be granted to the licensee when the voucher is issued to a voucher distributor.

A usage right that is not specifically stated in the voucher is not granted to the licensee.

<b>Right</b>	<b>Description</b>
Display	The licensee may display the work.
Print	The licensee may print the work
Copy	The licensee may copy the work
Narrow Rights	The licensee may narrow the rights if vouchers are issued from this voucher

Other possible usage rights:

Backup, Export, Preview, Delete, Annotate, Perform, Install, Uninstall

### 7.1.5 Authorization context

A right may be modified by an authorization context such that the right is constrained for use by users (licensees) within the authorization context. If the authorization context is not specified, the right is available to any licensee that holds the voucher.

It is recognized that the authorization context must be able to be determined by the reading system. The methods for this have not really been discussed. If the authorization context is specified, but cannot be determined, then the right is not granted.

<b>Authorization Context</b>	<b>Description</b>
Network	Right is restricted to some network
User Group	Right is restricted to some user group
Geographical Area	Right is restricted to some geographical area
Specific device	Right is restricted to some specific device
Subscription Holder	Right is restricted to some subscription holder
IP Address	Right is restricted to some IP Address

### 7.1.6 Consideration

A right may be modified by consideration such that the right is not available for use unless some exchange of value is performed. The implication is that there would be some URI or other information provided with the consideration modifier so that the licensee's reading system can provide the correct information to exchange value in order to remove the constraint. If consideration is not specified, the right is available to the licensee that holds the voucher, without any further consideration.

[**Editorial note.** Currently in EBX, we have not specified how a voucher may be updated, or exchanged. The implication of consideration is that the licensee’s voucher is reset to show that value has been exchanged.]

<b>Consideration</b>	<b>Description</b>
Consideration	Right is not available until value is exchanged

### 7.1.7 Portion

A right may be modified by a portion such that the right is restricted to a specific portion of the work, or on a contained object of the work. If the portion is not specified, the right is available across the entire work referenced by the voucher.

It is recognized that the portion should specify an appropriate value relative to the file format of the work. For example, some file formats may not be able to delineate portions such as paragraph, page, chapter, while others may.

If the portion is specified, and the reading system cannot determine/understand the portion of the work, then the right is not granted.

<b>Portion</b>	<b>Description</b>
TOC	Right is restricted to the Table of Contents
Chapter	Right is restricted to a certain chapters
Page	Right is restricted to certain pages

### 7.1.8 Target

Certain rights may be modified by a target such that the right is restricted to a specific target device class. If the target is not specified, the right may be exercised on any device class available to the licensee.

It is recognized that target is only appropriate when used with certain “rendering” types of rights (Display, Print, Copy, for example).

If the target is specified, and the reading system does not have the device class available, then the right is not granted.

<b>Target</b>	<b>Description</b>
Screen	Right is restricted to use on a screen
Daisy screen reader	Right is restricted to use on a ??? daisy screen reader
Braille Device	Right is restricted to use on a Braille Device
Image Printer	Right is restricted to use on a Image Printer
Trusted printer	Right is restricted to use on a Trusted printer
Conventional printer	Right is restricted to use on a Conventional printer
Disk	Right is restricted to use on a Disk



Target	Description
Clipboard	Right is restricted to use on the Clipboard

### 7.1.9 Scope

Certain rights may be limited in scope such that the right is restricted to a certain time period, interval, or count. If the scope is not specified, the right may be exercised as often as the licensee desires, forever into the future.

Scope	Description
Start time	Right may be exercised after the start time [absolute time]
End time	Right may be exercised until the end time [absolute time]
Metered time	Right may be exercised for a duration not to exceed the metered time [time period]
Count	Right may be exercised a certain number of times [integer]
Interval	Restricted rights may be exercised on a recurring interval [time period]. This normally would be used in conjunction with count or metered time.
Maximum	Maximum limits may be set upon the Count. [integer]

When count is used, there will be appropriate modifiers relative to the right that the count is limiting. For example:

Print 3 pages - Sell 5,000 copies – Display 30 times

Here are some other simple examples of how scope could limit a right:

Print 5 pages every 3 days (count, interval)

Print 5 pages every 3 days maximum 30 pages (count, interval, maximum)

Display starting 8/10/2000 until 8/15/2000 (start time, end time)

Display until 8/3/2000 (end time)

Display for 14 hours (metered time)

### 7.1.10 Consumer's knowledge of rights

A consumer in legitimate possession of an e-book should be advised of the specific rights available to the consumer.

## 7.2 Voucher Object

[**Editorial note.** This section has not yet been edited for Version 0.8.]

This section lists the initial set of EBX Voucher elements. A complete XML DTD will eventually be provided to formally define the syntax of a voucher.

An example of an EBX voucher encoded in XML is:

```

<?xml version="1.0" ?>
<EBX-Voucher version="0.5">
  <ID type="ISBN"> 0130614661 </ID>
  <ContentKey type="DES56" EncryptedWith="RSA1024">
    34293aaNKi83jaJKsdfkjakkajssdf849238989=
  </ContentKey>
  <CopyCount> 1 </CopyCount>
  <Rights> Lendable, Givable, Sellable </Rights>
  <PersonalUse>
    <Interval> Week </Interval>
    <CopyUnit> Paragraph </CopyUnit>
    <CopiesAllowed> 10 </CopiesAllowed>
    <CopiesMade> 0 </CopiesMade>
    <StartOfInterval> 1010 </StartOfInterval>
  </PersonalUse>
  <Lending>
    <Status> Lent </Status>
    <Period> 14 </Period>
    <StartTime> 1010 </StartTime>
    <TimeoutTime> 1024 </TimeoutTime>
  </Lending>
  <MAC Hash="SHA1"> 23abCuH23748== </MAC>
</EBX-Voucher>

```

### 7.2.1 EBX-Voucher – Voucher start-tag

Each voucher must be enclosed in an EBX-Voucher element. The EBX-Voucher start-tag which serves to identify the element as a Voucher, identify the revision code of the format.

The following attribute is required in the EBX-Voucher start tag:

- **Version** - Version number. A string where the first digit is the major rev code, the second digit is the minor rev code. The current version number is:
  - “0.5”

An example EBX-Voucher element is:

```

<EBX-Voucher version="0.5">

</EBX-Voucher>

```

### 7.2.2 ID – ISBN, DOI, or URN element

The ID element contains the ISBN or DOI of the protected e-book. ISBNs are numeric, and contain 10 digits (excluding hyphens). DOIs are in the standard DOI format [DOI] which is a DOI prefix, a delimiter (“/”) and a publisher-assigned value (typically an ISBN). The ID can also be a URN [URN97], which has the general form “type=name”.

The following attributes are required in the ID start-tag:

- **Type** – specifies the type of the identifier: “ISBN”, “DOI”, or “URN”.

An example ID element is:

```

<ID type=ISBN> 0130614661 </ID>

```

### 7.2.3 ContentKey – Content decryption key element

This element describes the type of the symmetric content decryption key for the book, and contains the key itself. The symmetric key value is ALWAYS sent and stored encrypted with the EBX public key of the owner of the voucher.

The following attributes are required for the ContentKey start-tag:

- **Type** – specifies the algorithm and key size of the symmetric key. Currently defined values are:
  - “RC440” – 40-bit RC4.
  - “DES56” – 56-bit DES (recommended).
  - “3DES” – Triple-DES.
- **EncryptedWith** – specifies the public key algorithm and key size that was used to encrypt the symmetric content key. Currently defined values are:
  - “RSA512” – RSA with a modulus of 512 bits.
  - “RSA1024” – RSA with a modulus of 1024 bits (recommended).
  - “RSA2048” – RSA with a modulus of 2048 bits.

The content of the ContentKey element is a symmetric key padded and encrypted with the RSA algorithm using the encryption process defined in Section 8 of PKCS#1 [PKCS95]. Since the symmetric key is encrypted with a public key, the block type is always 02. Therefore, if either a 56-bit DES or a 40-bit RC4 key is to be used and encrypted with a 512-bit RSA key, the resulting value is always the size of the modulus of 512 bits (64 bytes). [Is there any reason to use PKCS#12 here?]

Since the value of a PKCS#1 encryption block is binary, it is always converted to base64 encoding before it is used as the content of a ContentKey element.

### 7.2.4 CopyCount – Count of authorized copies element

The content of this value is the number of copies the holder of the voucher owns. There are currently no attributes defined for the CopyCount start-tag.

Typically, this element has the value “1” for consumers and libraries and a higher value for booksellers and distributors. If a library (or consumer) purchases more than one copy of an e-book, then this count will be higher than 1 in their vouchers as well. For booksellers and distributors, the CopyCount represents the number of copies of the e-book that the bookseller or distributor has purchased and is allowed to resell.

For example:

```
<CopyCount> 1 </CopyCount>
```

### 7.2.5 Rights – Basic permissions element

The basic rights define whether the owner has the permission to give, lend, and/or sell copies of the voucher. There are no attributes currently defined for the Rights start-tag.

The values in the content of the Rights element are separated by commas. The allowable values for the content of the Rights element are:

- **Lendable** – The e-book can be lent to another consumer or library.
- **Givable** – The e-book can be given to another consumer or library.
- **Sellable** – The e-book can be sold to another consumer, distributor, or bookseller.

For example:

```
<Rights> Lendable, Givable, Sellable </Rights>
```

### 7.2.6 Lending – Lending Timeout and Status Element

This element might not be present (the book has not been lent or borrowed). It contains a "status" code (book is borrowed, book is lent out, book is not lent out), which tells the system whether the book can be accessed on the current device. If the element is missing, then access is allowed.

The Lending elements are:

- **Status:** Possible status codes are:
  - **Borrowed** - Book is borrowed;
  - **Lent** - Book is lent out;
  - **Owned** - book is not lent out (equivalent to no Lending element).
- **Period** – Total number of days the book has been lent or borrowed.
- **StartTime** - A number indicating (in days) the time at which a lending period started. It is initialized on the current machine when a book is lent or borrowed, and is used to determine when the lending period is over. This field is only present when the status is either "Borrowed" or "Lent". The value is calculated as: (current clock ticks) / (ticks per day).
- **TimeoutTime** - A number representing the end of the lending period, in days. The system gets the current "day" as above (current ticks / ticks-per-day), and compares with the Lending Start Time element to determine whether or not the lending period has expired. This element is only present if the status is either "Borrowed" or "Lent". The StartTime

For example:

```
<Lending>
  <Status> Lent </Status>
  <Period> 14 </Period>
  <StartTime> 1010 </StartTime>
  <TimeoutTime> 1024 </TimeoutTime>
</Lending>
```

### 7.2.7 PersonalUse – Personal use element

If this element is missing, no personal use permissions are allowed. The PersonalUse element is a container element for the actual personal use permissions elements. The personal use permissions specify how often and what quantity of information can be copied (or printed) from the e-book. The personal use permissions elements are:

- **Interval** - The interval during which personal use copies and/or printouts are tracked. The choices are: Day, Week, Month, Year.
- **NumberAllowed** - The number of personal use copies and/or printouts that can be made during Interval.
- **CopyUnit** – A value indicating what portions of the work can be copied and/or printed. The choices are: Paragraph, Page, Chapter, EntireWork.
- **StartOfInterval**: The number of "days", calculated as: (current ticks) / (ticks per day). This field is used to determine when an interval has expired. This a local state value and is not transmitted between systems. It should be initialized the first time the voucher is used by the current owner.
- **CopiesMade** - The number of personal use copies and/or printouts that have been made in the current interval. This a local state value and is not transmitted between systems. It should be initialized the first time the voucher is used by the current owner.

For example:

```
<PersonalUse>
  <Interval> Week </Interval>
  <CopyUnit> Paragraph </CopyUnit>
  <CopiesAllowed> 10 </CopiesAllowed>
  <CopiesMade> 0 </CopiesMade>
  <StartOfInterval> 1089 </StartOfInterval>
</PersonalUse>
```

### 7.2.8 MAC – Message Authentication Code element

The MAC element is required to be the last element in all EXB-Vouchers and is used to provide a check on the rest of the elements in the EBX-Voucher element. The MAC is a hash of all of the elements in the EBX-Voucher (up to, but not including, the MAC element) combined with the symmetric content key (a plaintext version of it, of course) according to the algorithm described in the HMAC RFC, RFC2104 [HMAC].

The purpose of the MAC is to prevent anyone from tampering with the voucher, while it is in transit in the network or stored in permanent storage. However, to provide both privacy and integrity with a single key, the symmetric content decryption key is used to encrypt the hash.

To verify the MAC and validate the voucher, the recipient should first decrypt the content key element and then compute the HMAC hash on the voucher elements (up to but not including the MAC element). If the computed hash matches the encrypted hash in the MAC element, then the voucher is valid and has not been tampered with; otherwise, it should be rejected.

The MAC element content is currently computed using an SHA-1 hash (160 bits) combined with the symmetric content key. The resulting HMAC hash is then base64 encoded.

The following attribute is required on the MAC start-tag:

- **Hash** – Hash algorithm name. The only value currently supported is:
  - **“SHA1”** – Secure Hash Algorithm 1.

For example:

```
<MAC Hash="SHA1"> 23abCuH2akjskdjfkjjskdjkdj3748== </MAC>
```

## 8 Format of Other Objects

### 8.1 Credentials Object Format

This section lists the initial set of EBX Credentials elements. A complete XML DTD will eventually be provided to formally define the syntax of credentials.

An example of a Credentials object is:

```
<?xml version="1.0" ?>
<ebx:credentials version="0.8">
  <ebx:authenticationScheme
    ebx:encryptionAlgorithmID="tueh87yRHiuy975hfh4rd"
    ebx:encryptionAlgorithmName="RSA"
    ebx:signatureAlgorithmID="n457skYh29H8erg"
    ebx:signatureAlgorithmName="RSA-SHA1">pki-rsa-sha1
  </ebx:authenticationScheme>
  <nonce>lbJkMjTeUz1OL9tXaYhNSnR42iIq</nonce>
  <signedData>MIIDwQYJKoZIhvcNAQcCoIIDsjCCA64CAQExCzAJBgUrDgMCGGUAM
AsGCSqGSIb3DQEHAaCCAnQwgGJwMIIB2aADAgECAGEAMA0GCSqGSIb3DQEBBQUAMG
oxaDAJBgNVBAYTAlVTMBQGA1UECBMNTWFzc2FjaHVzZXR0czAhBgNVBAsTGlRFU1Q
gQ0VSVElGSUNBVEUgQVVUSE9SSVRZMCIgA1UEChMbr2xhc3Nib29rLCBJbmMuIC0g
VEVTVCBPTkxZMB4XDTAwMDcwOTIwMjYzOFoXDTEwMDcwODIwMjYzOFowaWwYjFoMAkGA
1UEBhMCMVVMwFAYDVQQIEw1NYXNzYWNodXNldHRZMCEGA1UECXMaveVTVCBDRVJUSU
ZJQ0FURSBVVVRIT1JjVjFkwIgwYDQVQKEExtHbGFzc2Jvb2ssIEluYy4gLSBURVNUIE9
OTFkwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANPWjIW6aw8yE9PwLd+IevFz
Q4aONTzI8ptNUS1auKzNWhh8Q1YXYP5eiSgQsS6y19zvXod17tgQxw5Wc3uvRsJt1
nU5LRqIALj+6z3BHovVbUoUX6QUoVWffZz575XXHZ5c8DxeCwVQJWNWSZ9qJA4HgW
6BvNUczVC/aYQaS6mDagMBAAGjJjAkMBIGA1UdEwEB/wQIMAYBAf8CAQAwDgYDVR0
iAQH/BAQwAjaAMA0GCSqGSIb3DQEBBQUAA4GBAHpAL2BuMHbPRgfSPg/SZDUsN81q
nxAZLtgzkd5hzYw68rCJfL8MZWGCBaTm7M60Cza95HYoeiFT68Sg85ZehcmBjNebZ
ypt4tYCh5A4uqjOuUIBzXzZcbCA8vouCPkUPcTdljJ0hrJc/SZZUe5wke2GWjn1t8
7PM0Z5JbJzcZQxMYIBFTCCARECAQEwbzBqMWgwcQYDVQQGEwJVUzAUBgNVBAGTDU1
hc3NhY2hlc2V0dHMwIQYDVQQLEExpURVNUIEUFULRJRklDQVRFIEFVVEhPUklUWTAi
BgNVBAoTG0dsYXNzYm9vaywSW5jLiAtIFRFRU1QgT05MMWQIBADAJBgUrDgMCGGUAM
A0GCSqGSIb3DQEBQUABIGAKyw6vEziw8aPiWJ/BPFCIAxRnFnYRy4xXYruM5myTg
eygAff//CbaFN92pXOE/48Mf987PRbECOVjxQ1kY2M6bHQq7r7CDEx8VmYEXjf6v6
IlCaEmESmTyPzUwbKzhtuBM8VHg6xlRm/GgPcdUDuAol0///lcujXJQ6+U/w/s9w=
  </signedData>
</ebx:credentials>
```

#### 8.1.1 Credentials – Credentials start-tag

Each credentials object must be enclosed in an Credentials element. The Credentials start-tag which serves to identify the element as a Credentials and identify the revision code of the element.

The following attribute is required in the Credentials start tag:

- Version - Version number. A string where the first digit is the major rev code, the second digit is the minor rev code. The current version number is:
  - “0.8”

An example Credentials element is:

```
<ebx:credentials version="0.8">  
  
</ebx:credentials>
```

### 8.1.2 AuthenticationScheme

The authenticationScheme element contains the authentication scheme that the recipient is using for the transaction. The authentication scheme used in the example is only an example. Refer to Appendix A, Certification Policies and Procedures, for details on the algorithms supported. The authentication scheme element contains the following optional attributes:

#### 8.1.2.1 EncryptionAlgorithmID

The encryptionAlgorithmID attribute contains the OID of the encryption algorithm used in a public-private key or other authentication scheme. This algorithm identifier specifies how the server should encrypt the content key, including any particular variations (e.g., padding, etc.). The algorithm identifier also specifies how the server should interpret the recipient's public key. The OID is encoded using the Basic Encoding Rules (BER) [Reference to \*\*\*], and then base64 encoded [MIME]. This attribute is optional.

#### 8.1.2.2 EncryptionAlgorithmName

The encryptionAlgorithmName attribute contains a human-readable name for the encryption algorithm. This attribute is optional, and is present only for readability.

#### 8.1.2.3 SignatureAlgorithmID

The signatureAlgorithmID attribute contains the OID of the signature algorithm used to produce the signed data message. The OID is encoded using the Basic Encoding Rules (BER) [Reference to \*\*\*], and then base64 encoded [MIME]. This attribute is optional.

#### 8.1.2.4 SignatureAlgorithmName

The signatureAlgorithmName attribute contains a human-readable name for the signature algorithm. This attribute is optional, and is present only for readability.

#### 8.1.2.5 Value

The value of the ebx:authenticationScheme element is a human-readable mnemonic, representing the entire authentication scheme. The server must be able to interpret the credentials based solely this value, and must not rely on the algorithm IDs or names.

### 8.1.3 Nonce

The nonce element contains a cleartext copy of the nonce issued by the current voucher owner. For a purchase or borrow transaction, the nonce is obtained from the fulfillment instructions described in the *Voucher Server Processing of Handoff Request* section on page 53. For a give or



lend transaction, the nonce is obtained from the transfer data as described in the *Transfer Request from Owner to Receiver (give, lend)* section on page 64. The nonce is base64 encoded. For example:

```
<nonce>lbJkMjTeUz1OL9tXaYhNSnR42iIq</nonce>
```

#### 8.1.4 SignedData

The signed data element contains a base64-encoded PKCS #7 signed data message [PKCS]. The following constraints apply to the signed data message:

The signed data message must contain a single signerInfo.

The signed data message must not contain any authenticatedAttributes.

The signed data message must include the signer's certificate and all certificates that form a certificate chain up to and including a certificate that is signed by an EBX trusted Root CA.

The signed data message may contain the certificate of an EBX trusted Root CA, but is not required to contain such a certificate. The certificate(s) of the EBX Root CA(s) may be assumed to be available at the server end of any transfers involving certificate chains.

The signed data message should not contain any crls. The voucher server must ignore any crls found in the signed data message.

The voucher server may any supported algorithm combinations as defined in Appendix A, Certification Policies and Procedures, and should list them in the algorithm negotiation section of the transfer protocol. See section 4.7 and 4.8 for details on the algorithm negotiation.

## **9 Appendix A: Certification Policies and Procedures**

This appendix will be enlarged to describe methods, policies, and procedures used by certifying organizations. Initially the policies will be based on interpretation of this specification. Reports from certifying organizations and from inspections will be used by the EBX Working Group to refine these procedures.

## 10 Appendix B: Applicable Laws

### 10.1 U.S. Export Laws

Export of encryption software in American-made software products requires an export license from the U.S. government. U.S. implementers are cautioned to check carefully with their legal advisers and with the relevant U.S. government agencies (such as the Bureau of Export Administration, [www.bxa.doc.gov](http://www.bxa.doc.gov)).

Traditionally, applications of usefully strong cryptography have been frustrated by export control laws and regulations administered by the U.S. government. These regulations limit the rights of U.S. companies to export products that use cryptography. Specifically forbidden are such practices as:

- Exporting software products (or other products) that have “plug-replaceable” encryption functions.
- Exporting software products that use encryption keys longer than a specified number of bits; the minimum value slowly increases as previous values become trivial to break.

Key exchange applications (use of strong encryption to encrypt and decrypt content keys, like the ones in EBX vouchers) and most digital signature applications (such as the authentication methods used in EBX) are loosely regulated if at all. The government’s motive for the regulations is to prevent strong encryption from being used by spies or other criminals, to encrypt data.

In 1999 the Clinton Administration announced its intention to relax these rules, although new regulations were not published until the turn of the century and still are being interpreted by regulators and lawyers. At this time (May 18, 2000), the net of these regulations appears to be as follows:

- A software product that is classified as both a “retail” and a “mass market” software product, according to specific government definitions, can use key lengths up to and including 64 bits.
  - Products classified this way still cannot be legally exported to certain countries, such as Serbia and Cuba.
  - Products classified this way are exempt from reporting requirements—in other words the vendor does not have to report individual shipments.
- Use of key lengths longer than 64 bits requires a specific review by the government of the software design and may also require reporting of specific exports.
- *All* products require a license if they do bulk data encryption. In other words, you cannot simply say after the fact that you use only 64 bit keys; you still need to have apply to the government and get the green light.

Some observers following this area, including some attorneys specializing in export law, think that all key length restrictions may be dropped in the future, at least for applications that are specifically aimed at copyright protection. However, that has not yet happened.

### **10.2 (U.S.) Digital Millennium Copyright Act**

The Digital Millennium Copyright Act is the U.S. law implementing the “World Intellectual Property Organization Copyright Treaty and Performances and Phonograms Treaty,” to which the U. S. A. is a signatory. The law was signed by the President of the United States and became public law 105-304 in October 1998.

The law has several implications for this specification and for copyright protection in general and is useful reading for most readers of this specification. Two implications are:

- Chapter 12 of the law states, “No person shall circumvent a technological measure that effectively controls access to a work protected under this title [this law].” With some exceptions, it is illegal in the United States (as of October 2000) to reverse engineer a copyright protection system, traffic in tools that have no other purpose than this, etc.
- The law contains a specific exception: the law does not protect activity that circumvents such a ‘technological measure’ if the motive of the circumvention is to remove “personally identifying” information from the technical protection. This section is sometimes cited as a reason to avoid inclusion of personal identifying information (such as a purchaser’s identity) in data used in the copyright protection system, data such as a digital signature.

A PDF version of the full law is available at:

([http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=105\\_cong\\_bills&docid=f:h2281enr.txt.pdf](http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=105_cong_bills&docid=f:h2281enr.txt.pdf)).

## 11 Appendix C - Content Format Requirements

[**Editorial note.** This section has not yet been edited for Version 0.8.]

EBX is designed to support multiple content formats. However, there are some minimal requirements imposed on the content format by EBX. The content format must provide the following capabilities:

- **Single Container File** - The content must be capable of transfer and storage in a single container file. This requirement is necessitated by the EBX use of HTTP for distribution, which for simplicity and efficiency, provides for the transfer of a single content object per transaction.
- **Encryption** – The content must be encryptable with a symmetric key from an EBX voucher.
- **Bibliographic Metadata** - The content format must provide for a small number of directly accessible bibliographic metadata attributes to be stored in the content file. For example, the Title, Author, Publisher, Copyright, and ISBN must be stored in the content file.
- **Scalable Text and Graphics** – The content format should provide scalable text, bit-mapped images, and, if provided at all, scalable vector graphics. This requirement is necessitated by the need for the same content to be displayable on displays of differing sizes.

The following are not required capabilities:

- **Re-flowable pages** – The content format can include either pre-paginated content or paginate-on-the-fly pages.

## 12 Appendix D - Content Format Usage Guidelines

[**Editorial note.** This section has not yet been edited for Version 0.8.]

The following sections specify how the content formats supported by EBX should be used.

### 12.1 Open eBook Format

*NOTE: This is a very rough concept of how Open eBook could be supported. Comments are gratefully encouraged.*

The Open eBook 1.0 Publication Structure Specification, Version 0.8 is the current draft document defining the Open eBook format. To quote the document:

“The Open eBook 1.0 Publication structure defines a format for content as released by the publisher ready for packaging and delivery to an end-user’s reading device. Other intermediate forms of the data may be used during delivery to the reading device; such intermediate formats are beyond the scope of this specification. In particular, secured delivery of non-public domain content requires a secure distributor, who is responsible for delivery the content in usable but protected form, perhaps using intermediate parties as appropriate.”

In other words, to provide interoperability of Open eBook publications, EBX must define an intermediate file format, encryption, etc. suitable for distribution and delivery.

#### 12.1.1 Container File Format

The first problem to be solved is to define an intermediate “container file” format for Open eBook. Open eBook requires a container file because it is based on HTML and XML, which implies that many of the components of a book (e.g., images, fonts, etc.) are contained in individual files (e.g., GIF, JPEG) and addressed by links within the HTML/XML.

One solution to this problem is MIME [RFC2045]. However, MIME is a 7-bit ASCII, stream-oriented format. In other words, it is usable as a transfer format, but to provide reasonable performance (i.e., efficient random access), it must be parsed and stored in a different format. Using the standard MIME encoding also is very inefficient because of the need to convert all data to 7-bit ASCII.

Another choice for an container file format is ZIP [ZIP98]. ZIP files provide a number of features that make it ideal as container file format:

- Compression – ZIP uses LZW compression.
- Full file path specification – Links in Open eBook typically use a full pathname.
- Random access – ZIP provides random access to the files in the ZIP file, which also is needed for efficiency.

- Tools – Tools for ZIP file handling are widely available. For example, Java includes ZIP file handling classes and PKWARE, among others, provides “C” language libraries.

The ZIP file should contain a special Manifest file named “manifest.xml” that contains an ordered list of file names for the cover page, front matter, table of contents, chapters, index, and back matter of the book. The definition of this file is T.B.S.

### 12.1.2 Encryption

One of the requirements for EBX is that the content must at all times be encrypted with a symmetric encryption key. For good interactive performance, rather than encrypting the whole container file, the individual contained files should be encrypted separately so they can be decrypted as needed.

PKZIP has defined a ZIP file encryption algorithm which provides individual file-level decryption. Unfortunately, according to some analyses [SCHNEIER96] the PKZIP encryption algorithm is “not that great”.

Therefore, the EBX encryption technique is to individually encrypt all the files and then compress them into a ZIP container file. Encrypting before compressing does not yield the best compression, but it does provide the best security.

### 12.1.3 Metadata

The Open eBook specification defines how the Dublin Core Metadata Element Set [DUBLIN97] is to be encoded in an Open eBook file. [Is it be stored in a separate XML file in the ZIP container file or embedded in the HTML?]

### 12.1.4 Display Properties

The Open eBook 1.0 specification defines that the minimum display must support 1 bit / pixel monochrome images of at least 320 x 320 pixels. Larger screens with more colors are also supported, though not defined at this time.

Properties like display resolution, size, orientation, etc. are not currently defined by the Open eBook specification. [To provide true interoperability, should they be specified in EBX?]

### 12.1.5 Font Embedding

T.B.S.

### 12.1.6 Open eBook Book Design Guidelines

When Open eBook is used as an EBX content format, the following design criteria should be used:

- ?? **Aspect Ratio** – [Does it matter?]

- **Nominal Page Size** – [Does it matter?]
- **Minimum Font Size** – To ensure easy viewing of text on the lowest resolution screen the minimum font size used in the body of the text should be 11 point. Smaller sizes can be used, but they will be difficult to read on low resolution screens.
- **Cover Page** –The ZIP file should contain a cover page named “FirstPage.html. This will be the first page displayed by the reading device. The cover page should contain a (color if possible) image of the cover of the book. Additional pages may simply be also contained in the FirstPage file, or there may be links to the additional pages.
- **Table of Contents** – The ZIP file should contain at least one file that is a table of contents. The table of contents should provide links to the pages listed in the contents. It should be named “Contents.html.
- **Index** – If the ZIP file contains an index, it should be named “Index.html.
- **Chapters** – If the book contains chapters, each chapter should be contained within a separate HTML file in the ZIP container file. The chapters should be linked-to from the Cover Page and the Table of Contents, as well as listed in the Manifest.



## 13 Appendix E - References

[**Editorial note.** This section has not yet been edited for Version 0.8.]

This specification makes use of other standards and specifications. The following are references to those documents.

[**DC11**] Dublin Core Metadata Element Set, Version 1.1: Reference Description, <http://purl.org/dc/elements/1.1/>

[**DCQ10**] Dublin Core Metadata Qualifiers Set: Reference Description, <http://purl.org/dc/qualifiers/1.0/>

[**DES93**] National Institute of Standards Data Encryption Standard, FIPS 42-6. <http://www.nist.gov/itl/div897/pubs/fip46-2.htm>

[**DOI**] Digital Object Identifier System, <http://www.doi.org/>.

[**DUBLIN97**] The Dublin Core Metadata Element Set, 1997. <http://purl.oclc.org/dc> and [http://purl.org/DC/about/element\\_set.htm](http://purl.org/DC/about/element_set.htm)

[**HMAC**] HMAC: Keyed-Hashing for Message Authentication, RFC2104. <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2104.txt>

[**HTTP1.1**] Hyper Text Transfer Protocol – HTTP 1.1, RFC2068, <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2068.txt>

[**IMT**] Internet Media Types, <http://www.isi.edu/in-notes/iana/assignments/media-types/media-types>

[**ISBN**] International Standard Book Numbering system, ISO Standard 2108.

[**MARC21**] MARC 21 Format for Bibliographic Data, <http://lcweb.loc.gov/marc/>

[**MIME**] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045. <http://info.internet.isi.edu/in-notes/rfc/files/rfc2045.txt>

[**ONIX101**] ONIX International Version 1.01, <http://www.editeur.org/onix.html>

[**PKCS95**] Public Key Cryptography Standard, RSA Laboratories, Security Dynamics, Inc. <http://www.rsa.com/rsalabs/pubs/PKCS/>

[**PDF96**] Portable Document Format Reference Manual, Version 1.2, November 1996, Adobe Systems, Inc. or Portable Document Format Reference Manual (ISBN 0-201-62628-4). <http://www.adobe.com/supportservice/devrelations/PDFS/TN/PDFSPEC.PDF>

[**POSTSCRIPT3**] Postscript 3, Adobe Systems. <http://www.adobe.com/prodindex/postscript/main.html>

[**RSA82**] RSA Public Key Crypto System, RSA Data Security Division, Security Dynamics, Inc.

[**SCHNEIER96**] Applied Cryptography, Second Edition, 1996, Bruce Schneier. ISBN 0-471-11709-9.

[**SSL96**] Secure Sockets Layer Specification 3.0, Netscape, Inc.,

[**URN97**] [rfc2141.txt](#) -- URN Syntax. R. Moats. May 1997.

[**WIPO96**] World Intellectual Property Organization Copyright Treaty, 1996.

<http://www.wipo.org/eng/diplconf/distrib/94dc.htm>

[**X509**] ITU (CCITT) Recommendation X.509: The Directory -- Authentication Framework. 1988.

[**XML98**] Extensible Markup Language, W3C, 1998. <http://www.w3.org/XML/>

[**ZIP98**] PKZIP Application Note – General Format of a ZIP File, PKWARE, 1998.

<http://www.pkware.com/appnote.html>

--

## **14 Appendix F - Edit History**

### **14.1 Version 0.1**

- First draft released to EBX Working Group 11/18/98.

### **14.2 Version 0.2**

- Added new requirements for consumers, publishers, booksellers, and libraries as discussed in 12/7/98 EBX Working Group conference call.
- Changed smart cards to be optional rather than required consumer voucher engines, as discussed in 12/7/98 EBX WG.
- Moved hardware and content format sections to appendices to indicate they are not primary parts of specification.

### **14.3 Version 0.3**

- Added additional requirements for consumers and publishers as agreed in 1/29/99 EBX WG meeting.
- Changed protocol to use HTTP instead of custom protocol as agreed in 1/29/99 EBX WG meeting.
- Changed voucher encoding from binary to XML as agreed in 1/29/99 EBX WG meeting.
- Changed book identifier to support ISBN, DOI, or other identifiers as agreed in 1/29/99 EBX WG meeting.
- Added a proposed section on Open eBook Usage Guidelines. Reworked the Postscript/PDF Usage Guidelines so both sections are parallel. Removed old Appendix B on Hardware Requirements. Changed PDF metadata model from MARC to Dublin Core.

### **14.4 Version 0.4**

- Added URN [RFC2141] as an ID type in a voucher, as agreed in 3/2/99 EBX WG meeting.
- Changed all hashing from MD5 to SHA-s1, as agreed in 3/2/99 EBX WG meeting.
- Rename “Fair Use” to “Personal Use” as discussed in 3/2/99 EBX WG meeting.
- Additional content key crypto algorithms and key lengths were added as discussed in 3/2/99 EBX WG meeting.
- The Nonce element in a Credentials object has been re-defined to be the signed nonce rather than simply the encrypted nonce. This change was recommended by a reviewer.

- Changed the query parameters in the HTTP request to agree with the discussion in the 3/2/99 EBX WG meeting. Added more description and examples as requested. Changed to all lower-case values to better match “accepted practice”.
- Changed the voucher MIC to a MAC based on the HMAC RFC because the HMAC appears to be a better-understood technology than just an encrypted hash.

#### **14.5 Version 0.5**

- Moved PDF to a separate Application Note document, at the request of the Open eBook Working Group.

#### **14.6 Version 0.6**

- Revised definitions and added roles as discussed in the EBX Working Group Technical Committee meeting held 29 February and 1 March 2000.
- Revised transfer protocol.
- Revised credentials object format.
- Added a section on ACK attacks.

#### **14.7 Version 0.7**

- Added work on the trust model completed by the Trust Model working group of the Technical Committee.
- Revised transfer protocol to make it independent of the transport protocol; converted HTTP specifications to implementation notes.
- Revised transfer protocol to move data from headers to XML bodies of requests and responses.
- Added metadata format.
- Revised voucher engine model to add EBX rules, generic operations, and versioning of basic-rights processing for compatibility with later EBX revisions.
- Removed Security Considerations section and parts of the voucher engine model that dealt with security.

#### **14.8 Version 0.8**

- In general, incorporated changes agreed to at the EBX Working Group Technical Committee meeting held 11-12 July 2000.

- Added elements to transfer data and response data to allow negotiation of authentication schemes.
- Revised credentials object format.
- Revised voucher engine interfaces, including addition of optional content encryption in the Issue Voucher operation to accommodate per-user encryption of content.
- Added proposed rights specification.