# EPUB Lightweight Content Protection

Jim Dovey, Digital Content Format Evangelist

Kobo Inc.
Toronto, ON, Canada

<jdovey@kobo.com>

September 5, 2012

**Abstract**

At present there are a number of different ePub-compatible Reading Systems being developed by different companies. Each of these is required by content publishers to implement some form of Digital Rights Management (DRM), and each usually implements either their own form of encryption and rights management. This has the side-effect of making content from one distributor usable only on that distributor's own systems, which runs contrary to the aim of a single universal standard for electronic publications such as [EPUB3]. As a result the International Digital Publishing Forum (IDPF), maintainer of the ePub standard, has issued a call for proposals on a lightweight protection system for ePub-based content which could be implemented by any reading system. This document outlines a proposed protection system designed by Kobo based on the [OCF3] format described by the IDPF.

# Contents

# Listings

# Proposal Information

## Background

Inspired by a "Read Freely" philosophy and a passion for innovation, **Kobo** is one of the world's fastest-growing eReading services. Read Freely stems from Kobo's belief that consumers should have the freedom to read any book, anytime, anyplace — and on any device. As a result, Kobo has attracted millions of readers from more than 170 countries and features one of the world's largest eReading catalogues with over 2.5 million eBook, newspaper and magazine titles.

Founded in 2009 Kobo is owned by Tokyo-based Rakuten, Japan's largest eCommerce operator. Headquartered in Toronto, our over 300 employees are proud of Kobo's top-ranked eReading applications for the iPad, iPhone, BlackBerry, Android, Windows and our own line of eReaders, including the award winning *Kobo Touch* and *Kobo Vox*.

**Open Standards Support** We believe open standards for eBooks are best for consumers, publishers, retailers and hardware manufacturers. Closed systems stifle innovation and growth. Kobo proudly supports EPUB and encourages our users to read a Kobo-purchased eBook on their smartphone, eReader, laptop, or whichever device they choose.

**Available on any device** Consumers are only just beginning to discover how they want to read digitally. Some will choose dedicated eReaders, others their smartphone or laptop. Most will choose all of the above. We support personal preference and are assembling the world's best catalogue of eBooks, no matter which device our customers use.

## The Proposed System

The system we propose is based entirely on open standards, primarily based around the ePub Open Container Format (OCF) version 3. Our proposal provides a specification for authentication and rights management described using custom XML schemas, and defines how these make use of the content signing and encryption facilities described in the OCF3 specification.

Our system has been designed from the ground up as an open standard, free from the encumbrance of commercial licensing concerns. We chose not to try and shoe-horn our

existing pre-ePub3 formats and system into an OCF3-equivalent form. Instead we decided to create a new system which builds upon the open-standards approach of OCF3 and ePub3 and makes the best use of existing technologies. When it comes time to implement this specification, we will be doing so from the same starting point as everyone else— we will "eat our own dog-food."

We additionally provide recommendations on using an *enveloped signature* for both the authentication and rights files to allow these to be signed independently of the rest of the document; this facilitates both changing authentication as part of a software-based lending process and the ability to update the `META-INF/rights.xml` file to keep an audit trail for the use of each right.

Full details are provided starting in Section 1. What follows is a brief description of the system's conformance with the requirements of the RFP.

**Requirement 1: Authentication**

Authentication takes place on a per-publication basis (satisfying {1.a}), although we anticipate that the most common authentication keys will be account details or device hardware IDs, meaning that the key is usually going to be the same across multiple publications (satisfying {1.e}). The password is always needed to open the file, but can be cached by the reading system after entry (satisfying {1.e.iv}). A one-way hashed form of the expected value is included for quick verification (satisfying {1.f}).

Our system includes means of specifying authentication keys directly from values known to a Reading System for transparent, interaction-free authentication, as well as details (satisfying {1.d}) used to request information from users directly, either as the only option or as a fallback when opened on another device (satisfying {1.c}). The authentication data is intended to hook directly into the key-retrieval specification of [XML ENC Core] or [XML DSIG Core].

The authentication system's output is a Key Encryption Key (KEK) which is then used to decrypt the real keys used to sign and/or encrypt content (satisfying {1.b}). This alleviates some potential concerns around the use of [X.509] certificates by encrypting the X.509 data within the `META-INF/encryption.xml` file, making public keys and certificate chains available only after successful authentication.

### Requirement 2: Configurable limitations on usage

Our rights system uses an extensible set of rights with a core set covering the types of requests we receive from publishers here at Kobo. This includes printing (satisfying {2.a}), copying to the clipboard (satisfying {2.b}), editing (satisfying {2.c}), quoting directly via social networks or email, and more.

It provides for audited rights (those with limits, whether per-action, lifetime, or both), full permission, and complete denial of any right. It also provides a means to exclude sections of content from the effect of a particular right limitation. Limits may be specified in a number of different units, including time ranges (satisfying {2.d}). Password protection for individual rights may be implemented by referencing an authentication mechanism within `META-INF/authentication.xml` by XML ID (satisfying {2.e}).

### Requirement 3: Content protection

We advocate the use of only well-known encryption standards: public/private key systems or an AES symmetric cipher for encryption and signing (satisfying {3.a, 3.b}). While we recommend the use of AES key wrapping as defined in [RFC3394] to protect the encryption keys, any algorithm supported by [XML ENC Core] is acceptable (satisfying {3.c, 3.d}).

Key revocation and renewability is only supported for the wrapped content protection keys when they are [X.509] key pairs; in this instance, a Certificate Revocation List (CRL) may be included in the X.509 eta packet within `META-INF/encryption.xml` (satisfying {3.e}).

We anticipate that encryption and signing will take place in a single pass, so that the obfuscated authentication or content protection keys may be embedded as watermarks in the content before being applied. We take no position on the design of the back-end systems implementing this, and assume that the watermarking tool will be able to obtain this information (satisfying {3.f}).

### Requirement 4: Client-server interactions

Our proposal requires that all information necessary to obtain authentication and thus be able to decrypt the content protection keys be embedded within a publication at production time (satisfying {4.c}). When a client for a service such as Kobo downloads a publication, the server is expected to ensure that all information is embedded correctly. The

client may provide a unique hardware ID while connecting to support authentication based on the device (satisfying {4.a, 4.b})

**Requirement 5: Format support**

Our system is designed purely as an extension to the [OCF3] specification, version 3.0 (satisfying {5.a}).

**Requirement 6: Code library**

No code library currently exists, though our use of existing encryption [XML ENC Core] and signing [XML DSIG Core] technologies, paired with a simple XML schema for authentication and rights management, makes our proposed system easy to implement. Standard implementations of XML encryption and digital signing exist for most major languages; in particular, there are implementations for C (and thus C++ or Objective-C), Java, and Ruby, along with bindings to the C libraries for python and other languages. Thus the burden of implementation lies primarily on the parsing of `META-INF/rights.xml` and `META-INF/authentication.xml`.

**Requirement 7: Cost elements**

As noted above, libraries to handle the difficult tasks of encryption and signing are widely available. Our use of simple XML schemas should provide little friction to implementors' efforts. We are prepared to offer up open-source implementations of this specification for commonly-used platforms (satisfying {7.a, 7.c}).

We envision a fairly simple root-of trust hierarchy based around standard certificates. An example would have a group such as the IDPF hold a root certificate from which they provide certificates to publishers and vendors. Vendors might then generate certificates for each account to use for encryption/signature purposes, using a CRL within the `META-INF/encryption.xml` file to distribute revocations of compromised certificates (satisfying {7.b})

Additionally, as Kobo is dedicated to the use and propagation of open standards in the eReading market, we do not intend to require any licenses for the use of technologies described in this specification (satisfying {7.e}). Use of only widely-implemented standards also decreases risks of IP overhang from other sectors (satisfying {7.f}).

# 1 Preamble

This specfication introduces a system of digitally signing and encrypting Electronic Publication (ePub) documents, along with a method and schema for describing available user-authentication mechanisms and DRM usage limitations. It builds on standards and specifications already developed and with readily-available implementations fir a number of platforms, primarily the [OCF3] specification from the IDPF and the [XML DSIG Core] and [XML ENC Core] specifications from the World Wide Web Consortium (W3C).

This specification is organized in six sections:

1. This preamble.
2. A high-level description of the techniques and technologies proposed.
3. Implementation details of the proposed XML documents.
4. Enumeration of all Required, Recommended, and Optional components.
5. Pseudocode detailing the required processing steps.
6. Full listings of the proposed XML schemas.

## 1.1 Editorial and Conformance Conventions

The specification provides normative XML Schemas [XMLSCHEMA-1] [XMLSCHEMA-2] for two types of XML data. The full normative grammar is defined by the XSD schemas and the normative text of this specification. In cases of discrepancy between the standalone XSD schemas and the portions of them presented in this specification, the standalone schema is to be considered authoritative.

The key words "Must", "Must Not", "Required", "Shall", "Shall Not", "Should", "Should Not", "Recommended", "May", and "Optional" in this document are to be interpreted as described in [RFC2119]:

> "They Must only be used where it is actually required for interoperation or to limit behaviour which has potential for causing harm (e.g., limiting retransmissions)"

We use these capitalized terms to indicate options and requirements which would affect the interoperability and security of conforming implementations. These words are not used when describing optional parts of e.g. XML grammar or DRM feature implementation, but will call out items of important consideration to implementors. For example, it is optional whether a reading system presents signature, encryption, or DRM details to a user, but the system's compliance with the Namespaces in XML specification [XMLNS] is REQUIRED.

## 1.2 Namespaces and Identifiers

This specification makes use of XML namespaces, and uses the Uniform Resource Identifier (URI) format to identify resources, identities, and mechanisms.

Implementations of this specification MUST use the following XML namespace URIs:

| URI | Namespace Prefix |
|---|---|
| `http://www.idpf.org/epub/30/lcp-auth#` | *default namespace*, `auth:` |
| `http://www.idpf.org/epub/30/lcp-rights#` | *default namespace*, `rights:` |

Implementations of this proposal MUST support XML namespaces. The namespace prefixes shown above however are OPTIONAL and are only used in this document for illustrative purposes. In the interest of brevity we also define the following XML internal entities for use in this document:

```
<!ENTITY auth ”http://www.idfp.org/epub/30/lcp-auth#”>
<!ENTITY rights ”http://www.idfp.org/epub/30/lcp-rights#”>
```

Additionally, this specification uses elements and algorithms from the XML Signature [XML DSIG Core] and XML Encryption [XML ENC Core] namespaces, with the corresponding XML namespaces and internal entities:

| URI | Namespace Prefix |
|---|---|
| `http://www.w3.org/2000/09/xmldsig#` | `ds:` |
| `http://www.w3.org/2009/xmldsig11#` | `ds11:` |
| `”http://www.w3.org/2002/06/xmldsig-filter2”` | `ds-xpath:` |
| `http://www.w3.org/2001/04/xmlenc#` | `xenc:` |
| `http://www.w3.org/2009/xmienc11#` | `xenc11:` |

```
<!ENTITY dsig ”http://www.w3.org/2000/09/xmldsig#”>
<!ENTITY dsig11 ”http://www.w3.org/2009/xmldsig11#”>
<!ENTITY ds-xpath ”http://www.w3.org/2002/06/xmldsig-filter2”>
<!ENTITY xenc ”http://www.w3.org/2001/04/xmlenc#”>
<!ENTITY xenc11 ”http://www.w3.org/2009/xmlenc11#”>
```

# 2 Overview

This section will describe the primary components of our proposed content protection system. Here you will find discussions of authentication, encryption, digital signing, and the different usage limitations supported by the proposed standard. In each area, references will be given to the location of more detailed information.

## 2.1 Authentication

Authentication is defined here as the process by which a key[1] is obtained which can be used to decrypt and/or verify the content of a publication. Authentication makes use of features described by both the [XML DSIG Core] and [XML ENC Core] specifications. The content signing and encryption facilities are described here in Sections 2.3 and 2.4 respectively.

The key obtained via authentication is used as a KEK. This is then used to decrypt a key used for signing and/or encrypting the content. The encrypted key would be stored within either or both of `META-INF/signature.xml` or `META-INF/encryption.xml` through the use of the `xenc:EncryptedKey` defined in Section 2.2.2 of [XML ENC Core]. It is RECOMMENDED that it be stored only in one place, with a `ds:RetrievalMethod` tag referencing it in any other locations the key might be used.

In our proposal, we view authentication as something which should not need to directly involve user action. For publications acquired from a distributor such as Kobo, we anticipate the password[2] would be some piece of user-specific information such as that user's account name or email. This proposal does not place any limitations on the nature of this value, but it is RECOMMENDED that a more personal value such as an email address be used wherever possible; there should be some level of reticence on the part of the user to disseminate this information widely.

For items which are created independently of a large content-manufacturing system, this proposal includes an authentication scheme based solely upon a user-enterable password. Unfortunately no means to specify that a password should be requested from the current user is defined by the XML Digital Signature or XML Encryption specifications; for that reason we describe the use of a `ds:RetrievalMethod` tag whose `URI` parameter directs

[1] While we use the word 'key' frequently within this proposal, note that in terms of authentication of public-key cryptography the computed value is likely to be used as an *input vector* or *salt* value.

[2] Where by 'password' we in fact mean *shared secret* in the cryptographic sense.

the parser to a node within the ePub interface which defines this information. Details and schema information can be found in Section 3.3.

XML authentication files are represented by the following structure (where "?" denotes zero or one occurrence; "+" denotes one or more occurrences; and "*" denotes zero or more occurrences):

```
<Authentication ID?>
   (<Mechanism ID?>
        <AuthInfo ID?>
           (<IdentityKey ID? />)?
           (<ds:Transforms>)?
           (<ConfirmationValue ID?>
                <ds:DigestMethod>
                <ds:DigestValue>
             </ConfirmationValue>)?
        </AuthInfo>
       (<Prompt>)?
       (<Hint>)?
    </Mechanism>)+
</Authentication>
```

### 2.1.1 Authentication Mechanisms

The [OCF3] specification describes a number of files useful for recording metadata associated with a publication. While the `META-INF/encryption.xml`, `META-INF/signature.xml`, and `META-INF/rights.xml` files all have their place in the protocol we define, the task of specifying a means of *authentication* in particular doesn't quite match with the files defined by [OCF3]. We could place the information into the `META-INF/metadata.xml` file, but capabilities such as sharing or lending might require updates (and therefore re-signing) of that file, which doesn't really gel with the idea of keeping authentication information sacrosanct. It is in regards to that, then, that we propose a new file to specify authentication data, namely `META-INF/authentication.xml`.

A number of different protocols are defined here, with their details enumerated in Section 3. There are two primary avenues of authentication suggested: *transparent* and *interactive*. As discussed below, these are not mutually exclusive: a content provider might provide an interactive authentication mechanism based on a password while defining a means for prepared devices to obtain via a built-in certificate chain the same password.

**Transparent Authentication** A transparent authentication mechanism is one which does not (or at least *should* not) involve any user interaction beyond an intent to access the protected publication. It is anticipated that content providers that define their own reading systems should be able to utilize a key-wrapping algorithm for which their own reading systems hold a decryption key in advance. The reading system would therefore be able to unwrap a provided content key to perform authentication of the content and the reader without requiring input from the user.

**Interactive Authentication** When no transparent authentication protocol is suitable, or when content is accessed outside of a realm for which a transparent authentication mechanism is defined, the user must be prompted to enter a password or similar secret value. This can then be used to unwrap keys used for the decryption or verification of content. The type of authentication required, along with any messages or hints to be displayed, is encoded within the `META-INF/authentication.xml` file using the `Mechanism` and `AuthInfo` tags described in Section 3.1.2 and Section 3.1.3 respectively.

In either case, the end result of the authentication process is the generation of a content key which is then used to verify the cryptographic signature of the content and if necessary decrypt it.

The mechanisms defined in this proposal include:

- *Device Keyed*
  The authentication is keyed to a device-specific value, such as a MAC address or other hardware identifier. A secure hash of the expected value may be provided to verify the chosen key; this feature is Optional.

- *Account Keyed*
  The authentication makes use of a user-specific value such as an email address or account name; if a vendor should keep a user's password in recoverable form[3], the key might well be the user's account password.

- *User Input*
  The user is required to enter a key value which is not recorded anywhere in the content. This type May include a hashed form of the key or a known value (for example, the publication's identifier) hashed by the key to use as verification; however it is anticipated that the document will at least be digitally signed using the

[3]We make no recommendations as to the advisability of this approach; we merely note its possibility.

same key, so standard signature verification could be performed to ascertain the validity of the input value.

In the list above, the *device keyed* mechanism is suitable only for transparent authentication, the *user input* mechanism requires interactive authentication, and the *account keyed* mechanism is suitable for either approach. Implementation of all three mechanisms is REQUIRED.

The following example shows how authentication can be keyed to a single reader device using the MAC address of its primary network interface.

```
<?xml version="1.0" encoding="utf-8"?>
<Authentication xmlns="http://www.idpf.org/epub/30/lcp-auth#" xmlns:ds="↩
      http://www.w3.org/2000/09/xmldsig#">
  <Mechanism Type="http://www.idpf.org/epub/30/lcp-auth#device-key">
    <AuthInfo Type="http://www.idpf.org/epub/30/lcp-auth#mac-address">
      <ds:Transforms>
        <!-- MAC address string should include separator characters -->
        <ds:Transform Algorithm="http://www.idpf.org/epub/30/lcp-auth#↩
              with-separators" />
        <!-- All letters should be lower-case -->
        <ds:Transform Algorithm="http://www.idpf.org/epub/30/lcp-auth#↩
              lowercase" />
      </ds:Transforms>
      <ConfirmationValue>
        <!-- SHA-256 digest of the expected value is included for ↩
              confirmation purposes -->
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#↩
              sha256" />
        <ds:DigestValue>rSXgNiiV4wv7PhcMSWPa37W3pSlYDXAHSFqQccplGIc=</↩
              ds:DigestValue>
      </ConfirmationValue>
    </AuthInfo>
  </Mechanism>
</Authentication>
```

One mechanism can provide a URI referencing a fallback mechanism to use when processing fails. The example below attempts to transparently authenticate the user by their account email; if the correct user is not signed in this will fail, causing the reading system to prompt the user to enter the correct value manually.

```
<?xml version="1.0" encoding="utf-8"?>
<Authentication xmlns="http://www.idpf.org/epub/30/lcp-auth#" xmlns:ds="↩
      http://www.w3.org/2000/09/xmldsig#">
  <!-- If this mechanism fails, the Next attribute selects an ↩
       alternative mechanism to try -->
  <Mechanism Type="http://www.idpf.org/epub/30/lcp-auth#account-key" ↩
       Next="#User">
    <AuthInfo Type="http://www.idpf.org/epub/30/lcp-auth#account-email">
      <ds:Transforms>
        <!-- Ensure email contains only lowercase letters -->
        <ds:Transform Algorithm="http://www.idpf.org/epub/30/lcp-auth#↩
             lowercase" />
      </ds:Transforms>
      </IdentityKey>
      <ConfirmationValue>
        <!-- SHA-256 digest of the expected value -->
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#↩
             sha256" />
        <ds:DigestValue>BSOEZHaZcytkRLoXrWiuwBhznDrC7o=</ds:DigestValue>
      </ConfirmationValue>
    </AuthInfo>
  </Mechanism>
  <Mechanism Id="User" Type="http://www.idpf.org/epub/30/lcp-auth#user-↩
       input">
    <!-- The Prompt value contains text to be shown e.g. in an input ↩
         dialog -->
    <Prompt>Please enter the email address associated with your Kobo ↩
         account.</Prompt>
    <!-- The Hint value provides a small hint to the user: in this case,↩
          the account user name -->
    <Hint>Jim Dovey</Hint>
    <AuthInfo>
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.idpf.org/epub/30/lcp-auth#↩
             lowercase" />
      </ds:Transforms>
      <ConfirmationValue>
        <!-- Confirmation value is the same here as above: we're ↩
             requesting the same email -->
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#↩
             sha256" />
```

```
        <ds:DigestValue>BSOEZHaZcytkRLoXrWiuwBhznDrC7o=</ds:DigestValue>
      </ConfirmationValue>
    </AuthInfo>
  </Mechanism>
</Authentication>
```

It is also possible to define multi-stage authentication using this schema. The `Mechanism` element's `Append` attribute is used to specify an additional mechanism whose output value is then appended to the ultimate result. For instance, a key may be the concatenation of a device serial number and a use's account email, as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<Authentication xmlns="http://www.idpf.org/epub/30/lcp-auth#" xmlns:ds="↩
      http://www.w3.org/2000/09/xmldsig#">
  <!-- If this mechanism succeeds, the mechanism identified by the ↩
       Append attribute runs -->
  <Mechanism Type="http://www.idpf.org/epub/30/lcp-auth#device-key" ↩
       Append="#Email">
    <AuthInfo Type="http://www.idpf.org/epub/30/lcp-auth#serial-number">
      <ds:Transforms>
        <!-- All letters should be lower-case -->
        <ds:Transform Algorithm="http://www.idpf.org/epub/30/lcp-auth#↩
             uppercase" />
      </ds:Transforms>
      <ConfirmationValue>
        <!-- SHA-256 digest of the expected output is included for ↩
             confirmation purposes -->
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#↩
             sha256" />
        <ds:DigestValue>EqkeSbvtX/0BfHAqYH3Xj+KFg2Lxpn84rEX8DXPKjHs=</↩
             ds:DigestValue>
      </ConfirmationValue>
    </AuthInfo>
  </Mechanism>
  <Mechanism Id="Email" Type="http://www.idpf.org/epub/30/lcp-auth#↩
       account-key">
    <AuthInfo Type="http://www.idpf.org/epub/30/lcp-auth#account-email">
      <ds:Transforms>
        <!-- Ensure email contains only lowercase letters -->
        <ds:Transform Algorithm="http://www.idpf.org/epub/30/lcp-auth#↩
```

```
            lowercase" />
      </ds:Transforms>
      </IdentityKey>
      <ConfirmationValue>
        <!-- SHA-256 digest of the expected value -->
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#↩
            sha256" />
        <ds:DigestValue>BSOEZHaZcytkRLoXrWiuwBhznDrC7o=</ds:DigestValue>
      </ConfirmationValue>
    </AuthInfo>
  </Mechanism>
```

More complicated authentication schemes can be implemented using a combination of these techniques. For instance, a user can be transparently authenticated by a device ID plus account email, and if this fails they can be asked to input the correct email and/or a password value, such as the account password associated with the email. Implementations MAY choose to pre-emptively evaluate appended mechanisms and present multiple user-input mechanisms in a single pass, such as a dialog asking for both email and password.

If the DRM settings for a publication allow for sharing or lending of content, then the contents of `META-INF/authentication.xml` may be signed using an *enveloped signature* as defined in Section 6.6.4 of [XML DSIG Core]. Signing the authentication file in-place rather than through `META-INF/signature.xml` enables a book to be given a new authentication key for the purpose of lending to another reader. This is discussed in more detail in Section 2.5.1.

Detailed information and XML schemas for `authentication.xml` are presented in Section 3, with requirements enumerated in Section A.1.

## 2.2 Certificate Chains and Root of Trust

In the following two sections we discuss the signing and encryption of content. It is our recommendation that publishing houses and distributors use [X.509] key-pairs for these tasks. This section suggests an infrastructure for this.

We envision that a global entity such as the IDPF obtain a root certificate as a Certificate Authority. Upon application, they would then provide similar CA certificates to publishers and distributors as children of their own. In this way, the IDPF would be able to

revoke those certificates. Distributors and publishers would then create leaf signing/encryption certificates for each account/employee from their CA certificates, which would be used for signing and publishing.

Signatures would be implemented by using the private key associated with a publisher/distributor to digitally sign a publication; the certificate would be included (protected by the authentication KEK), with the signature being verified using the publisher's public key.

Encryption, on the other hand, would work the other way. A user's certificate would be included with the bundled certificate chain along with the private key associated with it. The content would be encrypted by the publisher with the user's public key, and the private key would be used to decrypt it.

Should a particular user's keys or certificate become compromised, the publisher/distributor can revoke that certificate, disseminating the CRL by embedding it in any books it publishes or through any other means deemed appropriate.

## 2.3 Content Signing

Signing of content is implemented according to [XML DSIG Core] and is assembled as outlined in Section 2.5.6 of the [OCF3] specification, placing its information within `META-INF/signature.xml`.

The content as a whole is digitally signed by making secure hashes of each protected content file— ideally the entire contents of the publication, but only items referenced by the package manifest [OPF3] are required to be included. Files outside of the `META-INF` subfolder of the publication container are processed as simple octet-streams, regardless their actual content format. Within the `META-INF` folder, all XML files are processed with the exception of `signature.xml`; inclusion of these files is Required. These files are additionally canonicalized prior to processing using XML Canonicalization; use of version 1.1 [XML C14N] of the specification is Recommended.

The signature file is structured as suggested in the OCF 3.0 specification, based on the [XML DSIG Core] specification. The digests computed for each content file are stored within the `ds:Manifest` node in `ds:Reference` nodes. The `ds:Manifest` node is then hashed and its digest stored within the `ds:SignedInfo` node. This is then cryptographically signed, with the resulting value placed in the `ds:SignatureValue` node.

Details of how to obtain the key used to verify the signature are stored within the `ds:KeyInfo` node. This may contain any content described in [XML DSIG Core], but it is anticipated

that a ds:RetrievalMethod subnode would be provided with a URI referencing an AuthInfo node within META-INF/authentication.xml. An example of this can be seen below.

```
<ds:KeyInfo>
  <ds:RetrievalMethod URI="./authentication.xml#KEK" />
</ds:KeyInfo>
```

As specified in [XML DSIG Core], the result of evaluating the ds:RetrievalMethod is an XML node. However, since the returned node is within the auth: namespacue, it should be processed using that namespace's processing rules to obtain the resultant key.

If a signature cannot be verified, it is REQUIRED that the reading system alert the user to this fact. It is RECOMMENDED that the publication is not displayed at all, but a reading system MAY offer the user the opportunity to do so; note that if the content is additionally protected by encryption it will not be viewable at all.

### 2.3.1 Nested Signatures

There are potentially two levels at which content might be digitally signed. The most obvious is to have the distributor sign the package that they provide to end users. However, given that some ePub documents produced by publishers sometimes need an extra layer of transliteration before they are distributed—including correction and canonicalization of some elements of the content documents themselves—there exists the possibility of another signature provided by the content publisher. There two signatures would be used to verify two separate facts:

- *Distributed Content Verification* would utilize a signature on the data provided by the distributor. This would verify that the content has not been altered from the version shipped to a customer, including alterations to distributor content such as platform-specific metadata or DRMinformation.
- *Authored Content Verification* involves a signature of the content as provided by the publisher, and would verify that the actual text and assets of the publication have not been altered.

When verifying an author/publisher signature, an [XPath-Filter] can be used to reverse the changes made by the distributor. One example, which removes any content elements

whose `id` attribute begins with the string ‘kobo’ while retaining the children of any `span` nodes with such an `id`, would look like this:

```
<ds:Transform Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
  <ds-xpath:XPath Filter="subtract" xmlns="http://www.w3.org/2002/06/↩
        xmldsig-filter2">
    id("kobo.*")
  </ds-xpath:XPath>
  <ds-xpath Filter="intersect" xmlns="http://www.w3.org/2002/06/xmldsig-↩
        filter2">
    //span[@id="kobo.*"]/*
  </ds-xpath:XPath>
</Transform>
```

This transform could then be applied to the following XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>My Great Book</title>
  <link rel="stylesheet" href="kobo/kobo-stylesheet.css" type="text/css"↩
        id="kobo.css.1" />
</head>
<body>
  <p><span id="kobo.1.1">First sentence.</span> <span id="kobo.1.2">↩
        Second sentence.</span></p>
</body>
</html>
```

The resulting author document for signature verification is then:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>My Great Book</title>
</head>
<body>
```

```
  <p>First sentence. Second sentence.</p>
</body>
</html>
```

## 2.4 Content Encryption

There will be two forms of encryption involved in this proposal. For non-modifiable content such as purchased books (arguably the majority of the market) an asymmetric public/private key pair will be at the core of both the encryption and digital signature algorithms. For editable content a simpler symmetric password system may be used, operating in a manner similar to that common on password-protected PDF and Zip files. This would enable the user to re-encrypt the file, however see the comments on watermarking in Section 2.5.3 for some additional requirements when dealing with edited files.

We deem it useful to be able to determine that a publication was signed or encrypted by a particular entity. For that reason the use of [X.509] certificates is RECOMMENDED with content being encrypted using a private key associated with an attached certificate. The X.509 data describing the corresponding certificate chain MAY itself be encrypted using the key-wrapping algorithms described in Section 2.1. By using certificate-based encryption in this way a reading system can use the certificate chain to determine the original provider of the content.

Additionally, since [XML ENC Core] provides a means of including a CRL within a `ds:KeyInfo` node it is possible for a compromised certificate to be revoked purely by opening a protected publication. Reading systems can then direct their legitimate users to download a new version of the publication through the appropriate channel, which again can be inferred from the certificate (or directly encoded therein).

The encryption information is specified in `META-INF/encryption.xml` as outlined in Section 2.5.2 of the [OCF3] specification; this in turn specifies the use of [XML ENC Core]. Our specification for file encryption uses the OCF encryption model unchanged. This includes the stipulation that files MUST be compressed using the [Deflate] algorithm prior to encryption, then MUST be placed within the ZIP container in non-compressed format.

It is intended that the encryption key be wrapped using the key obtained through authentication. As a result, the `ds:KeyInfo` node is expected to direct the reading system to the authentication information in `META-INF/authentication.xml` in a similar manner to digital signatures, as described in Section 2.3.

## 2.5 Rights Management

Rights management is an issue close to the hearts of many publishers. The ease with which digital data can be transferred and disseminated greatly exceeds the means to do the same with physical publications. Likewise the medium by which digital publications are displayed often makes it easy to copy large swathes of text, which would be a laborious operation when done by hand.

Some common requests from publishers include:

- Watermarking publications with customer details— similar to the approach taken by Pottermore.
- A limit on printing content from the publication.
- A limit on the sharing of text using social media channels or similar. This can be in the form of a character count per share or a percentage limit on the amount of the book's contents that can be shared in total (for instance 5%).
- A limit on which sections of a publication allow any form of sharing; for instance sharing of text from the last chapter of a novel might be restricted to prevent its resolution being exposed.
- For libraries, a limit on the amount of time the file is accessible (or 'checked out').
- The total number of times book can be lent to another person.
- The number of times a book can be simultaneously lent out.
- A limit on the number of devices on which a book can be read. This is beyond the scope of this proposal, however, since it would require phoning-home on each new device, which is prohibited by the scope of the ePub LCP specification.

With regards to a limit on the number of devices, this is largely intended as a function of a particular reading system's available devices, such as the Kindle devices or Kobo's various applications and eReader platforms. As such, it is anticipated that device limits are to be applied only to particular devices made by the publication's distributor rather than holistically. Where differing platforms are involved it is our belief that the restrictions on lending and other forms of dissemination are enough to discourage (or at least litigate against) widespread copyright infringement.

We have defined an XML schema for the `META-INF/rights.xml` file which covers all the use cases above, with the exception of the device-count limitation. Our schema is, however, extendable by distributors so that they could provide information suitable for that purpose along with any others specific to their systems. We have also included a means for an individual right to require authorization. To do so, the right provides a URI to a mechanism within `META-INF/authentication.xml` which describes the means by which the authentication should take place.

XML rights management files are represented by the following structure (where "?" denotes zero or one occurrence; "+" denotes one or more occurrences; and "*" denotes zero or more occurrences):

```
<Rights>
  (<Right ID?>
    (<Authorization ID? />)?
    (<ExcludedContent>
      (<Manifest ID?>)*
      (<Fragment ID?>)*
     </ExcludedContent>)?
     (<LifetimeLimit>)?
      <Status>(Permitted|Audited|Denied)</Status>
     (<EligibilityPeriod ID?>)?
     (<Consumption>
       (<UseInfo ID?>
          <Timestamp>
         (<Amount>)+
        </UseInfo>)?
      </Consumption>)*
   </Right>)+
  (<SharingInfo ID?>
     <MaxDuration>
    (<MinDuration>)?
    (<LifetimeLimit>)?
    (<Consumption>)*
</Rights>
```

Our system of identifying individual rights mirrors that used for specifying algorithms in [XML DSIG Core]. A right is specified using a URI value based on the XML namespace URI for the rights format, e.g. `http://www.idpf.org/epub/30/rights#print`. The full list is enumerated in Section 4.2. Implementations of this proposal are REQUIRED to rec-

ognize all rights specified, although their implementation of restrictions would naturally be itself limited by the capabilities of the reading system.

### 2.5.1 Usage Information

Where individual rights are enumerated, they are given a *status*. This may be one of:

- *Permitted* to denote that a right is unrestricted.
- *Denied* to denote that a right is unconditionally prohibited.
- *Audited* to denote that a right has some limits on either per-action or lifetime uses.

Any rights not represented within `META-INF/rights.xml` MUST be implicitly allowed; it is RECOMMENDED, however, that any known permitted rights be included with an explicit status of *Permitted*.

It is possible to exclude individual content documents (or ranges of content documents) from the effects of a right specification. Entire documents are referenced by their manifest item's `id` attribute value, and ranges within a document can be specified using a ranged [EPUB-CFI]. Content referenced in this way inverts the status of the right: *Permitted* and *Audited* rights are *Denied* within excluded content, while *Denied* rights become *Permitted*. This can be used to prevent copying or sharing of content from the closing stages of a novel, for instance, or it can be used to create previews by denying access to all chapters but the first. In the latter case, it makes it possible to implement a preview based on the number of paragraphs rather than full chapters; this can be useful for content where chapters are either inordinately long or short, or where a plain percentage results in a preview the size of many commercial books.[4]

Rights may also be restricted using times and dates. This is most obviously useful for libraries, where access to a publication (its *read right*) can be limited between an explicit pair of dates. It could also be used to create embargoes on review copies, preventing copying or sharing until a predetermined 'street date.'

Personal lending is, by default, *Denied*. To enable lending, a `SharingInfo` element is used to provide information on the limits within which a publication may be lent to other

[4]Case in point: *Under The Dome* by Stephen King, where the first 10% of the book is several hundred pages in length on an iPhone.

readers.[5] This can be used to specify maximum (and minimum!) loan periods, and the number of times a copy may be loaned out at one time. It also allows the specification of *exclusive* lending, where the original reader is unable to access a publication while it is lent out.

Lastly, we have defined the means for implementations to record auditing information for all rights, essentially logging their use within the `META-INF/rights.xml` file; this applies both to rights and sharing information. Since this modifies this file, it is RECOMMENDED that the file be signed using an *enveloped signature* to keep its integrity verifiable. An additional benefit of using an enveloped signature is that replacement copies of `META-INF/rights.xml` and `META-INF/signature.xml` may be substituted directly into an existing publication. This would enable unlocking of the preview described above without the transfer of a whole new publication but only a new set of access rights and the associated signature file.

Full details of the rights management system are provided in Section 4.

### 2.5.2 Watermarking

Watermarks are a highly-requested feature, particularly as publishers begin to recognize the burden of maintaining a secure form of use restrictions in the form of 'traditional' DRM. There are many formats such watermarking might take: Pottermore, for example, edits the name of each content file to include a watermark.

The chief use of a digital watermark is to identify a single purchaser. If a publication is then decrypted and stripped of protection and disseminated widely, the watermark can be used to determine the originator of this copy. Watermark values can be a number of different things, but our recommendation, based on the certificate chain outlined in Section 2.2, would be to include the *Subject Name* details of the user account's certificate in base-64 format along with a base-64-encoded signature (using the publisher's private key) of that value. This would then be inserted into the content flies as appropriate for the file types: as an HTML `meta` element, as custom EXIF data in images, or custom container metadata for audio or video files.

As an example, a signature might be encoded in the following manner:

**procedure** GENERATEWATERMARK($user$, $key$)
    $user\_info \leftarrow$ GETSUBJECT($user$)  ▷ "`CN=Kobo Reader: Bob,ON=Bob,O=Bob`"
    $base64 \leftarrow$ BASE64OF($user\_info$)

[5] Note that this only applies to *personal* lending, not *library* lending.

$signed \leftarrow$ Sign($base64$, $key$)
$sig \leftarrow$ Base64Of($signed$)
**return** $base64 + sig$
**end procedure**

### 2.5.3 Editing

When dealing with editable publications, it is Recommended that the authentication mechanism used to access the KEK is a simple password, as this fits more into the use cases for shared, collaboratively edited content.

For collaborative editing within organizations, we recommend that each user use their S/MIME signing certificate or PGP key to sign the document after an edit is made; Additionally, the *edit* right can be audited (with a limit of '100%') to enable software to automatically apply an audit trail within `META-INF/rights.xml`.

When watermarking editable content, we suggest use of the name of the last editor as a watermark added to each file they edited.

### 2.5.4 Examples

Some example `rights.xml` files demonstrating different use cases are included below.

This example shows a few different rights of varying types.

```
<?xml version="1.0" encoding="utf-8"?>
<Rights xmlns="http://idpf.org/epub/lcp-rights#">
    <!-- Printing limitation: 2 pages at a time, up to a maximum of ↩
          thirty pages total -->
    <Right Unit="page" Type="http://www.idpf.org/epub/30/lcp-rights#↩
          print" Limit="2">
        <LifetimeLimit Unit="page">30</LifetimeLimit>
        <Status>Audited</Status>
        <Consumption>
            <!-- none used yet -->
        </Consumption>
    </Right>
    <!-- Social Quoting limitation: 80 characters per quote, max 5% of ↩
          book -->
```

```
    <Right Unit="characters" Type="http://www.idpf.org/epub/30/lcp-↩
          rights#social-share" Limit="80">
        <ExcludedContent xml:base="../content.opf">
            <!-- exclude the entire last chapter from being shareable ↩
                 with some shorthand -->
            <ManifestItem IdRef="chap22" />
            <!-- exclude the Big Reveal in the penultimate chapter using↩
                  a Range CFI -->
            <Fragment CFI="epubcfi(/6/4[chap21ref]!/4[body01]/10[para05↩
                 ],/2/1:1,/8/4:57)" />
        </ExcludedContent>
        <LifetimeLimit Unit="percentage">5</LifetimeLimit>
        <Status>Audited</Status>
        <Consumption>
            <!-- one share, 42 characters, 0.03% of total -->
            <UseInfo Id="Social1">
                <Timestamp>2012-08-28Z12:54:32.0</Timestamp>
                <Amount Unit="character">42</Amount>
                <Amount Unit="percentage">0.03</Amount>
            </UseInfo>
        </Consumption>
    </Right>
</Rights>
```

This example shows how a library loan might be implemented using the `EligibilityPeriod` element.

```
<?xml version="1.0" encoding="utf-8"?>
<Rights xmlns="http://idpf.org/epub/lcp-rights#">
    <!-- Reading limitation: disallows access to content outside META-↩
          INF or the OPF file(s) -->
    <Right Unit="time" Type="http://www.idpf.org/epub/30/lcp-rights#read↩
          ">
        <Status>Audited</Status>
        <EligibilityPeriod Id="Loan-2012-08-28">
            <Start>2012-08-28Z00:00:00.0</Start>
            <End>2012-09-12:23:59:59.99</End>
        </EligibilityPeriod>
    </Right>
</Rights>
```

Ability to lend would be enabled through use of the `SharingInfo` element as shown here. Note that if a the `SharingInfo` element is not present, implementations MUST assume that lending of content is **prohibited**.

```
<?xml version="1.0" encoding="utf-8"?>
<Rights xmlns="http://idpf.org/epub/lcp-rights#">
    <!-- This book can be lent for no more than seven days (168 hours), ↩
          and cannot be read by me until returned -->
    <SharingInfo Exclusive="true" Domain="http://www.idpf.org/epub/lcp-↩
          rights#open">
        <MaxDuration>168:00:00:0.0</MaxDuration>
    </SharingInfo>
</Rights>
```

# 3 Authentication Protocol Details

In this section we provide normative details of the authentication schema and data. Here you will find discussions of each element defined by the schema along with instructions on their use. This is followed with details of the URI specifiers for the various authentication types and the additional content transforms presented by this specification.

## 3.1 Authentication Core Syntax

This section provides detailed information on the syntax of the authentication data provided within `META-INF/authentication.xml`. All features described here are mandatory unless noted otherwise. The syntax is defined via [XMLSCHEMA-1] [XMLSCHEMA-2] with the following XML preamble, declaration, and internal entity:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.↩
      w3.org/2001/XMLSchema.dtd"
[
 <!ATTLIST schema xmlns:auth CDATA #FIXED "http://www.idpf.org/epub/30/↩
       lcp-auth#">
 <!ATTLIST schema xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/↩
       xmldsig#">
 <!ENTITY auth "http://www.idpf.org/epub/30/lcp-auth#">
 <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
 <!ENTITY % p "">
 <!ENTITY % s "">
]>
<schema xmlns:auth="http://www.idpf.org/epub/30/lcp-auth#" xmlns:ds="↩
      http://www.w3.org/2000/09/xmldsig#" targetNamespace="http://www.↩
      idpf.org/epub/30/lcp-auth#" version="0.1" elementFormDefault="↩
      qualified">
```

### 3.1.1 The `Authentication` Element

The `Authentication` element is the root of the `META-INF/authentication.xml` file. It MUST contain only `Mechanism` elements, and all mechanisms must evaluate to the same end authentication key. The implementation MUST conform to the following schema.

```
<element name="Authentication" type="auth:AuthenticationType"/>
<complexType name="AuthenticationType">
  <sequence>
    <element ref="ds:Signature"/>
    <element ref="auth:Mechanism" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Provider" type="anyURI" use="optional"/>
</complexType>
```

### 3.1.2 The `Mechanism` Element

The `Mechanism` element is the root of a single authentication mechanism description. It MUST contain a single `AuthInfo` element to provide information on authentication to the Reading System, and MAY include either a `Prompt` element, a `Hint` element, or both, containing information about the authentication in a form suitable to present to the user.

The three REQUIRED mechanisms are specified through URIswithin the `auth:` namespace, and are enumerated in Section 3.2.

There are two special attributes which may be specified on `Mechanism` elements:

- `Next` is used to identify another `Mechanism` to use if this one fails **for any reason**. A common use case is where an *account email* mechanism is described for transparent authentication with a *user input* mechanism to prompt the user for this information. If the transparent mechanism fails either because the reading system does not have an account email stored or because the stored account email is incorrect, the reading system will then process to the user input mechanism referenced by this attribute.

- `Append` is also used to reference another `Mechanism` by URI. In this case, when one mechanism succeeds in obtaining the correct value, the referenced mechanism is also evaluated. If this next mechanism succeeds, its resultant value is *appended* to the current result. In this manner implementations can concatenate discrete values together to obtain the ultimate KEK value. For example a book can be chained to a single account/device pair by appending an *account email* mechanism to a *device ID* mechanism.

```
<element name="Mechanism" type="auth:MechanismType"/>
<complexType name="MechanismType">
  <sequence>
    <element ref="auth:AuthInfo"/>
    <element ref="auth:Prompt" minOccurs="0"/>
    <element ref="auth:Hint" minOccurs="0"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    <!-- (0,unbounded) elements from (1,1) external namespace -->
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="Type" type="anyURI" use="required"/>
  <attribute name="Next" type="anyURI" use="optional" />
  <attribute name="Append" type="anyURI" use="optional" />
</complexType>
```

### 3.1.3 The `AuthInfo` Element

The structure of an AuthInfo element includes the type of authentication parameter expected, along with optional information used to canonicalize and verify the obtained key. The element's `Type` attribute is used to specify an already-known value for transparent authentication; its presence is REQUIRED for all mechanisms aside from user-input, where it serves no purpose. Implementations MUST NOT accept an `AuthInfo` element with no `Type` within a transparent-authentication mechanism.

If the key needs to be transformed in any way prior to use as a KEK, a list of transforms conforming to Section 6.6 of [XML DSIG Core] may be used. However, given that the key is unlikely to be an XML node-set, we have defined some additional transformations as URIs within the `auth:` namespace which we enumerate in Section 3.4.

When dealing with hashed passwords, a salt value is often used. If required, the `InputVector` element allows for this to be provided to the reading system in base-64 format.

The URIs for the predefined and REQUIRED transparent authentication specifiers are enumerated in Section 3.3.

```
<element name="AuthInfo" type="auth:AuthInfoType"/>
<complexType name="AuthInfoType">
  <sequence>
    <element name="InputVector" type="ds:CryptoBinary" minOccurs="0"/>
```

```
      <element ref="ds:Transforms" minOccurs="0"/>
      <element ref="auth:ConfirmationValue" minOccurs="0"/>
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
      <!-- (0,unbounded) elements from (1,1) external namespace -->
    </sequence>
    <attribute name="Id" type="ID" use="optional"/>
    <attribute name="Type" type="anyURI" use="optional"/>
  </complexType>
```

### 3.1.4 The `ConfirmationValue` Element

A key obtained through authentication may be quickly verified by computing a digest and comparing it to an existing value. The value and the digest algorithm used to generate it are recorded within the `ConfirmationValue` element. We borrow elements from the [XML DSIG Core] specification here to define the algorithm and value using the `dsig:` namespace.

```
<element name="ConfirmationValue" type="auth:ConfirmationValue"/>
<complexType name="ConfirmationValue">
  <sequence>
    <element ref="ds:DigestMethod"/>
    <element ref="ds:DigestValue"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

### 3.1.5 The `UserDisplayValue` Type

The `UserDisplayValue` type is used to incorporate text which may be presented to the user. As such it is the type used for both the `Prompt` and `Hint` elements. Its contents are a plain string, and it may contain an identifier attribute.

```
<complexType name="UserDisplayValue">
  <simpleContent>
    <extension base="string">
      <attribute name="Id" type="ID" use="optional"/>
```

```
    </extension>
  </simpleContent>
</complexType>
```

### 3.1.6 The `Prompt` Element

This element contains text presented to the reader of a publication during the authentication phase. It SHOULD be used to describe the input required and for what it will be used. For instance: *"Please enter the email address associated with your Kobo account to open this book."*

```
<element name="Prompt" type="auth:UserDisplayValue"/>
```

### 3.1.7 The `Hint` Element

This element contains text which might be presented to the user to assist their recall. We anticipate that this will likely only be used when encrypting or re-encrypting content for personal distribution. Such use would likely have a password-based form of authentication similar to that used by Portable Document Format (PDF) documents; in that situation a *password hint* would be appropriate, so we include explicit support for it here.

```
<element name="Hint" type="auth:UserDisplayValue"/>
```

## 3.2 Mechanism Specifiers

A `Mechanism` specification takes the form of a URI within the `auth:` namespace. There are three mechanisms defined in this specification, and all three are REQUIRED.

### 3.2.1 Device-Keyed Authentication

The *device-keyed* mechanism performs authentication using a value unique to the device on which a reading system is running. Examples would include an Ethernet or WiFi MAC address or a hardware serial number. This mechanism is primarily of use to vendors who

wish to keep track of which devices users are using to read their books; tying fulfilment to each device gives a way to implement such a scheme.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#device-key`

### 3.2.2 Account-Keyed Authentication

The *account-keyed* mechanism authenticates an individual reader based on their account information. Common types would include email addresses, user IDs, or account passwords. When using the password approach, it is advised that the key be the result of digesting the real password. For this reason we have included some digest methods in our additional content transformations. This way the password can be stored by the server in hashed form and used as a KEK directly, and the local device can re-hash the password as entered by the user.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#account-key`

### 3.2.3 User-Input Authentication

The use of a *user-input* mechanism declares that authorization cannot take place without direct user input. Ideally a `Prompt` element is provided which contains some instructions for the user. Implementations MAY choose to implement their own messages for well-defined value types (such as those seen in Section 3.3), but SHOULD prefer any value within a `Prompt` element.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#user-input`

## 3.3 Transparent Key Specifiers

When implementing *transparent authentication*, a means is required to specify a value known to the reading system that it can use to determine a pre-disseminated key value. The URIs presented here serve as static identifiers for certain values. Implementation of the following identifiers is REQUIRED, although we anticipate that content distributors will add their own URIs to this list.

### 3.3.1 MAC Address

The most frequently used form of machine identification is the MAC address assigned to its primary Ethernet or WiFi port. Given the prevalence of WiFi interfaces these days, it's also frequently available on much smaller single-purpose devices such as dedicated eReaders. This makes it a good candidate for use in device-keyed authentication. Note that implementations MUST render the address into hexadecimal format; whether to include or omit colons between each character pair may be specified using the transforms described in Section 3.4.4.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#mac-address`

### 3.3.2 Device Serial Number

Many computing devices have access to a unique serial number at the hardware level, not only on a sticker applied to their packaging. These can also be used to perform device-keyed authentication.

**Identifier**:
`http://ww.idpf.org/epub/30/lcp-auth#serial-number`

### 3.3.3 Email Account

Readers with accounts for distributors such as Kobo or Barnes & Noble will usually have an email address tied to that account. This email address can be used as part of the KEK, under the assumption that the reader will be reticent to hand out their email address beyond a relatively small circle of friends. Additionally, it acts as an identifier for this particular user, meaning it might serve as a watermark value in its own right.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#account-email`

### 3.3.4 Account Password Hash

A user can be made to enter their password which will then be hashed using an optional input vector to generate an output KEK. The password is something the user would rarely consent to share with anyone, and then only with their closest friends and family.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#pass-hash`

### 3.3.5 Publication Identifiers

Encrypted key wrapping can be tied to a specific book by appending some book-specific information to the shared secret during authentication. This can be accomplished using one of the mechanisms defined below, but implementations MUST NOT generate or accept one of these as the key specifier if any primary authentication mechanism; it MUST only be specified through an `Append` attribute on another `Mechanism` element.

The *publication-id* type refers to the value of the element referenced by the `unique-identifier` attribute of an [OPF3] package document's `package` element. For example, the value returned from the listing below would be "`urn:uuid:2ec5fbf0-2554-012f-c04e-12313926e17c`".

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<package xmlns="http://www.idpf.org/2007/opf" version="3.0" unique-identifier="pub-id">
    <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:identifier id="pub-id">urn:uuid:2ec5fbf0-2554-012f-c04e-12313926e17c</dc:identifier>
      ...
    </metadata>
    ...
</package>
```

Specifying the *isbn* type also selects a publication ID, but it requires that it be a valid International Standard Book Number (ISBN). In many books this is stored within the `dc:identifier` metadata block of a package document (identified with an `opf:role` attribute of '`ISBN`'), but it may be provided through an additional custom metadata element or as an attribute. A frequent practice is to embed it within a Unified Resource Name (URN), as seen in the example below. In all these cases, the numerical text of the ISBN itself is the only value to be returned, stripped of any surrounding characters or other formatting.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<package xmlns="http://www.idpf.org/2007/opf" version="3.0" xml:lang="en" unique-identifier="pub-id">
    <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:identifier id="pub-id">urn:ISBN:9781118036914</dc:identifier>
```

```
        ...
    </metadata>
    ...
</package>
```

The *title* specifier refers to the contents of the `dc:title` element of the package document. The *title-and-authors* element requires some formatting, which implementations MUST follow:

- The publication title, as for the *title* specifier.
- The contents of each `dc:creator` element within the package document's metadata, according to the ordering declared by `display-seq` attribute values. If no `display-seq` values are available, authors are listed in order of occurrence.
- Elements will be concatenated, interspersed by a single comma and space.

For example, the value processed from the metadata below would be "The New Wealth Management, Harold Evensky, Stephen M. Horan, Thomas R. Robinson".

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<package xmlns="http://www.idpf.org/2007/opf" version="3.0" xml:lang="en↩
      " unique-identifier="pub-id">
    <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:identifier id="pub-id">urn:ISBN:9781118036914</dc:identifier↩
              >
        <meta refines="#pub-id" property="identifier-type" scheme="↩
              xsd:string">uuid</meta>
        <dc:title id="t1">The New Wealth Management</dc:title>
        <meta refines="#t1" property="title-type">main</meta>
        <meta refines="#t1" property="display-seq">1</meta>
        <dc:creator id="creator01">Harold Evensky</dc:creator>
        <meta refines="#creator01" property="role" scheme="marc:relators↩
              ">aut</meta>
        <meta refines="#creator01" property="file-as">Evensky, Harold</↩
              meta>
        <meta refines="#creator01" property="display-seq">1</meta>
        <dc:creator id="creator02">Stephen M. Horan</dc:creator>
        <meta refines="#creator02" property="role" scheme="marc:relators↩
              ">aut</meta>
```

```
        <meta refines="#creator02" property="file-as">Horan, Stephen M.<↩
              /meta>
        <meta refines="#creator02" property="display-seq">2</meta>
        <dc:creator id="creator03">Thomas R. Robinson</dc:creator>
        <meta refines="#creator03" property="role" scheme="marc:relators↩
              ">aut</meta>
        <meta refines="#creator03" property="file-as">Robinson, Thomas R↩
              .</meta>
        <meta refines="#creator03" property="display-seq">3</meta>
        ...
    </metadata>
    ...
</package>
```

**Identifier**:
http://www.idpf.org/epub/30/lcp-auth#publication-id
http://www.idpf.org/epub/30/lcp-auth#isbn
http://www.idpf.org/epub/30/lcp-auth#title
http://www.idpf.org/epub/30/lcp-auth#title-and-authors

## 3.4 Additional XML Transforms

With string-based inputs, the usefulness of the transforms provided by [XML DSIG Core] is somewhat limited— passwords and similar KEK values are unlikely to be XML after all. The base64 transform retains its usefulness in this new context, but no others can claim the same fate.

As a result, we present some new transforms here with associated URIs within our document's namespace, which will be useful for the canonicalization or transformation of plain-text values.

### 3.4.1 Case Folding

With string-based input, it is useful to be able to specify that text should be adjusted to a known case. In this instance, we consider the [UNICODE] implementation of case-folding algorithms to be definitive.

**Identifier**:
http://www.idpf.org/epub/30/lcp-auth#lowercase
http://www.idpf.org/epub/30/lcp-auth#uppercase

### 3.4.2 Japanese Ideograph Canonicalization

In Japanese, where syllabic letters (kana) exist alongside ideographic characters (kanji), we provide a transform that stipulates that text be rendered in only characters from one of the two sets of kana.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#hiragana`
`http://www.idpf.org/epub/30/lcp-auth#katakana`

### 3.4.3 Character Encoding

The issue of text-encoding is likely to raise its head. Though this document assumes the use of UTF-8 as a primary format, conversion to other [UNICODE] formats is useful, particularly when considering internationalization. To this end, the following specifiers allow provide the means to direct the use of a particular UTF character set. For each of the two wide character sizes we specify three variants: two for explicit byte orders and one to require the use of the appropriate Byte-Order Mark (BOM).

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#text-utf16-LE`
`http://www.idpf.org/epub/30/lcp-auth#text-utf16-BE`
`http://www.idpf.org/epub/30/lcp-auth#text-utf16-BOM`
`http://www.idpf.org/epub/30/lcp-auth#text-utf32-LE`
`http://www.idpf.org/epub/30/lcp-auth#text-utf32-BE`
`http://www.idpf.org/epub/30/lcp-auth#text-utf32-BOM`

### 3.4.4 MAC Formatting

MAC addresses are rendered as ten hexadecimal characters, often with a colon character separating each pair (each byte). The following transforms can be used to require that colons be inserted into or stripped from the resultant string.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-auth#with-separators`
`http://www.idpf.org/epub/30/lcp-auth#without-separators`

# 4 Rights Core Syntax

In this section we provide normative details of the rights-management schema and data. Here you will find discussions of each element defined by the schema along with instructions on their use. This is followed with details of the URI specifiers for the various right types defined by this specification.

## 4.1 Rights Management Core Syntax

This section provides detailed information on the syntax of the rights management data provided within `META-INF/authentication.xml`. All features described here are mandatory unless otherwise noted. The syntax is defined via [XMLSCHEMA-1] [XMLSCHEMA-2] with the following XML preamble, declaration, and internal entity:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema
  PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.w3.org/2001/↩
        XMLSchema.dtd"
  [
   <!ATTLIST schema
     xmlns:rights CDATA #FIXED "http://www.idpf.org/epub/30/lcp-rights#"↩
          >
   <!ENTITY rights 'http://www.idpf.org/epub/30/lcp-rights#'>
   <!ENTITY % p ''>
   <!ENTITY % s ''>
  ]>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:rights="http://↩
      www.idpf.org/epub/30/lcp-rights#" targetNamespace="http://www.idpf↩
      .org/epub/30/lcp-rights#" version="0.1" elementFormDefault="↩
      qualified">
```

### 4.1.1 The `Duration` Simple Type

The `Duration` type defines a `time` value of one second or longer.

```
  <simpleType name="Duration">
```

```
    <restriction base="time">
      <minInclusive value="00:00:01.0" /> <!-- one minute -->
    </restriction>
  </simpleType>
```

### 4.1.2 The `Counter` Simple Type

The `Counter` type defines an `integer` value in the range $0 \rightarrow \infty$.

```
<complexType name="LimitType">
  <simpleContent>
    <extension base="rights:Counter">
      <attribute name="Unit" type="rights:UnitType" use="required" />
    </extension>
  </simpleContent>
</complexType>
```

### 4.1.3 The `LimitType` Complex Type

`LimitType` defines an element containing a `Counter` value with a required `Unit` attribute.

```
<complexType name="LimitType">
  <simpleContent>
    <extension base="rights:Counter">
      <attribute name="Unit" type="rights:UnitType" use="required" />
    </extension>
  </simpleContent>
</complexType>
```

### 4.1.4 The `UnitType` Attribute Type

A `UnitType` attribute type consists of a `token` from the following set, and is used to describe the meaning of any numeric values associated with its parent element.

- "character"
  Numbers are to be interpreted as an integral character count.

- “sentence”
  Numbers are to be interpreted as an integral count of complete sentences.

- “paragraph”
  Numbers refer to the number of HTML … tags or similar.

- “page”
  Only used when defining limits on printing, this declares that the unit of measurement is the printed page.

- “percentage”
  Value is a floating-point type in the range $0 \rightarrow 100$ denoting a percentage of the total content of a publication, measured in visible characters.

- “time”
  The associated value is an amount of time, represented as a `Duration` type. Used to specify allowed lending durations.

```
<simpleType name="UnitType">
  <restriction base="token">
    <!-- valid based on visible characters, not bytes or runes of HTML↩
         /XML source -->
    <enumeration value="character" />
    <enumeration value="sentence" />

    <!-- valid based on HTML <p> tags or similar -->
    <enumeration value="paragraph" />

    <!-- only valid for printing limits -->
    <enumeration value="page" />

    <!-- percentage of all visible characters -->
    <enumeration value="percentage" />

    <!-- Used for duration-based limitations such as library lends -->
    <enumeration value="time" />
  </restriction>
</simpleType>
```

### 4.1.5 The `ConsumptionAmount` Element Type

The `ConsumptionAmount` element type specifies that an element contains a floating-point value $\geq 0.0$ with a required `Unit` attribute.

```
<simpleType name="ConsumptionAmountContent">
  <restriction base="float">
    <minInclusive value="0.0" />
  </restriction>
</simpleType>
<complexType name="ConsumptionAmount">
  <simpleContent>
    <extension base="rights:ConsumptionAmountContent">
      <attribute name="Unit" type="rights:UnitType" use="required" />
    </extension>
  </simpleContent>
</complexType>
```

### 4.1.6 The `Rights` Element

The `Rights` element forms the root of the `META-INF/rights.xml` file. It MUST contain only `Right` or `SharingInfo` elements. Each `Right` references a distinct right or limitation, but many `SharingInfo` elements may be provided to specify different permissions for different kinds of sharing.

```
<element name="Rights" type="rights:RightList" />
<complexType name="RightList">
  <sequence>
    <element ref="rights:Right" maxOccurs="unbounded" />
    <element ref="rights:SharingInfo" minOccurs="0" maxOccurs="↩
          unbounded" />
  </sequence>
</complexType>
```

### 4.1.7 The `Right` Element

The `Right` element is used to describe a single right or limitation regarding the use of a publication. It has two required attributes:

- The `Type` attribute denotes the actual capability to which this right applies. Examples include printing, copying and quoting (e.g. on social networks). Its value is a URI; we specify a group of right URIs within the `rights:` namespace in Section 4.2 which all implementations MUST support (subject to the limitations of the implementation's host device). Vendors may OPTIONALLY implement rights control for implementation-specific features using URIs within another domain.
- The `Unit` attribute is used to denote the unit of measurement for audited rights; its type is `UnitType`, which is described in Section 4.1.4.

The `Right` element may optionally contain a `Limit` attribute; this defines a per-action limitation on the capability described by the right. The `Limit` attribute should not be confused with the `LifetimeLimit` element defined in Section 4.1.9, which puts a maximum cap on utilization of the right.

A `Right` element defines restrictions (or the lack thereof) through its child elements `ExcludedContent`, `LifetimeLimit`, `Status`, and `EligibilityPeriod`. Where desirable, use of a right can be recorded within the `Consumption` element. In addition, implementations MAY add their own child elements as proves useful.

Lastly, it is possible to reference an `auth:Mechanism` within the `META-INF/authentication.xml` file in order to require manual authentication to perform the action managed by this right.

```
<element name="Right" type="rights:RightType" />
<complexType name="RightType">
  <sequence>
    <element ref="rights:Authorization" minOccurs="0" />
    <element ref="rights:ExcludedContent" minOccurs="0" />
    <element name="LifetimeLimit" type="rights:LimitType" minOccurs="0↩
        " />
    <element ref="rights:Status" />
    <element ref="rights:EligibilityPeriod" minOccurs="0" />
    <element ref="rights:Consumption" minOccurs="0" maxOccurs="↩
        unbounded" />
```

```
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
      <!-- (0,unbounded) elements from (1,1) external namespace -->
    </sequence>
    <attribute name="Id" type="ID" use="optional" />
    <attribute name="Unit" type="rights:UnitType" use="required" />
    <attribute name="Type" type="anyURI" use="required" />
    <attribute name="Limit" type="integer" use="optional" />
  </complexType>
```

### 4.1.8 The `Authorization` Element

This element is used to provide a URI identifying an `auth:Mechanism`. When the right's action takes place, implementations MUST perform the authentication steps described by that mechanism. It is RECOMMENDED that the mechanism require direct user input to authorize the right.

```
  <element name="Authorization" type="rights:AuthorizationType" />
  <complexType name="AuthorizationType">
    <attribute name="Id" type="ID" use="optional" />
    <attribute name="URI" type="anyURI" use="required" />
  </complexType>
```

### 4.1.9 The `LifetimeLimit` Element

Places an absolute limit on the uses of a right. This may be expressed in any of the units defined for the `UnitType` in Section 4.1.4.

```
      <element name="LifetimeLimit" type="rights:LimitType" minOccurs="0↩
            " />
```

### 4.1.10 The `ExcludedContent` Element

In some cases, a publisher may wish to explicitly deny the ability to perform certain activities on certain areas of the content. For example, a publisher of mystery novels may choose to prohibit any sharing or printing of content from the final chapter of the book,

or from any point after the mystery is solved. The `ExcludedContent` element provides two means to describe this: entire manifest items can be excluded from a right, such as whole chapters, or an exclusion may be specified at a more granular level using a ranged ePub Canonical Fragment Identifier (CFI) [EPUB-CFI].

Where a right's `Status` (defined in Section 4.1.17) is set to "`Permitted`" or "`Audited`", this element is used to imply a "`Denied`" status for any referenced items or ranges. Where a right is "`Denied`", this element applies a status of "`Permitted`" to its contents.

```
<element name="ExcludedContent" type="rights:ExcludedContentList" />
<complexType name="ExcludedContentList">
  <sequence>
    <element ref="rights:ManifestItem" minOccurs="0" maxOccurs="↩
          unbounded" />
    <element ref="rights:Fragment" minOccurs="0" maxOccurs="unbounded"↩
           />
  </sequence>
</complexType>
```

### 4.1.11 The `Manifest` Element

The `Manifest` element has no content and has a required `IdRef` attribute. This attribute's value MUST match the `id` attribute of a manifest item from an [OPF3] document.

This element denotes that the entire manifest item referenced is excluded from its parent right.

```
<element name="Manifest" type="rights:ManifestRef" />
<complexType name="ManifestRef">
  <attribute name="Id" type="ID" use="optional" />
  <attribute name="IdRef" type="token" use="required" />
</complexType>
```

### 4.1.12 The `Fragment` Element

The `Fragment` element also has no child elements or content. Its required attribute is `CFI`, which SHOULD be a valid ranged [EPUB-CFI]; implementations MUST be prepared to handle erroneous values.

The range specified by the `CFI` attribute is excluded from the effects of its parent right.

```
<element name="Fragment" type="rights:FragmentRef" />
<complexType name="FragmentRef">
  <attribute name="Id" type="ID" use="optional" />
  <attribute name="CFI" type="string" use="required" />
</complexType>
```

### 4.1.13 The `Consumption` Element

The `Consumption` element holds a list of `UseInfo` elements denoting the times and details of a right's use for auditing purposes.

```
<element name="Consumption" type="rights:UseList" />
<complexType name="UseList">
  <sequence>
    <element ref="rights:UseInfo" minOccurs="0" maxOccurs="unbounded" ↩
          />
  </sequence>
</complexType>
```

### 4.1.14 The `UseInfo` Element

Each `UseInfo` element details the time at which a user exercised the capability described by its ancestor `Right` element along with information on the amounts used, where applicable. For instance, if a user prints two pages, that value will be recorded here. If printing is limited by a value of a different unit (such as printing at most 10% of a publication) then implementations MUST also record the same value in that unit as well.

An example of this can be seen in Section 2.5.

```
<element name="UseInfo" type="rights:UseInfoType" />
<complexType name="UseInfoType">
  <sequence>
    <element ref="rights:Timestamp" />
    <element ref="rights:Amount" maxOccurs="unbounded" />
```

```
    </sequence>
    <attribute name="Id" type="ID" use="optional" />
  </complexType>
```

### 4.1.15 The `Timestamp` Element

A `Timestamp` element MUST contain a valid XML `dateTime` value. It is used to record the time at which an audited right was exercised.

```
<element name="Timestamp" type="dateTime" />
```

### 4.1.16 The `Amount` Element

The `Amount` element is a `ConsumptionAmount`, specifying an amount used of an audited right. It requires a `Unit` attribute to specify the `UnitType` in which the amount is recorded.

```
<element name="Amount" type="rights:ConsumptionAmount" />
```

### 4.1.17 The `Status` Element

A right's *status*, as denoted by this REQUIRED child element, specifies one of three mutually-exclusive states for a right.

- A value of `Permitted` identifies a right as being allowed *without restriction*. Marking a right with this status has the same effect as not listing the right at all (that is, implementations MUST assume that any recognized right not specified is permitted without restriction). Including it and marking it as permitted makes the decision absolute; it is RECOMMENDED that conforming implementations generate entries for all rights they understand and mark unrestricted ones with this status.
- Marking a right `Audited` signals to the reading system that there are prescribed limits on the use of this right. There need not be a lifetime limit applied, but any limits provided MUST be enforced. Recording an audit trail in the right's `Consumption` element is REQUIRED for rights with a `LifetimeLimit`, but is OPTIONAL for those without.

- A value of `Denied` indicates that a right is to be unequivocally denied. Implementations SHOULD NOT edit a right that has met its lifetime limit to this state, but that will not make the document invalid. It is RECOMMENDED that implementations only use this status to indicate that a right has been absolutely denied from the outset, not to denote that a right's usage quota has been exhausted.

```
<element name="Status" type="rights:StatusType" />
<simpleType name="StatusType">
  <restriction base="token">
    <enumeration value="Permitted" /> <!-- Permitted without limit -->
    <enumeration value="Audited" />   <!-- Permitted only within ↩
          audited limits -->
    <enumeration value="Denied" />    <!-- Never permitted -->
  </restriction>
</simpleType>
```

### 4.1.18 The `EligibilityPeriod` Element

A right may have an eligibility period attached to it, such that it may only be exercised during the period indicated.

An eligibility period may contain either a `Start` element, an `End` element, or both. When both are provided, the eligibility period encompasses the provided dates, inclusive of the start time and exclusive of the end time. When the start time is not specified, the eligibility period includes every moment preceding the end time; when an end time is absent, the eligibility period is any moment from that time onwards.

The most common use for an `EligibilityPeriod` is a library loan— the `Start` value would be the date and time at which the item was checked out, and the `End` value indicates the time it would be automatically 'returned' to the library.

Another potential use might be the enforcement of embargoes, for example on review copies of highly-anticipated books. All rights to copy, lend, quote, print, and similar would unlock on the book's 'shelf date' only. In this example, only a `Start` value need be provided.

```
<element name="EligibilityPeriod" type="rights:EligibilityPeriodType" ↩
      />
```

```
<complexType name="EligibilityPeriodType">
  <sequence>
    <element name="Start" type="rights:EligibilityDelimiter" minOccurs=↩
          "0" />
    <element name="End" type="rights:EligibilityDelimiter" minOccurs="↩
          0" />
  </sequence>
  <attribute name="Id" type="ID" use="optional" />
</complexType>
```

### 4.1.19 The `EligibilityDelimiter` Type

The `EligibilityDelimiter` type is used to specify start and end dates for an `EligibilityPeriod`. It contains an XML `dateTime` value, and may have an XML identifier attribute.

```
<complexType name="EligibilityDelimiter">
<simpleContent>
  <extension base="dateTime">
    <attribute name="Id" type="ID" use="optional" />
  </extension>
</simpleContent>
</complexType>
```

### 4.1.20 The `SharingInfo` Element

The `SharingInfo` element is used to specify the means and limitations on lending a publication to other readers. Note that this does *not* attempt to prevent Alice from simply sending the file to Bob along with her account details, password, and any other information necessary to open the book as-is. Rather it attempts to provide guidance to software-assisted lending, where a book's access keys are re-encrypted using a new KEK, the use of which would allow the other user to access the book.

The means of providing the copy to another user is not specified as part of this proposal. Potential systems might be as simple as email or as elaborate as having a cloud-based repository generate a book for Alice, re-encode the KEK for Bob, update the `Consumption` details in the book's `rights.xml` with the details of the loan, and adding it directly to Bob's library.

Either way, the lent copy MUST be given a 'Read' right with an `EligibilityPeriod`, and the receiving reading system MUST honour that data.

Sharing can be made to require explicit authorization by referencing an [authorization!schema]Mechanism element using the `Authorization` element discussed in Section 4.1.8.

The child elements `MinDuration` and `MaxDuration` enable a publisher or distributor to place limits on the amount of time for which a book may be loaned to another reader, and a `LifetimeLimit` child element can place a hard limit on the number of times a book may be lent out in total.

The `Limit` attribute provides a means to allow lending more than one copy simultaneously; if not specified, implementations MUST assume a value of 1.

The `Exclusive` attribute, if set to `true`, will prevent the owner of the book from accessing their own copy while it is lent to another reader.

The `Domain` attribute can be used to limit the group of people to whom a book may be lent. Examples would include 'anyone' or 'other Kobo users' and so on.

```
<element name="SharingInfo" type="rights:SharingInfoType" />
<complexType name="SharingInfoType">
  <sequence>
    <element ref="rights:Authorization" minOccurs="0" />
    <element name="MaxDuration" type="rights:Duration" />
    <element name="MinDuration" type="rights:Duration" minOccurs="0" />
          >
    <element name="LifetimeLimit" type="rights:LimitType" minOccurs="0
          " />
    <element ref="rights:Consumption" minOccurs="0" maxOccurs="
          unbounded" />
  </sequence>
  <attribute name="Exclusive" type="boolean" default="false" />
  <attribute name="Id" type="ID" use="optional" />
  <attribute name="Limit" type="integer" use="optional" default="1" />
  <attribute name="Domain" type="anyURI" default="http://www.idpf.org/
        epub/30/lcp-rights#open" />
</complexType>
```

## 4.2 Right Specifiers

A right specification takes the form of a URI. We define a number of these within the `rights:` namespace. Implementation of all rights defined in this specification is REQUIRED.

### 4.2.1 Printing

Printing can be limited based on the number of pages printed at any one time. A `LifetimeLimit` may be based on either a printed-page count or a percentage of the book.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-rights#print`

### 4.2.2 Copying

Sets limits on the ability to copy selected content to the clipboard.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-rights#copy`

### 4.2.3 Social Sharing

This specifier identifies the right to share selected text directly as quotes through social networks or emails.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-rights#social-share`

### 4.2.4 Reading

Library loans can use this right to apply a time window to the accessibility of the book's content.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-rights#read`

### 4.2.5 Editing

The right to edit a document is added. When this right's state is "audited", we expect less that there be a useful limit, more that an audit trail be kept. This may obviously be allowed or denied outright as well, though in a collaboratively-edited document we could see the `ExcludedContent` element being used to 'lock' particular sections from further changes once consensus has been reached.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-rights#edit`

## 4.3 Lending Domains

In this specification we define only one domain, which places no restrictions on users to which a publication may be shared. Other potential domains might include anyone with an account on the same vendor, such as on the Kobo store or Amazon's Kindle store. Another example might be anyone employed by a particular publishing house, allowing free transfer between employees; access could be based on the domain of the sender's and recipient's email addresses or upon a directory service, e.g. LDAP.

### 4.3.1 Open Domain

The open domain is used to indicate that a book may be lent without restriction on the identity of the receiver.

**Identifier**:
`http://www.idpf.org/epub/30/lcp-rights#open`

“

# A Index of Requirements

All components and facilities are laid out here, organized into Required, Recommended, and Optional sets.

## A.1 Authentication

### Required

- The `META-INF/authentication.xml` file must be included and must specify an authentication mechanism according to the schema in Section 3.1.
- The `META-INF/authentication.xml` file must be protected from modification through use of an [XML DSIG Core] Signature. Placing a signature within the file itself is Optional but enables the modification of the authentication vector when lending; implementors may simply rely on the signed digest block within `META-INF/signature.xml`, but then that file would need to be modified any time the book is lent.
- Implementors Must Not evaluate any mechanism without being directed to it from another location, such as from a right, an encryption key `ds:RetrievalMethod`, or another mechanism.
- Implementations must support all authentication mechanisms defined in this proposal.
- Implementations must adhere to the requirements and recommendations or [XML DSIG Core] with regards to supported algorithms and their URIs.
- Implementations Must Not use a publication ID key specifier in the primary authentication mechanism, and Must reject any publication whose authentication does so.

### Recommended

- We recommend the use of the AES Keywrapping algorithm defined in [RFC3394] with a KEK size of at least 128 bits, although any algorithm described by [XML DSIG Core] may be used.

- We recommend the authorization key be constructed from at least two discrete pieces of data. For example the shared secret obtained through the primary authentication mechanism could be concatenated with the publication's unique identifier through an appended mechanism to generate the required signing/encryption key.

## A.2 Content Signing

### Required

- Implementations are Required to conform to the requirements and recommendations of the [XML DSIG Core] specification.

### Recommended

- The preferred signing algorithm is RSA-SHA256 as defined in RFC 3447 [PKCS1], using a public key size of at least 2048 bits. Public keys of 1024 bits or less Should Not be used, and implementors Should use at least 3072-bit signatures for content expected to be verified beyond 2030.
- To properly verify publisher/author signatures, implementations Should handle [XPath] and [XPath-Filter] transformations. Examples can be found in Section 2.3.1.

### Optional

- An distributor May use an enveloped signature transform when signing content which already contains a publisher transform. Any implementation of such Must conform to the specification in Section 6.6.4 of [XML DSIG Core].

## A.3 Encryption

### Required

- Implementations Must conform to the requirements and recommendations of the [XML ENC Core] specification.

- The use of an encrypted key is Required, as is an `xenc:RetrievalMethod` for the KEK referencing a `Mechanism` element inside `META-INF/authentication.xml`.
- Pursuant to Section 2.5.2 of the [OCF3] specification, all encrypted files Must be compressed using the [Deflate] algorithm before encryption and placed into the ZIP container in uncompressed format.

#### Recommended

- Wf recommend the use of [X.509] key-pairs to implement code-signing and encryption. The certificate chain and CRL Should be included within `META-INF/encryption.xml`.

#### Optional

- Implementations May choose to include the ancestor nodes of an [X.509] certificate chain in unencrypted format within `META-INF/encryption.xml`. The leaf certificate used to verify/decrypt content Must remain encrypted pursuant to the authentication system outlined in Section 2.1.

### A.4 Rights Management

#### Required

- Implementations Must recognize all rights specified in this document, although their implementation of the individual features being restricted are naturally bound by the capabilities of the reading system itself.
- Implementations Must treat unspecified rights as implicitly `Permitted`.
- If no `SharingInfo` elements are specified, personal lending Must be implicitly `Denied`.

#### Recommended

- Implementations Should include all known rights when generating the contents of `META-INF/rights.xml`, marking unrestricted rights as explicitly `Permitted`.
- The `META-INF/rights.xml` file Should be signed using an *enveloped signature* to provide independent verification of its contents.

### Optional

- Implementations May implement personal lending as a software-assisted feature, re-creating the `META-INF/signature.xml`, `META-INF/encryption.xml`, `META-INF/rights.xml` and `META-INF/authentication.xml` files as necessary to facilitate the transfer of content. This is not required, but the rights management system has been designed with this in mind.
- Implementations are free to declare certain rights as always prohibited, ignoring any permissions specified for those rights in a publication, but they Must communicate this fact to the user.

# B Processing Instructions

The algorithms defined here represent the envisioned process for each step: opening a publication and evaluating a right.

## B.1 Authorization Core Algorithm

**procedure** ProcessEPubAuth($mechanism$)
    $keyString \leftarrow$ “”
    **while** $mechanism \neq nil$ **do**
        $thisKey \leftarrow nil$
        $mechanismType \leftarrow$ GetAttribute($mechanism$, “Type”)
        $authInfo \leftarrow$ NodeForXPath($mechanism$, “./AuthInfo”)
        $authType \leftarrow$ GetAttribute($authInfo$, “Type”)

        **if** $mechanismType =$ ‘$device - keyed$’ **then**
            $thisKey \leftarrow$ GetDeviceKey($authType$)
        **else if** $mechanismType =$ ‘$account - keyed$’ **then**
            $thisKey \leftarrow$ GetAccountKey($authType$)
        **else if** $mechanismType =$ ‘$user - input$’ **then**
            $prompt \leftarrow$ NodeForXPath($authInfo$, “./Prompt/text()”)
            $hint \leftarrow$ NodeForXPath($authInfo$, “./Hint/text()”)
            $thisKey \leftarrow$ PromptUser($prompt$, $hint$)
        **end if**

        **if** $thisKey \neq nil$ **then** ▷ Confirm the key if possible
            $confirmationValue \leftarrow$ NodeForXPath($mechanism$, “./Confirmation-Value”)
            **if** $confirmationValue \neq nil$ **then**
                $digestAlgorithm \leftarrow$ NodeForXPath($confirmationValue$, “./Digest-Method/@Algorithm”)
                $digestValue \leftarrow$ NodeForXPath($confirmationValue$, “./DigestVal-ue/text()”)
                **if** $digestValue \neq$ ComputeDigest($thisKey$, $digestAlgorithm$) **then**
                    $thisKey \leftarrow nil$
                **end if**
            **end if**
        **end if**

```
        if thisKey ≠ nil then
            keyString ← keyString + thisKey
            appendURI ← GetAttribute(mechanism,"Append")
            if appendURI ≠ nil then
                mechanism ← ParseXMLAtURI(appendURI)
            else
                mechanism ← nil
            end if
        else
            nextURI ← GetAttribute(mechanism,"Next")
            if nextURI ≠ nil then
                mechanism ← ParseXMLAtURI(nextURI)
            else
                mechanism ← nil
            end if
        end if
    end while
end procedure
```

## B.2 Signature Verification

```
gAuthNamespace ← "http://www.idpf.org/epub/30/lcp-auth#

procedure AuthorizePublication(pub)
    if IsSigned(pub) then
        if VerifySignature(pub) = false then
            return NotAuthorized
        end if
    end if
    return AuthorizationOK
end procedure

procedure IsSigned(pub)
    if HasFile(pub, "META-INF/signature.xml") then
        return true
    else
        return false
```

```
    end if
end procedure

procedure VerifySignature(pub)
    xml ← ParseXML(pub, "META-INF/signature.xml")
    if xml = nil then
        return false
    end if

    key ← GetSigningKey(xml)
    if key = nil then
        return false
    end if

    signedInfo ← NodeForXPath(xml, "/Signature/SignedInfo")
    references ← AllNodesForXPath(signedInfo, "Reference")
    for all ref ∈ references do                          ▷ Verify all references
        valid ← VerifyReference(ref)      ▷ Implemented as per [XML DSIG Core]
        if valid = false then
            return false
        end if
    end for

    signature ← NodesForXPath(xml. "/Signature/SignatureValue/text()")
    return ValidateSignedInfo(signedInfo, key, signature)
end procedure

procedure GetSigningKey(xml)
    keyInfo ← NodeForXPath(xml, "./Signature/KeyInfo")
    keyValue ← NodeForXPath(keyInfo, "./KeyValue/text()")
    if keyValue ≠ nil then
        return UnpackKey(keyValue)
    end if

    key ← nil
    retrievalMethod ← NodeForXPath(keyInfo, "./RetrievalMethod")
    if retrievalMethod ≠ nil then
        uri ← GetAttribute(retrievalMethod, "URI")
```

```
        target ← ParseXMLAtURI(uri)
        if GetNamespace(target) ≠ gAuthNamespace then
            transforms ← NodesForXPath(retrievalMethod, "./Transforms/Transform")
            return TransformXML(target, transforms)
        else
            return ProcessEPubAuth(target)
        end if
    end if
    return nil
end procedure
```

## B.3 Rights Evaluation

```
Denied ← −1
Permitted ← 0

procedure EvaluateRight(name, manifestItemID, locationCFI)
    uri ← RightURIForName(name)
    if uri = nil then
        return Permitted                    ▷ Unknown rights are implicitly permitted
    end if

    xml ← ParseXML("META-INF/rights.xml")
    if xml = nil then
        return Permitted                    ▷ No rights.xml means all are permitted
    end if

    xpath ← FormatString("/Rights/[@Type=%s]", uri)
    rightInfo ← NodeForXPath(xml, xpath)
    if rightInfo = nil then
        return Permitted                    ▷ No right in file means implicitly permitted
    end if

    status ← NodeForXPath(rightInfo, "./Status/text()")
    if status = "Permitted" then
        if IsExcluded(manifestItemID, locationCFI, rightInfo) then
            return Denied
```

```
        else
            return Permitted
        end if
    else if status = "Denied" then
        if IsExcluded(manifestItemID, locationCFI, rightInfo) then
            return Permitted
        else
            return Denied
        end if
    else if IsExcluded(manifestItemID, locationCFI, rightInfo) then
        return Denied
    else            ▷ Note that we assume file is valid, so status must be "Audited" here
        return EvaluateAuditedRight(rightInfo)
    end if
end procedure

procedure IsExcluded(manifestItemID, locationCFI, rightInfo)
    exclusions ← NodeForXPath(rightInfo, "./ExcludedContent[1]")
    for all exclusion ∈ ChildNodes(exclusions) do
        name ← ElementName(exclusion)
        value ← ElementValue(exclusion)
        if name = "Manifest" and value = manifestItemID then
            return true
        else if name = "Fragment" and LocationInRange(locationCFI, value) then
            return true
        end if
    end for
    return false
end procedure

procedure EvaluateAuditedRight(rightInfo)
    limit ← GetAttribute(rightInfo, "Limit")
    LifetimeLimit ← NodeForXPath(rightInfo, "./LifetimeLimit/text()")

    authorizationURI ← NodeForXPath(rightInfo, "./Authorization/@URI")
    if authorizationURI ≠ nil then
        mechanism ← ParseXMLAtURI(authorizationURI)
        if mechanism = nil or ProcessEPubAuth(mechanism) = nil then
```

```
            return Denied
        end if
    end if

    eligibilityPeriod ← NodeForXPath(rightInfo, "./EligibilityPeriod'[1]')
    if eligibilityPeriod ≠ nil and WithinEligibilityPeriod(eligibilityPeriod) =
false then
        return Denied
    end if

    if lifetimeLimit = 0 then
        return Limit                ▷ No lifetime limit, so per-action limit returned as-is
    end if

    lifetimeUnit ← NodeForXPath(rightInfo, "./LifetimeLimit/@Unit/string()")
    lifetimeRemaining ← lifetimeLimit
    uses ← AllNodesForXPath(rightInfo, "./Consumption/UseInfo")
    for all useInfo ∈ uses do                                   ▷ Accumulate totals
        lifetimeRemaining ← lifetimeRemaining− TotalInUnit(use, lifetimeUnit)
    end for

    if lifetimeRemaining > 0 then
        if lifetimeRemaining ≥ limit then
            return limit                    ▷ Use the per-action limit unchanged
        else
            return lifetimeRemaining        ▷ Less than per-action limit remaining
        end if
    else
        return Denied                       ▷ Lifetime limit has been exhausted
    end if
end procedure

procedure TotalInUnit(useInfo, unit)
    xpath ← FormatString("./Amount[@Unit=%s]/text()", unit)
    total ← 0
    for all amount ∈ AllNodesForXPath(useInfo, xpath) do
        total ← total + amount
    end for
```

```
    return total
end procedure
```

## B.4 Sharing Evaluation

```
procedure EvaluateShare(receiver, requestedDuration)
    xml ← ParseXML("META-INF/rights.xml")
    for all sharingInfo ∈ AllNodesForXPath(xml, "/Rights/SharingInfo") do
        domain ← GetAttribute(sharingInfo, "Domain")
        if domain ≠ nil and UserInDomain(receiver, domain) = false then
            next                    ▷ Receiver not in domain— go to next SharingInfo
        end if

        limit ← GetAttribute(sharingInfo, "Limit")
        if limit = nil then
            limit ← 1                              ▷ Default limit is 1 share at a time
        end if
        lifetimeLimit ← NodeForXPath(sharingInfo, "./LifetimeLimit[1]/text()")
        available ← lifetimeLimit
        if lifetimeLimit = nil then
            available ← limit
        end if

        if lifetimeLimit ≠ nil then
            lifetimeUnit ← NodeForXPath(sharingInfo, "./LifetimeLimit[1]/@Unit/string()")
            uses ← AllNodesForXPath(sharingInfo, "./Consumption/UseInfo")
            current ← 0
            for all useInfo ∈ uses do                          ▷ Accumulate totals
                available ← available− TotalInUnit(useInfo, lifetimeUnit)
                if unit ="time" then                           ▷ Check for current shares
                    if UseInProgress(useInfo) then
                        next                ▷ In progress— look to the next SharingInfo
                    end if
                end if
            end for
            if lifetimeRemaining ≤ 0 then
                next                        ▷ Exhausted— go to the next SharingInfo
            end if
```

```
            end if

            if available ≤ 0 then
                next                    ▷ No available shares here, go to the next SharingInfo
            else
                minDuration ← NodeForXPath(sharingInfo, "./MinDuration/text()")
                maxDuration ← NodeForXPath(sharingInfo, "./MaxDuration/text()")
                if minDuration ≠ nil and requestedDuration < minDuration then
                    requestedDuration ← minDuration
                end if
                if requestedDuration > maxDuration then
                    requestedDuration ← maxDuration
                end if
                return requestedDuration                ▷ Return the allowed duration
            end if
        end for
        return nil                                      ▷ If we get here, sharing is denied
    end procedure

    procedure UseInProgress(useInfo)
        xpath ← gDateRangeXPathExpression               ▷ XPath defined in Section B.4.1
        result ← NodeForXPath(useInfo, xpath)
        return BooleanValue(node)
    end procedure
```

### B.4.1 Date Range Evaluation XPath Expression

Listing 1: WithinDateRange() XQuery Implementation

```
declare function rights:within-date-range ( $arg as rights:UseInfoType )↩
       as xs:boolean
{
    let $start := xs:dateTime($arg/Timestamp/text())
    let $end := op:add-dayTimeDuration-to-dateTime($start, xs:↩
          dayTimeDuration($arg/Amount/text()))
    op:dateTime-greater-than($start)
        and
    op:dateTime-less-than($end)
```

```
}
```

Listing 2: WithinDateRange() XSLT Implementation

```
<xsl:function name="rights:within-date-range" as="xs:boolean">
    <xsl:param name="arg" as="rights:UseInfoType"/>
    <xsl:variable name="start" as="xs:dateTime" select="$arg/Timestamp/↩
        text()"/>
    <xsl:variable name="end" select="op:add-dayTimeDuration-to-dateTime↩
        ($start, xs:dayTimeDuration($arg/Amount/text()))"/>
    <xsl:sequence select="op:dateTime-greater-than($start) and ↩
        op:dateTime-less-than($end)"/>
</xsl:function>
```

# C Proposed XML Schemas

In this section you can find the complete text of the XML schemas defined by this document. For details on implementation, refer to Sections 3 and 4.

## C.1 The LCP-AUTH Schema

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.↩
      w3.org/2001/XMLSchema.dtd"
[
 <!ATTLIST schema xmlns:auth CDATA #FIXED "http://www.idpf.org/epub/30/↩
       lcp-auth#">
 <!ATTLIST schema xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/↩
       xmldsig#">
 <!ENTITY auth "http://www.idpf.org/epub/30/lcp-auth#">
 <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
 <!ENTITY % p "">
 <!ENTITY % s "">
]>
<schema xmlns:auth="http://www.idpf.org/epub/30/lcp-auth#" xmlns:ds="↩
      http://www.w3.org/2000/09/xmldsig#" targetNamespace="http://www.↩
      idpf.org/epub/30/lcp-auth#" version="0.1" elementFormDefault="↩
      qualified">
    <!-- Start Authentication -->
    <element name="Authentication" type="auth:AuthenticationType"/>
    <complexType name="AuthenticationType">
        <sequence>
            <element ref="ds:Signature"/>
            <element ref="auth:Mechanism" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="Provider" type="anyURI" use="optional"/>
    </complexType>
    <!-- Start Mechanism -->
    <element name="Mechanism" type="auth:MechanismType"/>
    <complexType name="MechanismType">
        <sequence>
            <element ref="auth:AuthInfo"/>
```

```
            <element ref="auth:Prompt" minOccurs="0"/>
            <element ref="auth:Hint" minOccurs="0"/>
            <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
            <!-- (0,unbounded) elements from (1,1) external namespace -->
        </sequence>
        <attribute name="Id" type="ID" use="optional"/>
        <attribute name="Type" type="anyURI" use="required"/>
        <attribute name="Next" type="anyURI" use="optional" />
        <attribute name="Append" type="anyURI" use="optional" />
    </complexType>
    <!-- Start AuthInfo -->
    <element name="AuthInfo" type="auth:AuthInfoType"/>
    <complexType name="AuthInfoType">
        <sequence>
            <element name="InputVector" type="ds:CryptoBinary" minOccurs="0"/>
            <element ref="ds:Transforms" minOccurs="0"/>
            <element ref="auth:ConfirmationValue" minOccurs="0"/>
            <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
            <!-- (0,unbounded) elements from (1,1) external namespace -->
        </sequence>
        <attribute name="Id" type="ID" use="optional"/>
        <attribute name="Type" type="anyURI" use="required"/>
    </complexType>
    <!-- Start ConfirmationValue -->
    <element name="ConfirmationValue" type="auth:ConfirmationValue"/>
    <complexType name="ConfirmationValue">
        <sequence>
            <element ref="ds:DigestMethod"/>
            <element ref="ds:DigestValue"/>
        </sequence>
        <attribute name="Id" type="ID" use="optional"/>
    </complexType>
    <!-- Simple Elements -->
    <element name="Prompt" type="auth:UserDisplayValue"/>
    <element name="Hint" type="auth:UserDisplayValue"/>
    <complexType name="UserDisplayValue">
```

```
        <simpleContent>
            <extension base="string">
                <attribute name="Id" type="ID" use="optional"/>
            </extension>
        </simpleContent>
    </complexType>
</schema>
```

## C.2 The LCP-RIGHTS Schema

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema
  PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd"
  [
   <!ATTLIST schema
     xmlns:rights CDATA #FIXED "http://www.idpf.org/epub/30/lcp-rights#">
   <!ENTITY rights 'http://www.idpf.org/epub/30/lcp-rights#'>
   <!ENTITY % p ''>
   <!ENTITY % s ''>
  ]>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:rights="http://www.idpf.org/epub/30/lcp-rights#" targetNamespace="http://www.idpf.org/epub/30/lcp-rights#" version="0.1" elementFormDefault="qualified">
  <!-- Duration type -->
  <simpleType name="Duration">
    <restriction base="time">
      <minInclusive value="00:00:01.0" /> <!-- one minute -->
    </restriction>
  </simpleType>
  <!-- Counter Type -->
  <simpleType name="Counter">
    <restriction base="integer">
      <minInclusive value="0" />
    </restriction>
  </simpleType>
  <!-- Limit Type -->
```

```
  <complexType name="LimitType">
26  <simpleContent>
      <extension base="rights:Counter">
28      <attribute name="Unit" type="rights:UnitType" use="required" />
      </extension>
30  </simpleContent>
  </complexType>
32 <!-- Unit Type -->
  <simpleType name="UnitType">
34  <restriction base="token">
      <!-- valid based on visible characters, not bytes or runes of HTML↩
           /XML source -->
36    <enumeration value="character" />
      <enumeration value="sentence" />
38
      <!-- valid based on HTML <p> tags or similar -->
40    <enumeration value="paragraph" />

42    <!-- only valid for printing limits -->
      <enumeration value="page" />
44
      <!-- percentage of all visible characters -->
46    <enumeration value="percentage" />

48    <!-- Used for duration-based limitations such as library lends -->
      <enumeration value="time" />
50  </restriction>
  </simpleType>
52 <!-- Start Rights -->
  <element name="Rights" type="rights:RightList" />
54 <complexType name="RightList">
    <sequence>
56    <element ref="rights:Right" maxOccurs="unbounded" />
      <element ref="rights:SharingInfo" minOccurs="0" maxOccurs="↩
           unbounded" />
58  </sequence>
  </complexType>
60 <!-- Start Right -->
  <element name="Right" type="rights:RightType" />
62 <complexType name="RightType">
    <sequence>
```

```
64        <element ref="rights:Authorization" minOccurs="0" />
          <element ref="rights:ExcludedContent" minOccurs="0" />
66        <element name="LifetimeLimit" type="rights:LimitType" minOccurs="0↩
              " />
          <element ref="rights:Status" />
68        <element ref="rights:EligibilityPeriod" minOccurs="0" />
          <element ref="rights:Consumption" minOccurs="0" maxOccurs="↩
              unbounded" />
70        <any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
          <!-- (0,unbounded) elements from (1,1) external namespace -->
72      </sequence>
        <attribute name="Id" type="ID" use="optional" />
74      <attribute name="Unit" type="rights:UnitType" use="required" />
        <attribute name="Type" type="anyURI" use="required" />
76      <attribute name="Limit" type="integer" use="optional" />
    </complexType>
78  <!-- Start Authorization -->
    <element name="Authorization" type="rights:AuthorizationType" />
80  <complexType name="AuthorizationType">
        <attribute name="Id" type="ID" use="optional" />
82      <attribute name="URI" type="anyURI" use="required" />
    </complexType>
84  <!-- Start ExcludedContent -->
    <element name="ExcludedContent" type="rights:ExcludedContentList" />
86  <complexType name="ExcludedContentList">
      <sequence>
88      <element ref="rights:Manifest" minOccurs="0" maxOccurs="unbounded"↩
            />
        <element ref="rights:Fragment" minOccurs="0" maxOccurs="unbounded"↩
            />
90    </sequence>
    </complexType>
92  <!-- Start Manifest -->
    <element name="Manifest" type="rights:ManifestRef" />
94  <complexType name="ManifestRef">
      <attribute name="Id" type="ID" use="optional" />
96    <attribute name="IdRef" type="token" use="required" />
    </complexType>
98  <!-- Start Fragment -->
    <element name="Fragment" type="rights:FragmentRef" />
100 <complexType name="FragmentRef">
```

```
    <attribute name="Id" type="ID" use="optional" />
    <attribute name="CFI" type="string" use="required" />
  </complexType>
  <!-- Start Consumption -->
  <element name="Consumption" type="rights:UseList" />
  <complexType name="UseList">
    <sequence>
      <element ref="rights:UseInfo" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
  <!-- Start UseInfo -->
  <element name="UseInfo" type="rights:UseInfoType" />
  <complexType name="UseInfoType">
    <sequence>
      <element ref="rights:Timestamp" />
      <element ref="rights:Amount" maxOccurs="unbounded" />
    </sequence>
    <attribute name="Id" type="ID" use="optional" />
  </complexType>
  <!-- Start Timestamp -->
  <element name="Timestamp" type="dateTime" />
  <!-- Start Amount -->
  <element name="Amount" type="rights:ConsumptionAmount" />
  <!-- Start ConsumptionAmount -->
  <simpleType name="ConsumptionAmountContent">
    <restriction base="float">
      <minInclusive value="0.0" />
    </restriction>
  </simpleType>
  <complexType name="ConsumptionAmount">
    <simpleContent>
      <extension base="rights:ConsumptionAmountContent">
        <attribute name="Unit" type="rights:UnitType" use="required" />
      </extension>
    </simpleContent>
  </complexType>
  <!-- Start Status -->
  <element name="Status" type="rights:StatusType" />
  <simpleType name="StatusType">
    <restriction base="token">
```

```
      <enumeration value="Permitted" /> <!-- Permitted without limit -->
      <enumeration value="Audited" />       <!-- Permitted only within audited limits -->
      <enumeration value="Denied" />        <!-- Never permitted -->
    </restriction>
  </simpleType>
  <!-- Start EligibilityPeriod -->
  <element name="EligibilityPeriod" type="rights:EligibilityPeriodType" />
  <complexType name="EligibilityPeriodType">
    <sequence>
      <element name="Start" type="rights:EligibilityDelimiter" minOccurs="0" />
      <element name="End" type="rights:EligibilityDelimiter" minOccurs="0" />
    </sequence>
    <attribute name="Id" type="ID" use="optional" />
  </complexType>
  <complexType name="EligibilityDelimiter">
  <simpleContent>
    <extension base="dateTime">
      <attribute name="Id" type="ID" use="optional" />
    </extension>
  </simpleContent>
  </complexType>
  <!-- Start SharingInfo -->
  <element name="SharingInfo" type="rights:SharingInfoType" />
  <complexType name="SharingInfoType">
    <sequence>
      <element ref="rights:Authorization" minOccurs="0" />
      <element name="MaxDuration" type="rights:Duration" />
      <element name="MinDuration" type="rights:Duration" minOccurs="0" />
      <element name="LifetimeLimit" type="rights:LimitType" minOccurs="0" />
      <element ref="rights:Consumption" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <attribute name="Exclusive" type="boolean" default="false" />
    <attribute name="Id" type="ID" use="optional" />
    <attribute name="Limit" type="integer" use="optional" default="1" />
```

```
    <attribute name="Domain" type="anyURI" default="http://www.idpf.org/↩
        epub/30/lcp-rights#open" />
  </complexType>
</schema>
```

# References


[OCF3] *Open Container Format (OCF) 3.0.*
J. Pritchett et al., 11 October 2011.
http://idpf.org/epub/30/spec/epub30-ocf.html

[OPF3] *EPUB Publications 3.0.*
M. Gylling et al., 11 October 2011.
http://idpf.org/epub/30/spec/epub30-publications.html

[EPUB3] *EPUB 3.0 Overview.*
G. Conboy et al., 11 October 2011.
http://idpf.org/epub/30/spec/

[EPUB-CFI] *EPUB Canonical Fragment Identifier (epubcfi) Specification.*
P. Sorotokin et al., 11 October 2011.
http://idpf.org/epub/linking/cfi/epub-cfi.html

[XML] *Extensible Markup Language (XML) 1.0 (Fifth Edition).*
T. Bray et al., 26 November 2008..
http://www.w3.org/TR/2008/REC-xml-20081126/

[XML DSIG Core] *XML-Signature Syntax and Processing Version 1.1.*
M. Bartel et al., 3 March 2011.
http://www.w3.org/TR/xmldsig-core1/

[XML ENC Core] *XML Encryption Syntax and Processing Version 1.1.*
D. Eastlake et al., 3 March 2011.
http://www.w3.org/TR/xmlenc-core1/

[XML SIG Decrypt] *Decryption Transform for XML Signature.*
M. Hughes et al., 10 December 2002.
http://www.w3.org/TR/xmlenc-decrypt

[X.509] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.*
D. Cooper et al., Internet RFC 5280, May 2008.
http://www.ietf.org/rfc/rfc5280.txt

[XMLNS] *Namespaces in XML 1.0 (Third Edition).*
T. Bray, D. Hollander, A. Layman, R. Tobin, H. Thompson, 8 December 2009.
http://www.w3.org/TR/xml-names

[XML C14N] *Canonical XML Version 1.1.*
J. Boyer et al., 2 May 2008.
http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/

[RFC3394] *Advanced Encryption Standard (AES) Key Wrap Algorithm.*
J. Schaad and R. Housley, September 2002.
http://www.ietf.org/rfc/rfc3394.txt

[FIPS-186-3] *FIPS PUB 186-3: Digital Signature Standard (DSS).*
U. S. Department of Commerce/National Institute of Standards and Technology, June 2009.
http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[PKCS1] *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.*
J. Jonsson and B. Kalinski, RFC 3447 (Informational), February 2003.
http://www.ietf.org/rfc/rfc3447.txt

[XPath] *XML Path Language (XPath) 1.0.*
J. Clark and S. DeRose, 16 November 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116/

[XPath-Filter] *XML-Signature XPath Filter 2.0.*
J. Boyer et al., 8 November 2002.
http://www.w3.org/TR/2002/REC-xmldsig-filter2-20021108/

[XMLSCHEMA-1] *XML Schema Part 1: Structures Second Edition.*
Henry S. Thompson et al., 28 October 2004.
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[XMLSCHEMA-2] *XML Schema Part 2: Datatypes Second Edition.*
Paul V. Biron and Ashok Malhotra, 28 October 2004.
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[RFC2119] *Key Words for use in RFCs to indicate Requirement Levels.*
S. Bradner, Internet RFC 2119, March 1997.
http://www.ietf.org/rfc/rfc2119.txt

[URI] *Uniform Resource Identifiers (URI): generic syntax.*
T. Berners-Lee, R. Fielding, L. Masinter, Internet RFC 3986. January 2005.
http://www.ietf.org/rfc/rfc3986.txt

[UNICODE] *The Unicode Standard, Version 6.1.0.*
The Unicode Consortium, 2012, ISBN 978-1-936213-02-3.
http://www.unicode.org/versions/Unicode6.1.0/

[Deflate] *DEFLATE Compressed Data Format Specification version 1.3.*
P. Deutsch, Internet RFC 1951, May 1996.
http://www.ietf.org/rfc/rfc1951.txt

# Index