1. Introduction

The Unique Identifier Technology Solution (UITS) is an industry initiative that describes a common way to embed metadata into unprotected media files so that it is possible to detect when the metadata is modified. The UITS payload is primarily designed for audit purposes by record labels, artists, and others (Content Owners), but is flexible enough for other uses. It is understood that the embedded metadata can be removed from files, though there should not be any reason to do so.

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 RFC 2119.

Note the v1.1b specification is the same as v1.1 specification, but incorporates the changes and additions from the previously separate Errata and Addendum documents. The XSD has also been simplified.

2. Technical Goals

From an audit perspective, the UITS technical goals are three-fold:

1) Distinguish between different legitimately acquired instances of a single track based on the sale. This is accomplished using an identifier. This means that if user #1 buys a track and user #2 buys the same track, each track instance has a different identifier. At the same time, if for whatever reason a user is able to download a single track multiple times from a single sale (say, due to a pre-specified business arrangement), those downloads should share the same identifier. If, on the other hand, a user buys a given track multiple times on separate occasions, then each purchase must be differentiated either by identifier or by a combination of identifier and timestamp.
2) Identify legitimate payloads and detect when a payload has been modified. It is not necessary to determine what the original payload was, simply that the original payload has been altered.
3) Identify the original distributor, date of sale, and track solely from the payload so that the payload can be verified.

3. Tamper Detection

Tamper detection is accomplished by cryptographically signing a collection of metadata, then inserting both the metadata and signature into the file. Together, the metadata and signature are called the UITS payload. To enable verification that the payload is embedded into the correct file, a cryptographic (one-way) hash or fingerprint is created of the encoded media (e.g. the audio or video portion of the file) and included in the payload. For example, this means the descriptive metadata in an MP3 can be modified but the UITS payload can still be matched with the track. Furthermore, the hash or fingerprint can be pre-computed rather than generated at the time of sale. If any part of the payload is modified, this will become apparent when the signature is checked.

4. UITS Identifier Options

This section describes the different UITS identifier options open to distributors. One or both of these methods MUST be used. Unless other arrangements have been made, these transaction and/or User IDs MUST be reported back to the Content Owner in a standard marketing sales report.

4.1. OPTION 1: Embedded User IDs

In this implementation, the ID component in the UITS payload consists of user-specific information, optionally encoded such that it cannot be directly traced back to the user. The value MUST be static (for that distributor), meaning if a given user buys multiple tracks, the same information is embedded into each. Examples of user-specific information might include: a) an email address or obfuscated email address, b) obfuscated login, or c) a user's service ID number or obfuscated ID number. Any obfuscation method used must be deterministic. If a user buys a single track multiple times, then different purchase timestamps MUST be used. If a user buys a gift for someone else, then the user ID should be that of the recipient rather than the buyer. If no such recipient ID exists (because the recipient is anonymous), then a random, unique ID should be created.

4.2. OPTION 2: Embedded Unique Transaction IDs *(Preferred)*

In this implementation, the ID component of the UITS payload is a globally unique identifier (for that distributor) that identifies a specific customer transaction. This means that if the user purchased 3 tracks and 2 albums, all of the individual files would contain the same transaction ID. If a user is able to download a purchase multiple times, each download MUST contain the same transaction ID. If the user buys the same track multiple times, each purchase MUST have different transaction IDs.

5. Technical Specifications

In all cases, the UITS payload has two primary components: metadata and the associated signature. **Element names ARE case sensitive.** At a high level, the payload structure is

```
<UITS>
        <metadata></metadata>
        <signature></signature>
</UITS>
```

For all of the element descriptions below, the layout is:

> **Title**
> REQUIRED/OPTIONAL. Description of the element and a list of any attributes and valid attribute values.
> `<Example>`

5.1. Encoding

Data contained within a UITS document is encoded using either UTF-8 or 16, as noted by the XML processing header. Note that the only legal white space characters that can appear within the <UITS> element are spaces, tabs, carriage returns, and line feeds.

```
<?xml version="1.0" encoding="UTF-8"?>
```

5.2. UITS

REQUIRED. The UITS element contains a <metadata> element and a <signature> element. The version of the UITS specification is denoted by the namespace. The current version is 1.1. Within the UITS element, namespace notations MAY be included.

For the UITS element only, the namespace notation MUST be included. The namespace must be specified, but the prefix does not necessarily need to be "uits". The elements contained within the UITS element are considered local to the UITS element and MUST NOT contain a namespace qualifier.

```
<uits:UITS xmlns:uits="http://www.udirector.net/schemas/2009/uits/1.1">
```

5.3. Metadata

REQUIRED. This element wraps all of the support metadata below.

```
<metadata>
```

The metadata element contains the following sub-elements, which should be included in the UITS payload in the order listed below.

5.3.1. Nonce

REQUIRED. The nonce is a random value generated at the time of sale whose Base64 encoded length is 8 digits. The nonce is used to randomize the hash computation of the signature value for security purposes.

In the case that a service has the rights to offer redownloads of a given transaction, the nonce in a redownloaded file does NOT have to be the same as the nonce in the original download. All other information related to the track and transaction must be the same as in the original download, however. EXCEPTION: If the redownload has a different bitrate, the media hash will be different.

```
<nonce>QgYnkgYS</nonce>
```

5.3.2. Distributor ID

REQUIRED. This is the retailer or distributor's name or an associated globally unique identifier in clear text. Note that this does not identify the CDN or entity generating and/or embedding the UITS payload, but rather the name of the company that has the record of sale (or its agent) and issues sales reports back to the Content Owner. A distributor that operates in multiple territories may use different IDs in each territory.

```
<Distributor>Music Retailer</Distributor>
```

5.3.3. Date of transaction

REQUIRED. The date and time of the purchase (not download or signature) in ISO 8601 format. For example, a file purchased at 14:15 on the 30th August 2008 in the UK should include the string:

```
<Time>2008-08-30T13:15:04Z</Time>
```

Either UTC, or local time plus offset MUST be used. Further documentation on ISO 8601 can be found at http://www.w3.org/TR/NOTE-datetime

5.3.4. Product ID

REQUIRED. The Product ID identifies the parent collection from which the current item originates. If the entire collection has been purchased or completed during the transaction, then this is indicated through the "completed" attribute. The ProductID type is passed as an attribute. Valid ProductID Types currently include "UPC" or "GRID".

For example, if a user purchases a single track from an album, the Product ID identifies the associated album and the completed attribute is set to "false". If instead the user purchases the entire album, the Product ID still identifies the associated album, but the completed attribute is set to "true." Some retailers support a "complete my album" sale, and for those transactions the completed attribute would also be set to true. Only when the entire collection is completed in the same transaction is the completed attribute set to true.

```
<ProductID type="UPC"
completed="false">00602517758056</ProductID>
```

5.3.5. Asset ID

REQUIRED. The Asset ID identifies an individual element of a product, such as a track or video. The AssetID type is passed as an attribute. Valid AssetID types currently include "ISRC". Delimiters such as dashes MUST NOT be included.

```
<AssetID type="ISRC">ES1700800499</AssetID>
```

5.3.6. Transaction ID

Note: at least one of Transaction ID or User ID is REQUIRED. The transaction ID is the unique identifier for the transaction. Currently the only valid version is "1".

Obfuscation MAY be used so long as the algorithm is deterministic, meaning the same output is calculated given the same input. For example, a standard cryptographic HMAC algorithm may be used. A list MUST be maintained of all issued identifiers to prevent collisions and to assist with breach detection.

```
<TID version="1">39220345237</TID>
```

5.3.7. User ID
Note: at least one of Transaction ID or User ID is REQUIRED. The User ID is the unique identifier for the user. Currently the only valid version is “1”.

Obfuscation MAY be used so long as the algorithm is deterministic, meaning the same output is calculated given the same input. For example, a standard cryptographic HMAC algorithm may be used. A list MUST be maintained of all issued identifiers to prevent collisions and to assist with breach detection.

```
<UID version=”1”>A74GHY8976547B</TID>
```

5.3.8. Media Identifier
REQUIRED. A hash of the media (e.g. the audio, video, etc.) portion of the file MUST be included, such that the UITS payload can be directly tied to the file associated with it. The hash type is passed as an attribute, and the currently valid type is “SHA256”. Metadata fields MUST NOT be included in the hash computation, because it must be possible for users to update standard ID3 or similar tags without affecting the media hash. Note that the hash can be pre-computed so that it is not necessary to compute it at the time of sale.

Instructions for MP3s: The hash value for the audio part of the MP3 is calculated using the hash algorithm over all of the audio frames within the file in the order in which they appear in the file. Non-audio data such as ID3 tags of any version (ID3.1.x through ID3v2.4.x) are excluded. Beware that ID3V1 tags appear following the end of audio, not the beginning. The audio frames all start with an 11-bit syncword, their frame size is calculated in the standard manner from the bitrate, sample rate, and padding. In addition, variable bit-rate (“VBR”) files usually contain a “Xing”, “Info”, or “VBRI” frame, which must be excluded. Such frames appear as the first audio frame, though they do not produce any sound when decoded. Therefore, when they are present, the hash should start at the first byte of the second audio frame. These special VBR frames are documented at: http://gabriel.mp3-tech.org/mp3infotag.html and http://www.codeproject.com/KB/audio-video/mpegaudioinfo.aspx.

Instructions for generating the media hash for other file types are in section 6.3 below. Note that future versions of UITS may support fingerprints, embedded watermarks, or other techniques for identifying audio.

Note that the media hash should be expressed in hex. The XML Schema Definition (XSD) indicates that the media hash can be expressed in base64, but it is standard for hashes to be expressed in hex.

```
<Media algorithm=”SHA256”>A675BD878C8658C99A…</Media>
```

5.3.9. URL
OPTIONAL. The URL field is used for links that are worth transporting in a cryptographically signed manner. The type field specifies the URL type and currently

uses a subset of ID3v2.3 tag names. Options for the type field include WCOM, WCOP, WOAF, WOAR, WOAS, WORS, WPAY, and WPUB, which have the meaning described in the ID3 specification. In addition, a legal URL type is KeyURI, which identifies the public key needed to validate the signature (see the Implementation Details section below). More than one URL element MAY be included.

```
<URL type="WPUB">http://www.universalmusic.com/</URL>
```

5.3.10. Parental Advisory
OPTIONAL. The Parental Advisory field is used to indicate the parental advisory status for the track. The three valid values are "unspecified", "explicit", and "edited". The absence of the PA element can be interpreted as "unspecified".

- **unspecified** means no information about the PA status is being communicated. Absence of a PA field is the same as unspecified.
- **explicit** means that the content to which it is applied would, if sold as a physical product or in its originally released compilation, warrant the application of a parental advisory mark or sticker in the region for which it is originally intended.
- **edited** means that the content is an edited version of original content to which a parental advisory mark or sticker has been applied.

```
<PA>explicit</PA>
```

5.3.11. Copyright Status
OPTIONAL. This field can be used to communicate copyright information about the associated track. The value field can be one of these pre-defined values: "**unspecified**", "**allrightsreserved** ", "**prerelease** ", or "**other**".

- **unspecified** means no information about the copyright status is being communicated. Absence of a copyright field is the same as unspecified.
- **allrightsreserved** means all rights reserved unless otherwise expressly authorized by the content owner.
- **prerelease** means the same as all rights reserved, but additionally, that the content is pre-release material and not intended for release to the public.
- **other** means that the copyright element contains a URL that has more information about the copyright. For example, this might be used for a creative commons license.

```
<Copyright value="allrightsreserved"></Copyright>  or
<Copyright value="other"> http://tinyurl.com/awmp4f</Copyright>
```

5.3.12. Extra
OPTIONAL. The Extra field is used for miscellaneous metadata that is worth transporting in a cryptographically signed manner. The type field specifies the type of metadata and the value field specifies the value. More than one Extra element MAY be included.

```
<Extra type="Foo">Bar</Extra>
```

5.4. Signature

A cryptographic signature is used to verify the integrity of the <metadata> element. The contents of the <metadata> element are signed including the opening and closing <metadata> tags, and the resulting signature is placed in a <signature> element. The <signature> element is then appended after the end of the metadata element. The whole thing is then wrapped in a UITS element (see below for an example).

The signature block consists of five parts. The wrapping <signature> element, three attributes, and content representing the Base64 encoded result of the signature algorithm. The following attributes relate to the mechanism used for creating and interpreting the <signature> value.

- algorithm – identifies the signature and hash algorithms plus any critical configuration details such as padding used to generate the signature. Current options are "RSA2048" and "DSA2048" and distributors can pick either one.
    - RSA2048 means generating the signature with the RSA algorithm using RSASSA-PKCS1 –v1_5 padding as defined in RFC 3447, with a 2048 bit key. The SHA256 algorithm must be used to hash the metadata element.
    - DSA2048 means using the DSA signature algorithm as described in FIPS 186-3 with a 2048 bit key. The SHA224 algorithm must be used to hash the metadata element.
- canonicalization – identifies the XML canonicalization method used to normalize the data contained within the <metadata> element. Currently, the only method supported is "none". This means that when the hash is computed on the <metadata> element, the <metadata> element **must be hashed exactly as it exists in the file including any white space characters (tabs, carriage returns, line feeds, and spaces) that it contains**.
- keyID – an identifier associated with the public/private key pair used by the distributor or CDN for computing the signature value. The keyID MUST be generated according to one of the three methods described below, in order of preference:

    Option 1 (Preferred)

    The keyID is the SHA1 hash of the PEM-encoded public key file sent to the content owner for validation purposes. The hash MUST be of the entire file needed to validate the signature, inclusive of the standard header and footer.

    Example: openssl dgst -sha1 rsa-public-key.pem

Option 2 (For Certificates)

When a certificate is used, the KeyID is generated using the first method described in section 4.2.1.2 of RFC 3280. Specifically, the keyID is composed of the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey (excluding the tag, length, and number of unused bits).

Option 3 (To be deprecated in a later version)

For RSA, the keyID is the hash of the DER-encoded RSAPublicKey (as defined in RFC 3447). For DSA, the keyID is the hash of the DER-encoded DSAPublicKey (as defined in RFC 3279).

Example: openssl dgst -sha1 rsa-public-key.der

Here is an example signature, formatted for readability.

```
<signature algorithm="RSA2048"
     canonicalization="none"
     keyID="b2568ca44decce80066ece19bf47488b42d337ec">
UjBsR09EbGhjZ0dTQUxNQUFBUUNBRU1tQ1p0dU1GUXhEUzhi…
</signature>
```

6. Implementation Details

6.1. Keys

Partners are responsible for generating and maintaining the security of their public/private key pair. Private keys MUST be protected with the same care as highly confidential information. If a partner believes that a key may be compromised, they MUST immediately generate a new key pair and alert any affected content owners.

6.2. Payload creation

For security reasons, the payload MUST only be generated on the distributor's or CDN's servers.

The signature is generated by executing the hash function on the <metadata> block, signing the result using the private key associated with KeyId and encoding the result in Base64. The signature extends over the entire <metadata> element.

```
D1 = Canonicalize("<metadata>…</metadata>")
D2 = Hash(D1)
D3 = Sign(private_key, D2)
D4 = Base64encode(D3)
```

(Note: there is no canonicalization algorithm in the present version of the specification, so D1 results in no change to the contents of the <metadata> element).

The resulting signature is encoded in a <signature> block and combined with the original <metadata> block to create the full <UITS> element. The resulting UITS element MUST parse using the associated XML Schema.

For example, to perform steps D2-D4 above on a UITS <metadata> element in the file payload.txt, the following openSSL commands could be used (assuming the private key is in the file privateRSA.pem file):

```
openssl dgst -sha256 -out sig.bin -sign privateRSA.pem payload.txt
openssl base64 -in sig.bin -out signature.txt
```

The contents of the signature.txt file would then be inserted between the <signature></signature> tags.

6.3. Payload embedding process

If no download manager is used, the payload MUST be embedded on the distributor's or CDN's servers. If a download manager is used, then the payload MAY be embedded by client software, so long as no file is made available to end-users without a payload (i.e., the embedding must happen during the download process, prior to the user being able to access the file). Systems that support client-side embedding MUST be designed to prevent and detect unauthorized access or interference with the embedding process.

The specific area of the file where the UITS payload MUST be embedded varies by file type. In all cases, the UITS payload should be treated as a single monolithic block of data. The difference between file formats is limited to the methods for computing the media hash and embedding the UITS payload.

In all cases, computation of the media hash value should be based on *all* of the media (e.g. audio or audio/video) data and *only* the media data. In compressed audio formats, this includes the header information of each audio frame as it generally contains data that is used to reconstruct the output signal. In uncompressed audio (PCM) formats, the hash is based on the audio data starting from the first true PCM sample, ignoring header data. If zero-padding is used at the end of PCM audio data in order to assure data structure boundaries are multiples of some number of bytes (typically 2 or 4), the padding is included in the hash computation as it is generally impossible to distinguish pad bytes from trailing silence.

6.3.1. MP3

6.3.1.1. Media Hash Generation

The Media Hash Generation for MP3s is described in section 5.3.8 above.

6.3.1.2. UITS Payload Injection

The payload MUST be properly embedded in a PRIV ID3 tag. All frames within the ID3 tag SHOULD be properly formatted.

Common Problems to Avoid

- Actual length of tag and tag length stated in header are not the same
- Extra nulls before owner put owner in position normally occupied by the UITS

The owner value in PRIV frame MUST be:

```
mailto:uits-info@umusic.com
```

Section 5 of the ID3v2.3 informal standard describes an optional extended header construct. ID3v2.3 extended headers MUST NOT be used in conjunction with UITS-embedded files.
Section 5 of the ID3v2.3 informal standard also describes an optional method of providing backward compatibility with older playback software and devices that misinterpret a certain byte sequence in ID3 tags as the beginning of the audio. ID3v.2.3 is not specific about the byte count calculation.

For the purposes of UITS, the byte count calculation must be performed as described in ID3v2.4:

```
[3.1] The ID3v2 tag size is the sum of the byte length of the extended
header, the padding and the frames after unsynchronisation. If a footer
is present this equals to ('total size' - 20) bytes, otherwise ('total
size' - 10) bytes. [3.1]

[4] The frame ID is followed by a size descriptor containing the size
of the data in the final frame, after encryption, compression and
unsynchronisation. The size is excluding the frame header ('total frame
size' - 10 bytes) and stored as a 32 bit synchsafe integer.
```

6.3.2. M4A (AAC)

6.3.2.1.Media Hash Generation

The hash value for the audio part of the AAC (or variant) file is calculated using the hash algorithm over the entirety of all of the audio frames (and only the audio frames, including their individual frame headers) in the order in which they appear in the file.

The hash for the AAC audio frame data is generated using the following algorithm:

1. Parse the file until the 'mdat' atom is found
2. Hash the data in the 'mdat' atom

6.3.2.2.UITS Payload Injection

The UITS payload is injected as a 'uuid ' atom at the end of the file.

The UITS uuid is a 128 bit integer, the value of which must be (represented here in hex):

"99454E27963A4B568E761DB68C899CD4".

The uuid atom has the following format:

size
'uuid'
99454e27963a4b568e761db68c899cd4
UITS payload

6.3.3. WAV

6.3.3.1. Media Hash Generation

The hash value for the audio part of the WAV file is calculated using the hash algorithm over the entire PCM 'data' chunk, excluding any chunk overhead, but starting from the first sample word to the last, exclusive of any zero padding.

The hash value for the audio frames is calculated using the following algorithm:

1. Find the 'data' chunk in the WAV file
2. Hash the audio frames in the data chunk, not including the pad byte if it exists

6.3.3.2. UITS Payload Injection

WAV files are a subset of the RIFF specification, which allows the creation of arbitrary new chunks by applications so long as (much like the AIFF variant of IFF) the chunk size value in the container chunk is updated to reflect the new size of the contents. Therefore, UITS is added by appending a 'UITS' chunk to the end of a file, and updating the container as necessary.

The UITS payload is injected into the file using the following algorithm:

1. Copy the contents of the input file to the output file
2. Append a 'UITS' chunk containing the UITS payload
3. Update the size of the RIFF chunk to reflect the additional 'UITS' chunk

6.3.4. AIFF

6.3.4.1. Media Hash Generation

The hash value for the audio part of the AIFF file is calculated using the hash algorithm over the entire SSND (Sampled Sound) chunk, excluding the 8 bytes of chunk overhead, but starting at the first actual PCM sample word, until the last, inclusive of any required zero padding at the end.

6.3.4.2. UITS Payload Injection

Files delivered to customers in AIFF (or its AIFC variant) format should include the UITS payload in an "Application Specific" chunk, identified by the Chunk ID 'APPL'. The APPL chunk must use an "OSType" string of 'UITS'. The AIFF specification allows any number of APPL chunks in a file, so care should be taken to place only one UITS payload (in an APPL chunk) in any file, and applications should not replace, delete or otherwise modify any other APPL chunks which are present in the file.

The AIFF specification provides a clear explanation of the logical organization of AIFF files. Note that when a UITS payload is inserted into a file, the file size values near the top of the FORM data structures must be updated accordingly to reflect the increased size. Note also that the UITS data can appear in an APPL chunk near the end of the file as easily as the beginning, so it can be appended instead of inserted, as long as the wrapper (FORM) chunks are updated.

6.3.5. FLAC

6.3.5.1. Media Hash Generation

The hash value for the audio part of the FLAC file is calculated using the hash algorithm over the entire contiguous sequence of audio frames, from the first frame to the last, inclusive of any frame overhead (non-sample) data.

6.3.5.2. UITS Payload Injection

FLAC files use the OGG format, which permits arbitrary application-specific data in a metadata block of type “Application”. The UITS payload should be stored in such a block, using the application ID “UITS”, treated as a 32-bit integer (big-endian).

6.3.6. MPEG4 Video Files & Quicktime

MPEG4 and QuickTime files are supported the same way as M4A (AAC) files. See the description in section 6.3.2 of this document. The hash is calculated over the entirety of the media frames in the mdat atom.

6.3.7. HTML Files

6.3.7.1. Media Hash Generation

The media hash for an HTML file is generated by hashing the entire file, not including a UITS payload. Note that in order to verify an HTML file that contains a UITS payload, the payload must be removed from the HTML before calculating the media hash for verification.

6.3.7.2. UITS Payload Injection

The UITS payload is injected into the HTML file at the end of the <HEAD> element. It should be inserted right before the </HEAD> tag. The UITS payload should have no extra whitespace (such as carriage returns or line feeds).

Here is a sample HTML file before the UITS payload is injected:

```
<html>
<head>
        <title>UITS HTML Test File</title>
</head>
<body>
<h1> This is a test </h1>
</body>
</html>
```

Here is the same file with the UITS injected:

```
<html>
<head>
  <title>UITS HTML Test File</title>
<uits:UITS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:uits="http://www.udirector.net/schemas/2009/uits/1.1"><metadata><no
nce>QgYnkgYS</nonce><Distributor>A</Distributor><Time>2008-08-
30T13:15:04Z</Time><ProductID type="UPC"
completed="true">My</ProductID><AssetID
type="ISRC">ES1700800500</AssetID><TID version="1">Transaction</TID><UID
version="1">User</UID><Media
algorithm="SHA256">d9f6a8f128b1e7e555a244a736c199da0f808224ee2fb49a0a9546
324355820e</Media><URL
type="WPUB">http://www.umusic.com</URL><PA>explicit</PA><Copyright
value="allrightsreserved">allrightsreserved</Copyright><Extra
type="blah">extra</Extra></metadata><signature algorithm="RSA2048"
canonicalization="none"
keyID="33dce5a4f8b67303a290dc5145037569ca38036d">HAiPq0eaLDPaxyeRWDd1m+DD
DBSipFNJkL8PLpQeOsg8ykplrxdE5G2jO0nDS3jbtxl36Pt0anjscR6Jw172aM/u2UqhEVKKu
UnG2x8sBxs2KzZft97BA5Us9ISGyVCSizp3rbSv9EJ9ylHMnHGtCzCx2sR1qhYEk9UVRQHZlB
uMlrhcHoPCNDEktv99PbQqzSWgI5DmDRTxFPQovKB2AYuXkmK0J92Qcp5c/o97VXkWvo57biI
1wwYV993ZvtTTJEyi74iP8cmaZr7lwrvtXrUlhiTeSnkMhUB0F25RaosFmueQmHCvKAsJgaXX
32f8ON5/EgmXY4nTHRIvbCRWmg==</signature></uits:UITS></head>
<body>
<h1> This is a test </h1>
</body>
</html>
```

7. Example UITS Payload

Note the hash and signature values are not real, and formatting is for legibility only.

```
<?xml version="1.0" encoding="UTF-8"?>
<uits:UITS xmlns:uits="http://www.udirector.net/schemas/2009/uits/1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <metadata>
      <nonce>QgYnkgYS</nonce>
      <Distributor>Music Retailer</Distributor>
      <Time>2001-12-17T09:30:47.0Z</Time>
      <ProductId type="UPC"
                 completed=”false”>00602517178656</ProductId>
      <AssetID type="ISRC">USUV70603512</AssetID>
      <TID version="1">39220345237</TID>
      <UID version="1">A74GHY8976547B</UID>
      <URL type=”WPUB”>http://www.umusic.com</URL>
      <Media algorithm="SHA256">
      d5b17cc1975d3095c6353f3fdced45ae867c06e02c1efb7c09662cdc796724b0
      </Media>
   </metadata>
   <signature algorithm=”RSA2048”
         canonicalization="none"
         keyID="b2568ca44decce80066ece19bf47488b42d337ec">
iQEcBAEBAgAGBQJJ0S2XAAoJECY0BcYmEHZbpjcIAJNKgwFg93vvCxfn6cDVthZr
ERjp1gq1Nqt6vcBBP2zfdhgsA6eay4EMpM7K9o1cpN7UH1ksnFTpeniUrdfZs9FW
```

```
RTusFwTSHv82D1ZJQ3mvq0r25KKrtAB7FUz2G3XFP8zE/TdUCLjO7A0DhME9sytl
yZrPyZlraJGpfzUK9lRLY4IfupBnPDhav9Quycfoq2Kfnv2in/8PMvZdQH0Kim9b
z02gWsRijoMM7Vy7esBPLeXA++oOo28H7cnd9+BaZ5wtxbxXGg9G5pAWwdW45Ddi
2q+GpiUCqeHbqnucsHVfa2wyP7b6ASTUSzZHJEGle7UkHm057fCCa72n6vqTNfA=
=VYk6
   </signature>
</uits:UITS>
```

8. Verification

The <metadata> element MUST validate if it is directly copied and pasted into a file, treating white space as indicated in the canonicalization rules, and then checked against the signature hash.

To validate the <metadata> block:

```
D1 = Canonicalize("<metadata>…</metadata>")
D2 = Base64decode(signature)
D3 = Verify(public_key, D1, D2)
```

(Note: there is no canonicalization algorithm in the present version of the specification, so D1 results in no change to the contents of the <metadata> element).

For example, to do steps D2-D3 above and verify the signature using openSSL, you could do the following (assuming the payload was in payload.txt, the signature was in signature.txt, and the public key was in the pubRSA.pem file):

```
openssl base64 -d -in signature.txt -out sig.bin
openssl dgst –sha256 -verify pubRSA.pem -signature sig.bin payload.txt
```

9. Appendix – XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:uits="http://www.udirector.net/schemas/2009/uits/1.1"
targetNamespace="http://www.udirector.net/schemas/2009/uits/1.1">
  <xs:element name="UITS">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="metadata">
            <xs:complexType>
                <xs:sequence>

                    <xs:element name="nonce">
                        <xs:annotation>
                            <xs:documentation>The nonce is an 8 digit Base64 encoded random
value generated at the time of sale that is used to randomize the hash computation of the
signature value for security purposes.</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:length value="8"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:element>

                    <xs:element name="Distributor" type="xs:string">
                        <xs:annotation>
                            <xs:documentation>This is the retailer or distributor’s name or an
associated identifier in clear text.  Note that this does not identify the CDN or entity
generating and/or embedding the UITS payload, but rather the name of the company that has the
record of sale (or its agent) and issues sales reports back to the Content
Owner.</xs:documentation>
                        </xs:annotation>
                    </xs:element>

                    <xs:element name="Time" type="xs:dateTime">
                        <xs:annotation>
                            <xs:documentation>The date and time of the purchase (not download
or signature) in ISO 8601 format. </xs:documentation>
                        </xs:annotation>
                    </xs:element>

                    <xs:element name="ProductID">
                        <xs:annotation>
                            <xs:documentation>The Product ID identifies the parent collection
from which the current item originates. If the entire collection has been purchased or
completed during the transaction, then this is indicated through the “completed” attribute.
The ProductID type is passed as an attribute. Valid ProductID Types currently include "UPC" or
"GRID". For example, if a user purchases a single track from an album, the Product ID
identifies the associated album and the completed attribute is set to "false".  If instead the
user purchases the entire album, the Product ID still identifies the associated album, but the
completed attribute is set to "true."  Some retailers support a "complete my album" sale, and
for those transactions the completed attribute would also be set to true.  Only when the
entire collection is completed in the same transaction is the completed attribute set to
true.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:string">
                                    <xs:attribute name="type" use="required">
                                        <xs:simpleType>
                                          <xs:restriction base="xs:NCName">
                                          <xs:enumeration value="UPC"/>
                                          <xs:enumeration value="GRID"/>
                                          </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                    <xs:attribute name="completed" type="xs:boolean"/>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>

                    <xs:element name="AssetID">
                        <xs:annotation>
                            <xs:documentation>The Asset ID identifies an individual element of
a product, such as a track or video.  The AssetID type is passed as an attribute.  Valid
AssetID types currently include “ISRC”. Delimiters such as dashes MUST NOT be included.
</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:string">
                                    <xs:attribute name="type" use="required">
                                        <xs:simpleType>
                                          <xs:restriction base="xs:NCName">
                                          <xs:enumeration value="ISRC"/>
                                          </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>

                    <xs:choice maxOccurs="2">
                        <xs:annotation>
                            <xs:documentation>At least one of TID or UID is
REQUIRED.</xs:documentation>
                        </xs:annotation>
                        <xs:element name="TID">
                            <xs:annotation>
                                <xs:documentation>The transaction ID is the unique identifier
for the transaction.  Currently the only valid version is “1”.  Obfuscation MAY be used so
long as the algorithm is deterministic, meaning the same output is calculated given the same
input. For example, a standard cryptographic HMAC algorithm may be used.</xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                                <xs:simpleContent>
                                    <xs:extension base="xs:string">
                                        <xs:attribute name="version" use="required">
                                          <xs:simpleType>
                                          <xs:restriction base="xs:unsignedInt">
                                          <xs:enumeration value="1"/>
                                          </xs:restriction>
                                          </xs:simpleType>
                                        </xs:attribute>
                                    </xs:extension>
                                </xs:simpleContent>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="UID" minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>The User ID is the unique identifier for the
user.  Currently the only valid version is “1”. Obfuscation MAY be used so long as the
```

```
algorithm is deterministic, meaning the same output is calculated given the same input. For
example, a standard cryptographic HMAC algorithm may be used. Note TID is the preferred
identification method rather than UID.</xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                                <xs:simpleContent>
                                    <xs:extension base="xs:string">
                                        <xs:attribute name="version" use="required">
                                          <xs:simpleType>
                                          <xs:restriction base="xs:unsignedInt">
                                          <xs:enumeration value="1"/>
                                          </xs:restriction>
                                          </xs:simpleType>
                                        </xs:attribute>
                                    </xs:extension>
                                </xs:simpleContent>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>

                    <xs:element name="Media">
                        <xs:annotation>
                            <xs:documentation>A hash of the binary media asset. The only valid
algorithm attribute value is "SHA256". Metadata fields from the binary file MUST NOT be
included in the hash computation, because it must be possible for users to update standard ID3
or similar tags without affecting the audio hash.   </xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:base64Binary">
                                    <xs:attribute name="algorithm" use="required">
                                        <xs:simpleType>
                                          <xs:restriction base="xs:NCName">
                                          <xs:enumeration value="SHA256"/>
                                          </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>

                    <xs:element name="URL" maxOccurs="unbounded" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>The URL field is used for links that are worth
transporting in a cryptographically signed manner.  The type field specifies the URL type and
currently uses a subset of ID3v2.3 tag names.  Options for the type field include WCOM, WCOP,
WOAF, WOAR, WOAS, WORS, WPAY, and WPUB, which have the meaning described in the ID3
specification.  Another legal URL type is KeyURI, which identifies the public key needed to
validate the signature.  More than one URL element MAY be included.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:anyURI">
                                    <xs:attribute name="type" use="required">
                                        <xs:simpleType>
                                          <xs:restriction base="xs:NCName">
                                          <xs:enumeration value="WCOM"/>
                                          <xs:enumeration value="WCOP"/>
                                          <xs:enumeration value="WOAF"/>
                                          <xs:enumeration value="WOAR"/>
                                          <xs:enumeration value="WOAS"/>
                                          <xs:enumeration value="WORS"/>
                                          <xs:enumeration value="WPAY"/>
                                          <xs:enumeration value="WPUB"/>
                                          <xs:enumeration value="KeyURI"/>
                                          </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>

                    <xs:element minOccurs="0" name="PA">
                        <xs:annotation>
                            <xs:documentation>The Parental Advisory field is used to indicate
the parental advisory status for the track. The three valid values are “unspecified”,
“explicit”, and “edited”.  The absence of the PA element can be interpreted as “unspecified”.
</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                            <xs:restriction base="xs:NCName">
                                <xs:enumeration value="unspecified"/>
                                <xs:enumeration value="explicit"/>
                                <xs:enumeration value="edited"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:element>

                    <xs:element minOccurs="0" name="Copyright">
                        <xs:annotation>
                            <xs:documentation>This field can be used to communicate copyright
information about the associated track.  The value field can be one of these pre-defined
values: “unspecified”, “allrightsreserved ”, “prerelease ”, or “other”. The URI is only
expected when the value attribute is "other"</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:anyURI">
                                    <xs:attribute name="value" use="required">
                                        <xs:simpleType>
                                          <xs:restriction base="xs:NCName">
                                          <xs:enumeration value="unspecified"/>
                                          <xs:enumeration value="allrightsreserved"/>
                                          <xs:enumeration value="prerelease"/>
                                          <xs:enumeration value="other"/>
                                          </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>

                    <xs:element name="Extra" maxOccurs="unbounded" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>The Extra field is used for miscellaneous
metadata that is worth transporting in a cryptographically signed manner.  The type field
specifies the type of metadata and the value field specifies the value. More than one Extra
element MAY be included.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:string">
```

```
                                    <xs:attribute name="type" type="xs:string"
use="optional"/>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>

                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:element name="signature">
            <xs:annotation>
                <xs:documentation>Contains the base64 encoded signature. The algorithm
attribute is one of RSA2048 or DSA2048.  Canonicalization is currently only
"none".</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="xs:base64Binary">
                        <xs:attribute name="algorithm" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:NCName">
                                    <xs:enumeration value="RSA2048"/>
                                    <xs:enumeration value="DSA2048"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="canonicalization" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:NCName">
                                    <xs:enumeration value="none"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="keyID" type="xs:string" use="required"/>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```