**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 3**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 3 m** **56343**

**Online – April 2021**

**Title: Proposed additional updates in ISO/IEC 14496-22 AMD2**

**Authors: Peter Constable, Vladimir Levantovsky (on behalf of AHG)**

This contribution proposes additional updates to various parts of the ISO/IEC 14496-22:2019 OFF (4th edition) that are based on the issues raised by the AHG members either on the AHG list or at MPEG\_OFF GitHub.

*5.7.1 “**DSIG – Digital signature table”*

*Replace the content of the entire subclause with the following:*

The DSIG table contains the digital signature of the OFF font. Signature formats are widely documented and rely on a key pair architecture. Software developers, or publishers posting material on the Internet, create signatures using a private key. Operating systems or applications authenticate the signature using a public key.

The W3C and major software and operating system developers have specified security standards that describe signature formats, specify secure collections of web objects, and recommend authentication architecture. OFF fonts with signatures will support these standards.

OFF fonts offer many security features:

* Operating systems and browsing applications can identify the source and integrity of font files before using them,
* Font developers can specify embedding restrictions in OFF fonts, and these restrictions cannot be altered in a font signed by the developer.

The enforcement of signatures is an administrative policy that may be supported by the host environment in which fonts are used. Systems may restrict use of unsigned fonts, or may allow policy to be controlled by a system administrator.

Anyone can obtain identity certificates and encryption keys from a certifying agency, such as Verisign or GTE's Cybertrust, free or at a very low cost.

The DSIG table is organized as follows. The first portion of the table is the header.

*DSIG Neader*

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| uint32 | version | Version number of the DSIG table (0x00000001) |
| uint16 | numSignatures | Number of signatures in the table |
| uint16 | flags | Must be set to 0x0001 |
| SignatureRecord | signatureRecords[munSignatures] | Array of signature records |

The version of the DSIG table is expressed as a uint32, beginning at 0. The version of the DSIG table currently used is version 1 (0x00000001).

Permission bit 0 allows a party signing the font to prevent any other parties from also signing the font (counter-signatures). If this bit is set to zero (0) the font may have a signature applied over the existing digital signature(s). A party who wants to ensure that their signature is the last signature can set this bit.

The DSIG header has an array of signature records, which specifying the format and offset of signature blocks.

*SignatureRecord*

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| uint32 | format | Format of the signature |
| uint32 | length | Length of signature in bytes |
| Offset32 | signatureBlockOffset | Offset to the signature block from the beginning of the table |

Signatures are contained in one or more signature blocks. Signature blocks may have various formats; currently one format is defined. The format identifier specifies both the format of the signature block, as well as the hashing algorithm used to create and authenticate the signature.

*Signature Block Format 1*

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| uint16 | reserved1 | Reserved for future use; set to zero. |
| uint16 | reserved2 | Reserved for future use; set to zero. |
| uint32 | signatureLength | Length (in bytes) of the PKCS#7 packet in the signature field. |
| uint8 | signature[signatureLength] | PKCS#7 packet |

For more information about PKCS#7 signatures see [10].

For more information about counter-signatures, see [11].

**Format 1: For whole fonts, with either TrueType outlines and/or CFF data**

PKCS#7 or PKCS#9. The signed content digest is created as follows:

1. If there is an existing DSIG table in the font:
   1. Remove the DSIG table from font.
   2. Remove the DSIG table entry from the Table Directory.
   3. Adjust table offsets as necessary.
   4. Recalculate the checksumAdjustment in the ‘head’ table.
2. Hash the revised font data using a secure one-way hash (such as MD5) to create the content digest.
3. Create the PKCS#7 signature block using the content digest.
4. Create a new DSIG table containing the signature block.
5. Add the DSIG table to the font, adjusting table offsets as necessary.
6. Add a DSIG table entry to the Table Directory.
7. Recalculate the checksumAdjustment in the ‘head’ table.

Validation of a signature in a font is done by repeating steps 1 – 4 in an in-memory copy of the font file. Note that changing the checksumAdjustment in the last step does not break the signature because verification is done on an in-memory copy with these changes.

Prior to signing a font file, ensure that all the following attributes are true:

* The magic number in the ‘head’ table is correct.
* Given the numTables value in the Table Directory, the other values in the Table Directory are consistent.
* The table records in the Table Directory are ordered alphabetically by the table tags, and there are no duplicate tags.
* The offset of each table is a multiple of 4. (That is, tables are long word aligned.)
* The first actual table in the file comes immediately after the directory of tables.
* If the tables are sorted by offset, then for all tables i (where index 0 means the table with the smallest offset), Offset[i] + Length[i] <= Offset[i+1] and Offset[i] + Length[i] >= Offset[i+1] - 3. In other words, the tables do not overlap, and there are at most 3 bytes of padding between tables.
* The pad bytes between tables are all zeros.
* The offset of the last table in the file plus its length is not greater than the size of the file.
* The checksums of all tables are correct.
* The ‘head’ table's checksumAdjustment field is correct.

**Signatures for Font Collections**

The DSIG table for a Font Collection (TTC) must be the last table in the TTC file. The offset to the table is put in the TTCHeader (version 2). Signatures of TTC files are expected to be Format 1 signatures.

The signature of a TTC file applies to the entire file, not to the individual fonts contained within the TTC. Signing the TTC file ensures that other contents are not added to the TTC.

Individual fonts included in a font collection should not be individually signed as the process of making the TTC could invalidate the signature on the font.

When DSIG table is created for a collection file, the steps given above are used, with these revisions:

* In step 1: if there is an existing DSIG table referenced in a version 2.0 TTC header, the DSIG table is removed, and the DSIG fields in the header is set to NULL. No recalculation of a checksumAdjustment is required.
* In steps 6 and 7: the DSIG table is added to the file, not to any individual font within the collection. A version 2.0 TTC header is required, with the DSIG fields in the header set to reference the DSIG table.
* Step 8 is not applicable.

See the TTC Header description (subclause 4.6.3) for related information.

*6.4.2*

*In the Language systems and tags table, in the row defining the language system tag ‘SXT ‘ - replace the “Corresponding ISO 639 ID” field with the following:*

ngo, xnj, xnq

*Add the following additional changes:*

*7.1.7*

*In the end of the sub-clause 7.1.7 “Algorithm for interpolation of instance values”, add the following paragraph:*

In calculation of scalars (S, AS) and of interpolated values (scaledDelta, netAjustment, interpolatedValue), at least 16 fractional bits of precision should be maintained. Rounding should be done only when the final result is used, and may retain greater fractional bit-depth than that of the data type of the item to which deltas are applied. When scaled deltas are applied to a default value, the possibility of overflow exists. The integer bit-depth used in calculation must be at least that of the data type of the item to which deltas are applied; for example, at least 16 integer bits when applying scaled deltas to an FWORD value. Even if a final result is within range for the type of the default value, the possibility remains for overflow to occur at intermediate steps in calculation, and for the order in which deltas are added to affect the result. Sufficient integer bit-depth should be used to avoid any possible overflow at intermediate steps during calculation. An implementation-determined representation may be used for the final result (interpolatedValue), but saturation arithmetic must be used: values must not wrap from maximum to minimum values, or vice versa. The final result should be clamped to the range of the data type of the item to which deltas are applied; for example, [-32768, 32767] for an FWORD value.

*7.2.1 “COLR table and OFF Font Variations”*

*Replace the fifth paragraph with the following:*

A variable font includes a large number of deltas. At the highest level, deltas are organized into collections for different target item sets:

* Deltas for positions of points of a ‘glyf’ table are stored in a ‘gvar’ table.
* Deltas for positions of points of a ‘CFF2’ table are stored within the ‘CFF2’ table.
* Deltas for CVT values are stored in a ‘cvar’ table.
* Deltas for glyph metrics in an ‘hmtx’ table are stored in an ‘HVAR’ table; and deltas for glyph metrics in a ‘vmtx’ or ‘VORG’ table are stored in a ‘VVAR’ table.
* Deltas for anchor positions in ‘GPOS’ lookups and other items used in ‘GDEF’, ‘GPOS’ or ‘JSTF’ tables are stored within variation data contained in the ‘GDEF’ table.
* Deltas for font-wide metrics and other items from the ‘OS/2’, ‘hhea’, ‘gasp’ or other tables are stored in an ‘MVAR’ table.
* Deltas for values in other tables are stored in the respective table: deltas for baseline metrics in the ‘BASE’ table and for various items in the ‘COLR’ table are stored in each table.

*7.2.3*

*In 7.2.3.4 (previously 7.2.3.2), “Item variation store and item variation data tables”, replace the second paragraph and the NOTE with the following:*

The item variation store includes an offset to a variation region list and an array of offsets to item variation data subtables. A NULL offset in the array indicates that there is no item variation data subtable for that index into the array.

NOTE Indices into the itemVariationDataOffsets array are stored in parent tables as delta-set “outer” indices with each such index having a corresponding “inner” index. If the outer index points to a NULL offset, then any inner index will be invalid and can be ignored: items associated with this index do not have any variation.

*In 7.2.3.4 (previously 7.2.3.2), “Item variation store and item variation data tables”, replace the fifth paragraph with the following:*

The representation of delta values uses a mix of long types (“words”) and short types. If the LONG\_WORDS flag is set, deltas are represented using a mix of int32 and int16 values. If the flag is not set, deltas are presented using a mix of int16 and int8 values. See the description of the DeltaSet record below for additional details.