Collabora

Handling PDF digital signatures with PDFium

By Miklos Vajna

Software Engineer at Collabora Productivity

2021-02-07

About Miklos

From Hungary

- More details: https://www.collaboraoffice.com/about-us/

Google Summer of Code 2010 / 2011

- Rewrite of the Writer RTF import/export

Then a full-time LibreOffice developer for SUSE

Now a contractor at Collabora

Handling PDF digital signatures in LibreOffice with PDFium

The digital signing matrix

(via techcrunch.com)

“The” document signing has many factors

- Signing or verification
- Visible signatures or invisible ones
- Different document formats:
  PDF and editable formats
- Different platforms: NSS and MSCNG
- Different certificate types: X509 or GPG
- Different encryption algorithms: ECDSA or RSA
- Different hash algorithms: e.g. SHA-1 or SHA-256
- When it “doesn’t work”: several combinations

Document formats: PDF, ODF and OOXML

Initially just ODF, then PDF and OOXML

- Verification:
  - Check if the digest (hash) matches
  - Validate the certificate
  - Check if the whole document is signed
- PDF: tricky
  - Need incremental updates for multiple signatures
  - Want to detect modify-after-sign
- OOXML is ugly, leaks your details:

```
<WindowsVersion>6.1</WindowsVersion>
<OfficeVersion>16.0</OfficeVersion>
<ApplicationVersion>16.0</ApplicationVersion>
<Monitors>1</Monitors>
<HorizontalResolution>1280</HorizontalResolution>
<VerticalResolution>800</VerticalResolution>
<ColorDepth>32</ColorDepth>
```

Platforms: NSS and MSCNG

(via mozilla.org)

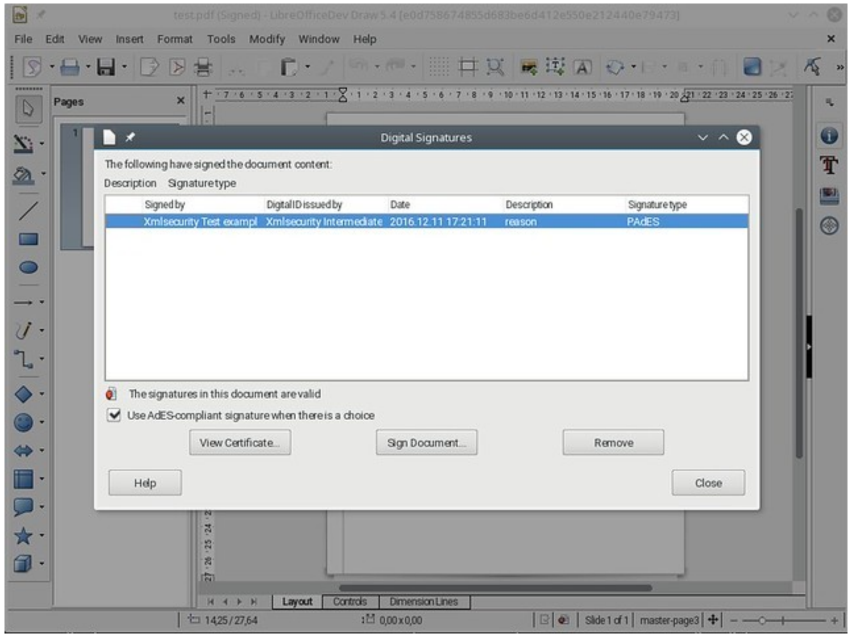
Multiple crypto backends in xmlsecurity/

- Not own crypto, just using NSS on Linux
  - Is this certificate valid?
  - Tricky question, delegate the decision to Mozilla
- MSCNG on Windows
  - CryptoAPI for certificate handling
  - CNG for actual hashing and encryption
    - CryptoAPI itself doesn't support ECDSA

Certificate types: X509 and GPG

PDF and OOXML mandates X509

- ODF supports both, see the GPG effort from CIB
- Governments like to give e-IDs to citizens
    - These are also based on X509



The following have signed the document content:

Description Signature type

Signed by	Digital ID issued by	Date	Description	Signature type
Xmlsecurity Test exampl	Xmlsecurity Intermediate	2016.12.11 17:21:11	reason	PAdES

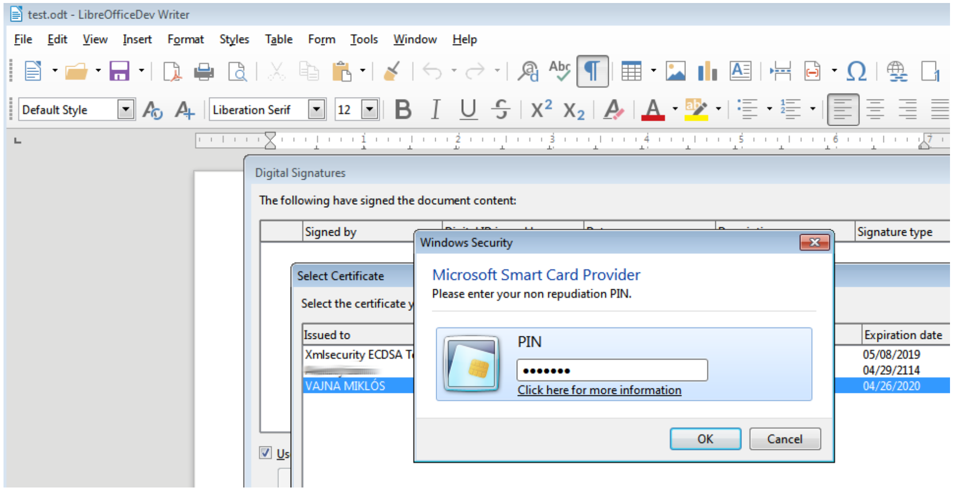
The signatures in this document are valid

[✓] Use AdES-compliant signature when there is a choice

Underlying encryption: RSA and ECDSA

RSA worked even back in the OOo times

- ECDSA is more tricky
- All XML-based signing is done via libxmlsec:
  - Its mscrypto backend used CryptoAPI → no ECDSA
  - A whole new mscng backend was needed in libxmlsec
  - Now it's on par with NSS
- Works nicely with e.g. my own Hungarian e-ID :-)
  - i.e. not only with software certificates



The following have signed the document content:

Microsoft Smart Card Provider

Please enter your non repudiation PIN.

Select the certificate y

Issued to	Expiration date
Xmlsecurity ECDSA T	05/08/2019
[redacted]	04/29/2114
VAJNA MIKLÓS	04/26/2020

Underlying hash algorithm: MD* or SHA*

OOo defaulted to SHA1

- Nowadays only SHA-256 is considered to be modern
- Needed to upstream the huge patchset of libxmlsec
  - Then could upgrade libxmlsec to a modern version
  - Which gives SHA-256 support for free

PDF signature verification

(via ascertia)

Using an own tokanizer first, nothing provided what we needed:

- Poppler was out of process, painful
- PDFium did not have a signature API
- We had an own boost spirit-based tokenizer to detect hybrid PDFs (embedded ODF)
  - Very hard to modify and maintain
- vcl::filter::PDFDocument:
  - clang-style close tracking of each parsed token
  - Provides just what’s necessary to verify and create PDF signatures
  - Later reused to copy PDF images into a PDF export result as-is

Verification with PDFium: PDFium side

Provide a whole set of new PDFium (from Chrome) APIs:

- https://pdfium.googlesource.com/pdfium/+/refs/heads/master/public/fpdf_signature.h
- Get signature objects
- Get signature properties:
  - Content: PKCS#7 blob
  - ByteRange: offset + size of signed data blocks
  - SubFilter: how to parse the content
  - Reason/comment
  - Timestamp

(via wikipedia.org)

Verification with PDFium: LibreOffice side

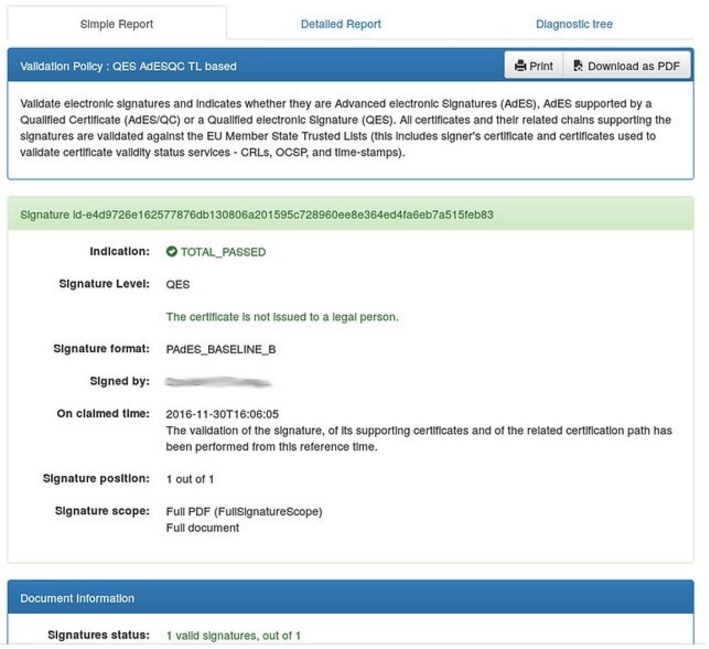
Focus on the most painful part: implicit verification on every file open

- A more battle-tested tokenizer is a huge win here
- If a problem requires explicit user interaction, it’s much less interesting
- Idea: use PDFium to extract the info from the PDF file
- Keep our own code to actually verify the signature (offload to real crypto libs)
- Keep our existing certificate verification code (continue to delegate the decision to Mozilla/Microsoft)

Verification with PDFium: benefits

Immediate access to all those tiny little bugfixes from PDFium

- Sample document where the old tokanizer failed: xmlsecurity/qa/unit/pdfsigning/data/good-custom-magic.pdf
- Junk between the PDF header and the first PDF object
  - We rejected that previously, to be on the safe side
- Additional benefits:
  - Can detect modify-after-sign better: unsigned incremental updates between signatures
  - Can also detect comment-only incremental updates after signing
    - Those are valid, but hard to detect without PDFium



Validation Policy : QES AdESQC TL based

Validate electronic signatures and indicates whether they are Advanced electronic Signatures (AdES), AdES supported by a Qualified Certificate (AdES/QC) or a Qualified electronic Signature (QES). All certificates and their related chains supporting the signatures are validated against the EU Member State Trusted Lists (this includes signer's certificate and certificates used to validate certificate validity status services - CRLs, OCSP, and time-stamps).

Signature Id-e4d9726e162577876db130806a201595c728960ee8e364ed4fa6eb7a515feb83

Indication:	✓ TOTAL_PASSED
Signature Level:	QES
	The certificate is not issued to a legal person.
Signature format:	PAdES_BASELINE_B
Signed by:	[redacted]
On claimed time:	2016-11-30T16:06:05 The validation of the signature, of its supporting certificates and of the related certification path has been performed from this reference time.
Signature position:	1 out of 1
Signature scope:	Full PDF (FullSignatureScope) Full document

Document Information

Signatures status:	1 valid signatures, out of 1

How is this implemented?

PDFium side: implementation

PDFium internal C++ API had this information mostly already

- Just adding wrapper stable C APIs around these
- Tricky case: detecting incremental updates
- PDF is normally read from end, to find the trailer
  - Then that refers to the latest version of all objects
- Normally the tokenizer doesn’t even read previous trailers
- New special mode is added in PDFium to detect all trailer ends
  - Needed to detect unsigned and non-commenting incremental updates after signing

PDFium side: documentation, testing

All new PDFium APIs need:

- Manually written PDF test file template (no redundant file offsets)
- Generate a minimal, yet valid PDF “binary” from it
- googletest testcase asserting correct behavior
  - And test the various failure modes
- Documentation on the intended behavior
  - e.g. is the returned UTF-16 string little endian?

LibreOffice side: implementation

Had to do this incrementally

- First, pdfium doesn’t depend on any crypto libraries
  - So all code only has unit-tests, no integration tests
  - pdfiumsig: external cmdline tool that does integration tests with NSS
- Then separate usage of vcl::filter::PDFDocument in xmlsecurity/
  - Into a single xmlsecurity/source/helper/pdfsignaturehelper.cxx
- Finally switch from vcl::filter::PDFDocument to PDFium APIs
- Clean-up: switch to vcl::pdf::PDFiumDocument, which is a C++ wrapper around the PDFium C APIs

LibreOffice side: testing

The old verifier had good coverage, so this should be safe...

- CppunitTest_xmlsecurity_pdfsigning in xmlsecurity/ gets a new testGoodCustomMagic()
- Something that failed with the old tokenizer
- Then random manual testing with random signed PDF invoices I get, so far so good :-)

Thanks

Collabora is an open source consulting and product company

- What we do and share with the community has to be paid by someone

The Dutch Ministry of Defense in cooperation with Nou&Off

- Made most of this this work by Collabora possible

(via nouenoff.nl)

Summary

Good digital signature support of ODF, OOXML & PDF

- Including signature descriptions, XAdES & PAdES
- Modern hash & encryption algorithms: SHA-256 & ECDSA
- Interoperable with MS Office & Adobe Acrobat
- Latest news is visible PDF signatures & PDFium

Thanks for listening! :-)

- Slides: https://people.collabora.com/~vmiklos/slides/