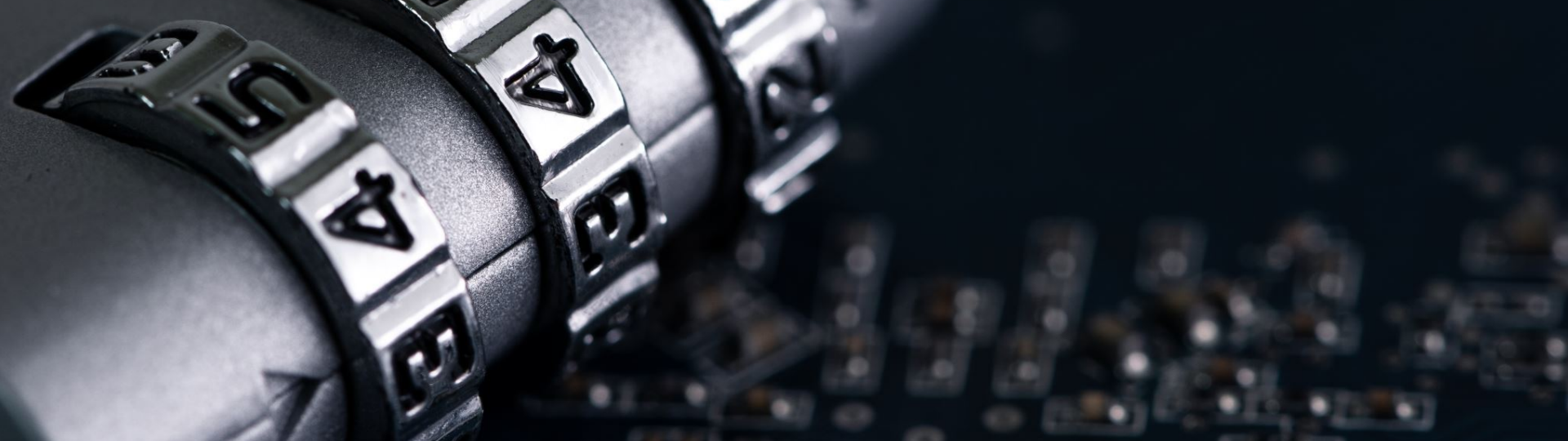




**PDF Days
Europe
2025**

Document Security & Authenticity

Practical Implementations & Common Pitfalls of working with Digital Signatures

Harshil Parikh | Eugene Bochilo | Yulian Gaponenko
Apryse

What is on the Agenda?

- **Introduction & Theme**

- SDKs follow specifications, but validation still fails.
- Why this issue persists in real-life cases and how we fixed it?

- **Real-World Support Cases**

- Case 1 : PDF Viewers reject incremental timestamp update to Signatures
- Case 2 : Fields locking can break trust in unexpected ways
- Case 3 : Are there "allowed changes" to signed PDF documents?

- **Common threads across cases**

- PDF processors behaviour vs specifications conformance

- **Key takeaways & conclusions**

- Practical lessons for SDK Developers
- Why specification conformance is not always enough

Case One

The Timestamp that broke a valid Signature : A DocMDP Dilemma



Brief Context

A PDF signed with **DocMDP Level 1** (no changes allowed) had valid field locking that was later removed, causing malformed dictionary. A compliant timestamp was added via incremental update but viewers flagged it as invalid, despite meeting PAdES standards.

- DocMDP level 1 forbids changes, but a valid timestamp was added.
- Field locking was altered, resulting in malformed dictionary
- Viewers flagged the file as invalid despite being PAdES compliant.

Broken Document | Causes

- /Fields key was added then removed -> Malformed lock dictionary
- Viewers saw this as a DocMDP Violation (spec says its allowed)
- Timestamp update modified /StructTreeRoot
- Viewers flagged it as a structural change
- Compliant file shown as invalid by viewer logic.

Specification Behavior vs Viewer Behavior

- ISO 32000-1 (PDF 1.7) : DocMDP Level 1 forbids any changes

“No changes to the document shall be permitted” (Sec 12.8.2.2)

- ETSI TS 102 778-4 (PAdES Part 4) :

“DocMDP Restrictions... shall not apply to incremental updates that contain a DSS dictionary” (Annex A.1 V1.1.2 p13)

Table 254 – Entries in the DocMDP transform parameters dictionary

Key	Type	Value
Type	name	(Optional) The type of PDF object that this dictionary describes; if present, shall be TransformParams for a transform parameters dictionary.
P	number	(Optional) The access permissions granted for this document. Valid values shall be: 1 No changes to the document shall be permitted; any change to the document shall invalidate the signature. 2 Permitted changes shall be filling in forms, instantiating page templates, and signing; other changes shall invalidate the signature. 3 Permitted changes shall be the same as for 2, as well as annotation creation, deletion, and modification; other changes shall invalidate the signature. Default value: 2.
V	name	(Optional) The DocMDP transform parameters dictionary version. The only valid value shall be 1.2 . NOTE this value is a name object, not a number. Default value: 1.2 .

DocMDP restrictions (see ISO 32000-1 [1] clause 12.8.2.2) shall not apply to incremental updates to a PDF document containing a DSS dictionary and associated VRI, Certs, CRLs and OCSPs.

NOTE: *ISO 32000-1 [1], 12.8.2.2, discusses the **DocMDP** (Modification, Detection and Prevention) feature whereby a set of permissions can be associated with a PDF in conjunction with a certification signature. The permissions of **DocMDP** are present in the **P** key of the **DocMDP** transform parameters dictionary, as an integer in the range 1 through 3. Values of 2 and 3 allow for additional signatures to be included after the certification but a value of 1 does not allow any change so allow Document Time-stamps. This provision will need to be changed from that in ISO 32000-1 [1], to allow for the inclusion of LTV, including DSS and Document Time-stamps.*

How was it Fixed?

- Suppressed Structural updates during timestamping (e.g. No changes to / StructTreeRoot)
- Adjusted annotation tagging for invisible timestamp signatures to avoid unintended visibility issues in Signature Panel of viewers
- Ensured timestamp updates remained within incremental update scope.
- Fixed malformed field lock dictionary by retaining the required /Fields key
- Avoided breaking Document locking level 1 policy while still supporting PAdES-LTA.

When Viewers Dictate the Rules : The SDK Developer's Dilemma

- Must Validate not just against the SPEC, but how major viewers actually enforce them
- SDKs must build guard rails to suppress changes that could trigger viewer-side invalidation (e.g. /StructTreeRoot)
- Developers must balance spec-compliance with real world viewer quirks
- SDK teams must educate users when compliant files are falsely marked invalid
- Often forced to reverse-engineer undocumented blackbox viewer behaviour to stay compatible

Case One Takeaways : Spec isn't always enough

- Even when all specs are followed, viewer behaviour can still invalidate a signature
- The Fix was not about correcting an error, It was made to ensure compatibility for viewers.
- Developers must be ready to adapt to quirks beyond the standard recorded in the PDF Spec.
- Clear need for SDKs to offer practical guardrails against invisible breaking changes

Case Two

When locking breaks trust : A FieldMDP Timing Ambiguity



Brief Context

This case centers around ambiguity in the interpretation of the **FieldMDP** dictionary when locking is applied to a digital signature. The issue emerged for client when :

- A PDF was signed with locking set to ALL
- A second signature was added with locking set to NONE
- Result : The first signature became invalid, even though no fields were changed, just the locking flags.

What the specification states

- **PDF 1.7, Section 12.8.2.4 (FieldMDP)** defines restrictions on form field changes after signing, but lacks clarity.
- “The FieldMDP transform method.. Describes which form fields do not permit changes after the signature is applied but it is vague
 - “Any modifications to specified form fields shall invalidate the recipient's signature.”
- **PDF spec (FieldMDP) is vague** about what is the exact expected scope of such locks in case of "All"

12.8.2.4 FieldMDP

The **FieldMDP** transform method shall be used to detect changes to the values of a list of form fields. The entries in its transform parameters dictionary are listed in Table 256.

Table 256 – Entries in the FieldMDP transform parameters dictionary

Key	Type	Value
Type	name	(Optional) The type of PDF object that this dictionary describes; if present, shall be TransformParams for a transform parameters dictionary.
Action	name	(Required) A name that, along with the Fields array, describes which form fields do not permit changes after the signature is applied. Valid values shall be: All All form fields. Include Only those form fields that specified in Fields . Exclude Only those form fields not specified in Fields .
Fields	array	(Required if Action is <i>Include</i> or <i>Exclude</i>) An array of text strings containing field names.

Table 256 – Entries in the FieldMDP transform parameters dictionary (continued)

Key	Type	Value
V	name	(Optional: PDF 1.5 required) The transform parameters dictionary version. The value for PDF 1.5 and later shall be 1.2. NOTE This value is a name object, not a number. Default value: 1.2

On behalf of a document author creating a document containing both form fields and signatures the following shall be supported by conforming writers:

- The author specifies that form fields shall be filled in without invalidating the approval or certification signature. The **P** entry of the **DocMDP** transform parameters dictionary shall be set to either 2 or 3 (see Table 254).
- The author can also specify that after a specific recipient has signed the document, any modifications to specific form fields shall invalidate that recipient's signature. There shall be a separate signature field for each designated recipient, each having an associated signature field lock dictionary (see Table 233) specifying the form fields that shall be locked for that user.
- When the recipient signs the field, the signature, signature reference, and transform parameters dictionaries shall be created. The **Action** and **Fields** entries in the transform parameters dictionary shall be copied from the corresponding fields in the signature field lock dictionary.

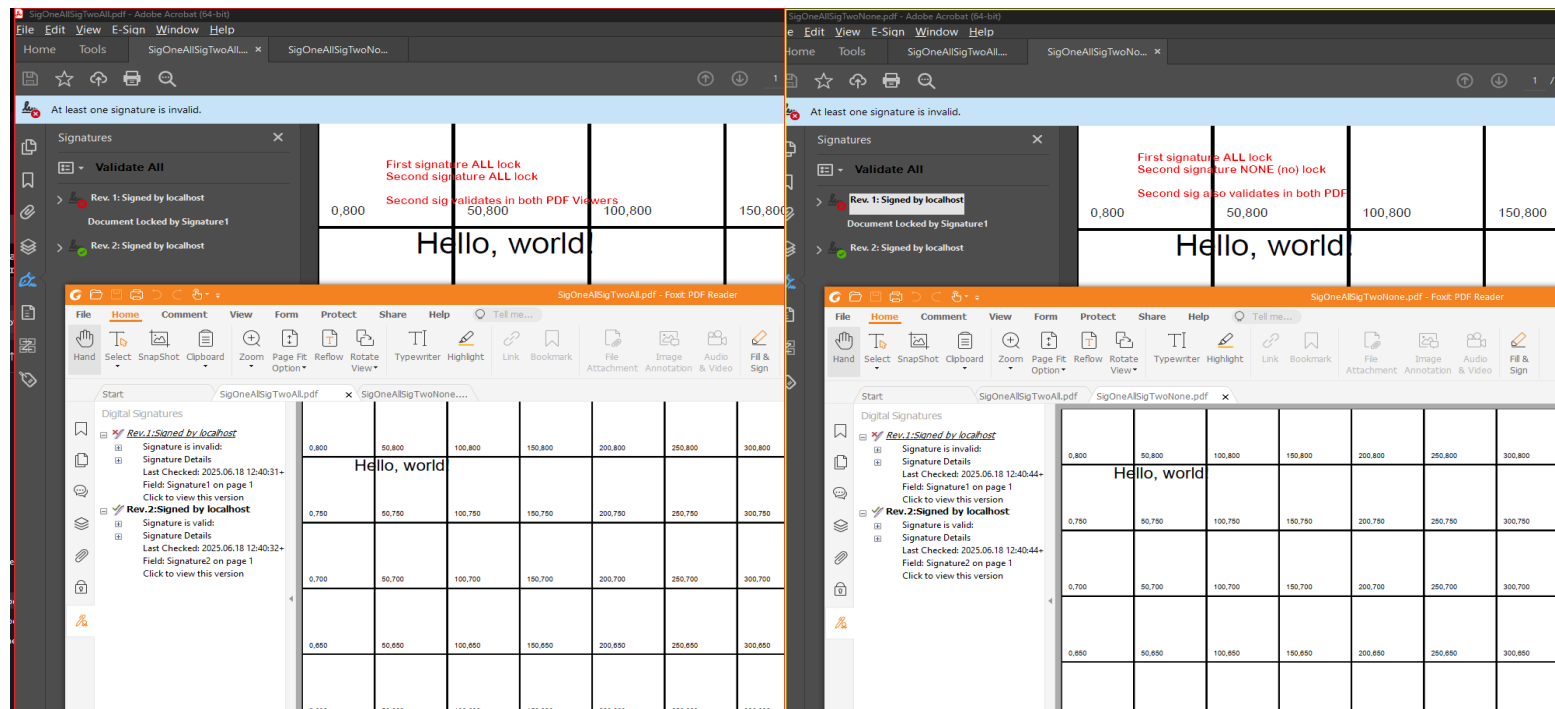
NOTE This copying is done because all objects in a signature dictionary must be direct objects if the dictionary contains a byte range signature. Therefore, the transform parameters dictionary cannot reference the signature field lock dictionary indirectly.

FieldMDP signatures shall be validated in a similar manner to **DocMDP** signatures. See Validating Signatures That Use the DocMDP Transform Method in 12.8.2.2, “DocMDP” for details.

What Happened? Field Locking Ambiguity in SigFieldLock Usage

- PDF Allows setting /FieldMDP to lock form fields on the signature
- Client Scenario :
 - Subsequent signatures signing
 - 1st Signature locks ALL,
 - 2nd subsequently added signature has NONE
- **Specification interpretation questions :**
 - Is FieldMDP scoped only to the signature that includes it?
 - Should it affect subsequent modifications to the fields list?

What do the viewers say?



How do signature validators interpret locking?

- Signature Structure is technically valid, passes format and cryptographic checks
- Signature warning due to **FieldMDP** constraint violation, even with no existing field edits.
- There is a consensus: Lock ALL is treated as affecting previously signed fields, even without edits

SIGNATURE (id = SIGNATURE_localhost_20250617-1055_2)

Format Checking : PASSED

Does the signature format correspond to an expected format?
Is the signature identification not ambiguous?
Is the signed references identification not ambiguous?
Is only one SignerInfo present?
Is the /ByteRange dictionary consistent?
Does the /ByteRange not overlap with other signature/timestamp?
Is the signature dictionary consistent?
Do signed and final revisions contain equal amount of pages?
Is no element overlapping detected in the PDF?
Is there no visual difference between signed and final revisions in the PDF?
Does the document contain none of the undefined object modifications?

Identification of the Signing Certificate : PASSED

Is there an identified candidate for the signing certificate?

Validation Context Initialization : PASSED

Is the signature policy known?

X509 Certificate Validation : NO_CERTIFICATE_CHAIN_FOUND

Can the certificate chain be built till a trust anchor?

Cryptographic Verification : PASSED

Has the reference data object been found?

Is the reference data object intact?

Is the signature intact?

Signature Acceptance Validation : PASSED

Is the structure of the signature valid?

Is the signed attribute: 'signing-certificate' present?

Is the signed qualifying property: 'signing-time' present?

Is the signed qualifying property: 'message-digest' or 'SignedProperties' present?

Are cryptographic constraints met for the signature creation?

Are cryptographic constraints met for the message digest?



SIGNATURE (id = SIGNATURE_localhost_20250617-1055)

Format Checking : PASSED

Does the signature format correspond to an expected format?
Is the signature identification not ambiguous?
Is the signed references identification not ambiguous?
Is only one SignerInfo present?
Is the /ByteRange dictionary consistent?
Does the /ByteRange not overlap with other signature/timestamp?
Is the signature dictionary consistent?
Do signed and final revisions contain equal amount of pages?
Is no element overlapping detected in the PDF?
Is there no visual difference between signed and final revisions in the PDF?
Is signature valid against /FieldMDP dictionary?
Is signature valid against /SigFieldLock dictionary?
Does the document contain none of the undefined object modifications?

Identification of the Signing Certificate : PASSED

Is there an identified candidate for the signing certificate?

Validation Context Initialization : PASSED

Is the signature policy known?

X509 Certificate Validation : NO_CERTIFICATE_CHAIN_FOUND

Can the certificate chain be built till a trust anchor?

Cryptographic Verification : PASSED

Has the reference data object been found?

Is the reference data object intact?

Is the signature intact?

Signature Acceptance Validation : PASSED

Is the structure of the signature valid?

Is the signed attribute: 'signing-certificate' present?

Is the signed qualifying property: 'signing-time' present?

Is the signed qualifying property: 'message-digest' or 'SignedProperties' present?

Are cryptographic constraints met for the signature creation?

Are cryptographic constraints met for the message digest?



Case Two : Takeaways

- Changing only the signature field locking attribute caused unexpected invalidation of earlier signatures
- Solutions developers must align their workflows both with PDF Spec constraints to form modifications and common validators.
- SDK Developers must also handle such ambiguities defensively to ensure cross-viewer validation.
- Adding validation checks in SDKs to enforce FieldMDP locking behavior, consistently across viewer interpretations.

Case Three

Strictly Compliant – Still Misaligned



Brief Context

Some tools confirm the signed document as valid and unmodified, while others flag changes outside the signature scope.

- How can the allowed changes (DSS/timestamps) lead to inconsistent validation results?
 - Either risking false negatives even for compliant PDFs
 - Or misrepresenting the relation between a signature and the document

Signature Validation mismatch

- Customer implemented SDK-based validation logic to verify PAdES-B-LT signed documents

Viewers Verdict	 Document has not been modified after signing
Online Validator Verdict	 PAdES-B-LT Valid, full integrity confirmed
User Implementation Verdict	 Signature integrity and authenticity confirmed.  Signature does not cover whole document.

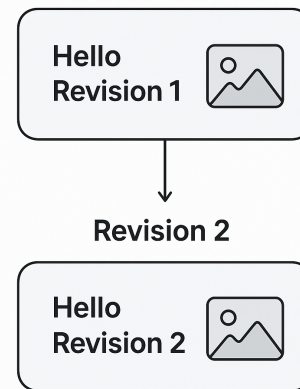
Different tools. Same PDF. Opposite Verdicts.

The Core Issue

```
102 0000000000000000 n
103 00000000573 00000 n
104 00000000785 00000 n
105 00000000828 00000 n
106 00000000189 00000 n
107 00000000059 00000 n
108 00000000015 00000 n
109 00000000475 00000 n
110 00000000399 00000 n
111 00000000307 00000 n
112 00000000335 00000 n
113 00000000363 00000 n
114 trailer
115 <<
116 /Size 15
117 /Root 3 0 R
118 /Info 11 0 R
119 /ID [<399f6c259d5697401bf4c9ff8ecd2cbb> <399f6c259d5697401bf4c9ff8ecd2cbb>]
120 >>
121 startxref
122 1064
123 %%EOF
124
125 3 0 obj
126 <<
127 /Type /Catalog
128 /Pages 1 0 R
129 /Names 2 0 R
130 /ViewerPreferences 5 0 R
131 /AcroForm <<
132 /Fields [15 0 R]
133 /SigFlags 3
```

New revision begins after EOF

PDF File Internal Structure



SDK-based Validation Flags the Signature

- SDK's Signature Integrity and Authenticity checker returns **true**
- Chain of Trust checker returns **true**
- SDK's check to see if signature protects entire PDF document returns **false**
 - Document had two revisions, post-signature content found
 - New revisions are beyond original signature's ByteRange

There are two defined techniques for computing a digital signature of the contents of a PDF file:

- A *byte range digest* shall be computed over a range of bytes in the PDF file, that shall be indicated by the **ByteRange** entry in the signature dictionary. This range should be the entire PDF file, including the signature dictionary but excluding the signature value itself (the **Contents** entry). In case of multiple digital signatures this range shall be the sequence of bytes starting from the "%PDF-" comment at the beginning of the PDF document to the end of the "%EOF" comment, possibly followed by an optional EOL marker, terminating the incremental update that adds the digital signature dictionary to the document. When a byte range digest is present, all values in the signature dictionary shall be direct objects.
- Additionally, modification detection may be specified by a *signature reference dictionary*. The **TransformMethod** entry shall specify the general method for modification detection, and the **TransformParams** entry shall specify the variable portions of the method.

What actually happened in the file?

- Technically : this is not covered by the original signature
 - The verdict the customer got is not wrong.
 - There were modifications and new revisions cannot be blindly trusted based on the given signature field validation only.
- Functionally : it's still valid PAdES-B-LT extension
- Original PDF Document [1.3]
- After signing :
 - PDF Upgraded to version 1.7
 - DSS dictionary added
 - Timestamp dictionary added

What's the “Allowed Changes”?

- ETSI EN 319 142-2 and ISO 32000-2 permits additions like timestamps, CRLs, DSS metadata

DocMDP restrictions (see ISO 32000-1 [1], clause 12.8.2.2) shall not apply to incremental updates to a PDF document containing a **DSS** dictionary and associated **VRI**, **Certs**, **CRLs** and **OCSPs**.

NOTE: ISO 32000-1 [1], clause 12.8.2.2, addresses the DocMDP (Modification, Detection and Prevention) feature whereby a set of permissions can be associated with a PDF in conjunction with a certification signature. The permissions of DocMDP are present in the entry with the **P** key of the DocMDP transform parameters dictionary, as an integer in the range 1 through 3. Values of 2 and 3 allow for additional signatures to be included after the certification but a value of 1 does not allow any change but allows Document Time-stamps.

- PDF processors developers are left to interpret the spec on their own:
 - What is the exact set of allowed modifications to the document structure that are needed to introduce those entities.
 - What is allowed when DocMDP is not explicitly defined (i.e. for approval signatures).
- Viewers and validator tools verdict is not wrong: such modifications are indeed allowed

Case Three : Takeaways

- All the tools confirm one thing: the signature is cryptographically valid, however that was not enough for document validation
- High-level validators prioritize clarity and usability.
- There are "allowed changes", but in that case you have to trust the PDF processor judgement and interpretation that nothing else has been affected in the document.

In conclusion

- We hope these several cases to be a practical lesson for PDF processors developers.
- We wanted to highlight possible misalignment in interpretations of the validation procedure.
- What can be improved?
 - Introduce a corpus of example documents to establish consensus on which document updates are valid.
 - Define common guidelines for interpreting digital signatures validity in presence of allowed updates.
 - Educate end-users and the technical community.

Thank you!

Harshil Parikh - Technical Support Engineer - harshil.parikh@apryse.com

Eugene Bochilo - Senior Software Developer - eugene.bochilo@apryse.com

Yulian Gaponenko - Engineering Manager - yulian.gaponenko@apryse.com

