



PDF Days Europe 2022 | Berlin

Ideas for interoperable self-updating PDF documents

Matthias Valvekens

Plan of the talk



- Why would we even want this?
- What should we watch out for?
- Relevant current features of the specification
- Concrete implementation ideas

Motivation: the whys-and-why-nots

PDFs are typically static



- This is a good thing 95% of the time!
- But: web pages can be kept up to date centrally.

PDFs are typically static



- This is a good thing 95% of the time!
- But: web pages can be kept up to date centrally.
- At times, “pushing” updates to PDFs sounds awfully convenient:
 - Distributing new versions of publicly available documents
 - Correcting a mistake in that email attachment you just sent out
 - Sharing form inputs, digital signatures, etc.
 - ...

Question: wait, isn't that a terribly stupid idea?



- Wouldn't that be at odds with PDF's status as a "format of record"?
- What about security? Surely that's a problem!
- Wouldn't updates be an interoperability nightmare?
 - How to distribute updates interoperably?
 - How to make sure they are *applied* uniformly?
- All of this sounds terribly complicated for little gain.

Answer: maybe...but it's not totally ridiculous



- All of these concerns are valid.
- Most of them can be addressed fairly comprehensively.
 - Most security questions have fairly uncomplicated solutions.
 - For most workflows, you don't need much more than a basic file server.

Architectural sketches

The basic ideas



- Updatable PDFs self-declare with an AutoUpdate dictionary
 - Repository URL
 - Addressing mode (content hash or document ID)
 - Update type (full replacement, incremental, XFDF, merge operation, ...)
 - Integrity settings
- The repo server does not need to understand PDF

Abstract workflow



Given: updatable PDF

- Generate update URL from contents of AutoUpdate dict
- HTTP GET to fetch update data
- Verify integrity of the update using declared info in AutoUpdate dict
- Apply the update

Security issues



- 1 Could auto-updates serve as a malware vector?
- 2 Could the auto-update mechanism itself be exploited?
- 3 How to defend against unauthorised updates?

Security issues



- 1 Could auto-updates serve as a malware vector?
 - **Absolutely!**
 - ...but so can regular PDFs; similar hardening principles apply.
- 2 Could the auto-update mechanism itself be exploited?
- 3 How to defend against unauthorised updates?

Security issues



- 1 Could auto-updates serve as a malware vector?
 - **Absolutely!**
 - ...but so can regular PDFs; similar hardening principles apply.
- 2 Could the auto-update mechanism itself be exploited?
 - **Potentially, yes!**
 - e.g. abusing it for tracking data, as a DoS vector...
 - These are also largely implementation issues.
- 3 How to defend against unauthorised updates?

Security issues



- 1 Could auto-updates serve as a malware vector?
 - **Absolutely!**
 - ...but so can regular PDFs; similar hardening principles apply.
- 2 Could the auto-update mechanism itself be exploited?
 - **Potentially, yes!**
 - e.g. abusing it for tracking data, as a DoS vector...
 - These are also largely implementation issues.
- 3 How to defend against unauthorised updates?
 - Let's zoom in on that one...

Simplified security model for update integrity



- The updates can only be “as trusted as” the document to which they apply
 - If the base document is compromised, it’s game over
 - Might as well take the base document at face value

Simplified security model for update integrity



- The updates can only be “as trusted as” the document to which they apply
 - If the base document is compromised, it’s game over
 - Might as well take the base document at face value
- The update server is not necessarily a trusted intermediary
 - This also applies to confidentiality.
 - We focus on authenticity for now.

Simplified security model for update integrity



- The updates can only be “as trusted as” the document to which they apply
 - If the base document is compromised, it’s game over
 - Might as well take the base document at face value
- The update server is not necessarily a trusted intermediary
 - This also applies to confidentiality.
 - We focus on authenticity for now.
- We only need the update to be authenticated until it is applied

Simplified security model for update integrity



- The updates can only be “as trusted as” the document to which they apply
 - If the base document is compromised, it's game over
 - Might as well take the base document at face value
- The update server is not necessarily a trusted intermediary
 - This also applies to confidentiality.
 - We focus on authenticity for now.
- We only need the update to be authenticated until it is applied
- Minimise reliance on external parties
 - Avoid heavy PKI requirements
 - No interactive communication between author and recipient

Update integrity: central question



In summary:

Question

Given an update payload, can we ensure that it was

- *intended to be applied to our base document, and*
- *authored (or at least approved) by the original author of that base document?*

Update integrity: contrasts



There are some interesting contrasts with e.g. document signing:

Update integrity: contrasts



There are some interesting contrasts with e.g. document signing:

- We only care about whether the update is *authorised*

Update integrity: contrasts



There are some interesting contrasts with e.g. document signing:

- We only care about whether the update is *authorised*
- The *identity* of the author doesn't really matter

Update integrity: contrasts



There are some interesting contrasts with e.g. document signing:

- We only care about whether the update is *authorised*
- The *identity* of the author doesn't really matter
- We have a natural “ground truth”: the base document

Update integrity: contrasts



There are some interesting contrasts with e.g. document signing:

- We only care about whether the update is *authorised*
- The *identity* of the author doesn't really matter
- We have a natural “ground truth”: the base document
- Integrity check is *relative to* that base document

Update integrity: enforcement mechanism example



- Work with pre-shared keys
 - Public workflows: public key is part of the update dictionary in the base document
 - Private/closed workflows: use shared secret (e.g. derived from file encryption key, or separate)

Update integrity: enforcement mechanism example



- Work with pre-shared keys
 - Public workflows: public key is part of the update dictionary in the base document
 - Private/closed workflows: use shared secret (e.g. derived from file encryption key, or separate)
- Update payload is protected by a token based on that pre-shared key
 - Can be either a signature or a MAC
 - Several standards to pick from: PASETO, good old CMS, ...
 - Token binds to payload and (indirectly) to base document

Update integrity: enforcement mechanism example



- Work with pre-shared keys
 - Public workflows: public key is part of the update dictionary in the base document
 - Private/closed workflows: use shared secret (e.g. derived from file encryption key, or separate)
- Update payload is protected by a token based on that pre-shared key
 - Can be either a signature or a MAC
 - Several standards to pick from: PASETO, good old CMS, ...
 - Token binds to payload and (indirectly) to base document
- Token can be precomputed, or generated by the server on-demand

What's an "update", even?



Two broad categories:

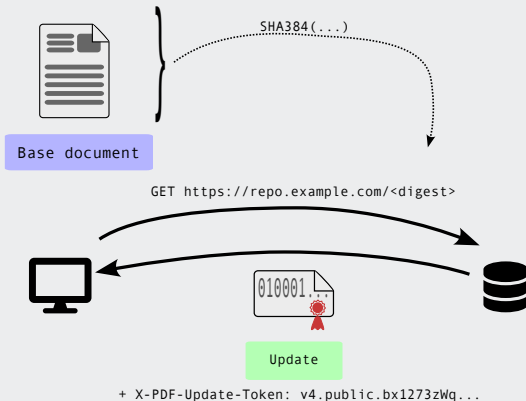
- updates that can be applied completely deterministically;
- updates where there is some degree of implementation-dependence in the output.

Types of update: application fully deterministic

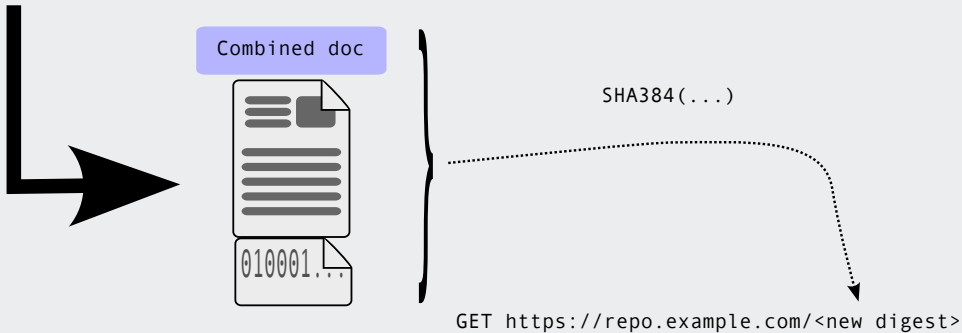


- Full replacement
 - Incremental update (can simply be concatenated)
- ~> We can work with hash-based addressing in these cases

Example: requesting an incremental update



Example: requesting an incremental update



Types of update: result not fully defined



- Page-level modifications (adding/removing/replacing pages)
 - Annotation/form updates through XFDF
- ~> Hash-based addressing is not stable!

Types of update: result not fully defined



- Page-level modifications (adding/removing/replacing pages)
- Annotation/form updates through XFDF

~> Hash-based addressing is not stable!

- Not byte-for-byte deterministic, but still plausibly interoperable
- Addressing based on document ID seems workable
- Need a procedure to update the second part of the ID

Update types: summary



- Not all kinds of update can be served using the exact same process
- Which type of update is preferable would be workflow-dependent
- Regardless of the chosen update type, there are still many limitations

Examples and demonstration

Example workflows



- Simple, one-way: updating documents disseminated to the general public.
 - Would use a token based on asymmetric crypto, with pre-shared keys
 - The server can act passively, and doesn't need to do anything other than serving updates and tokens

Example workflows



- Simple, one-way: updating documents disseminated to the general public.
 - Would use a token based on asymmetric crypto, with pre-shared keys
 - The server can act passively, and doesn't need to do anything other than serving updates and tokens
- More complex, collaborative: coordinate multi-signer workflows and (pseudo-)push signatures revision by revision
 - Would most likely rely on a shared secret known to authorised participants
 - The server needs to support more sophisticated upload capabilities

Demo time!

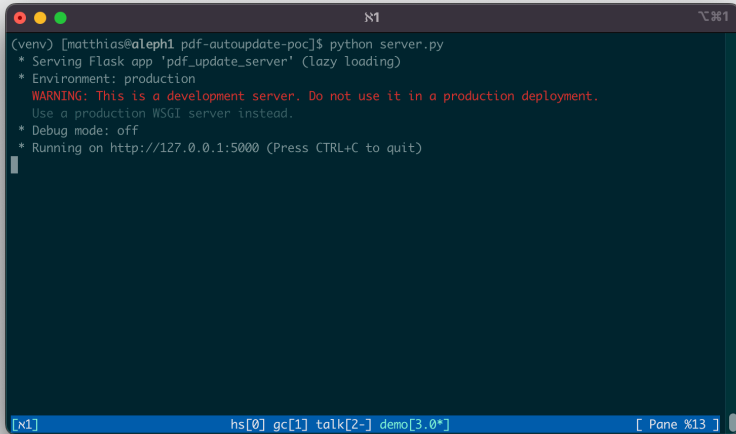
Matthias Valvekens *

```
def generate_token(upd_body: bytes, resource_id: str, upd_type: UpdateType,
                  key: ed25519.Ed25519PrivateKey) -> bytes:
    payload_obj = {
        'protocolVersion': 'v1',
        'resourceId': resource_id,
        'updateLength': len(upd_body),
        'updateType': str(upd_type.value)[1:]
    }
    payload = json.dumps(payload_obj).encode('utf8')
    key_bytes = key.private_bytes(
        Encoding.PEM, PrivateFormat.PKCS8, encryption_algorithm=NoEncryption())
    pk = Key.new(version=4, purpose="public", key=key_bytes)
    return pyseto.encode(pk, payload, implicit_assertion=upd_body)
```

24

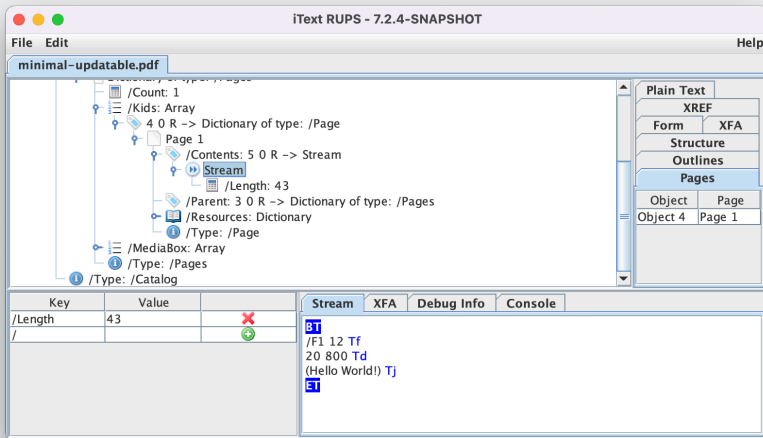
Matthias Valvekens

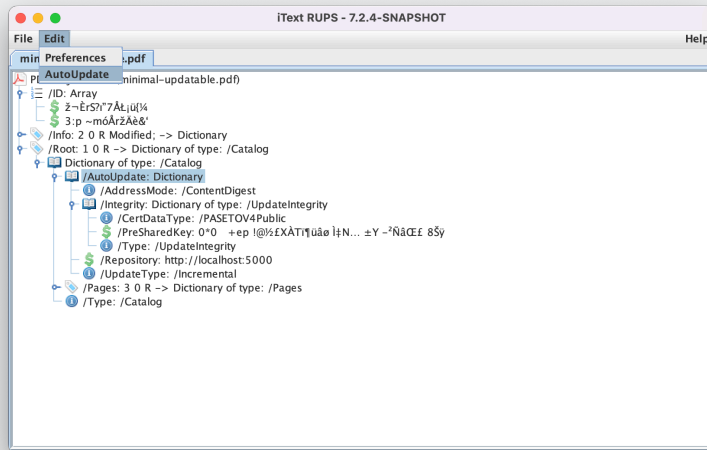
```
25 def send_update(source_dir, fname_base) -> Response:
26
27     try:
28         token_path = safe_join(
29             os.fspath(source_dir), os.fspath(fname_base) + ".token"
30         )
31         with open(token_path, 'r') as tok_in:
32             token_data = tok_in.read().strip()
33     except FileNotFoundError:
34         return abort(404)
35
36     response = send_from_directory(source_dir, fname_base + ".dat")
37     response.headers['X-PDF-Update-Token'] = token_data
38     return response
```

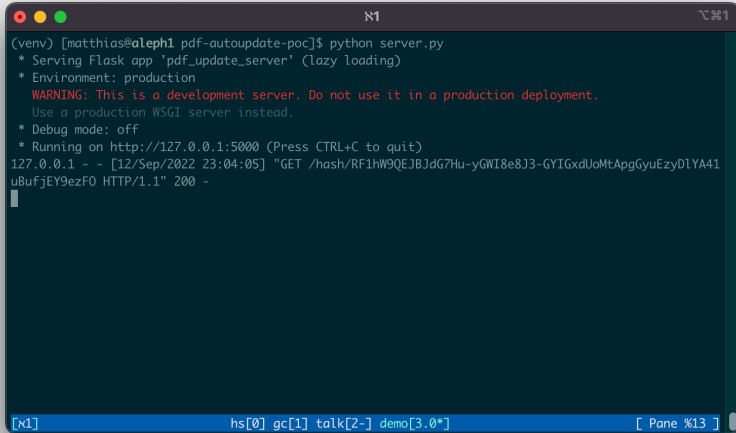

A terminal window with a dark blue background and white text. The window has a title bar with three colored circles (red, yellow, green) on the left and a close button on the right. The text inside the terminal shows the output of running a Flask application. The status bar at the bottom is blue and contains window management icons and labels.

```
(venv) [matthias@aleph1 pdf-autoupdate-poc]$ python server.py
* Serving Flask app 'pdf_update_server' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

[N1] hs[0] gc[1] talk[2-] demo[3.0*] [Pane %13]





A terminal window with a dark blue background and white text. The window title bar shows three colored circles (red, yellow, green) on the left, the text 'N1' in the center, and a magnifying glass icon on the right. The terminal content shows the execution of a Python script, outputting server status and a log entry. The status bar at the bottom is blue with white text showing file names and pane information.

```
(venv) [matthias@aleph1 pdf-autoupdate-poc]$ python server.py
* Serving Flask app 'pdf_update_server' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
127.0.0.1 - - [12/Sep/2022 23:04:05] "GET /hash/RF1hW9QEJBjdG7Hu-yGWI8e8J3-GYIGxdUoMtApgGyuEzyDlYA41
uBufjEY9ezF0 HTTP/1.1" 200 -
```

[N1] hs[0] gc[1] talk[2-] demo[3.0*] [Pane %13]

