PDF to John

```
#!/usr/bin/env python3

# This software is Copyright (c) 2023 Benjamin Dornel <benjamindornel@gmail.com>
# and it is hereby released to the general public under the following terms:
# Redistribution and use in source and binary forms, with or without
# modification, are permitted.



import argparse
import logging

try:
    from pyhanko.pdf_utils.misc import PdfReadError
    from pyhanko.pdf_utils.reader import PdfFileReader
except ImportError:
    print("pyhanko is missing, run 'pip install --user pyhanko==0.20.1' to install it!")
    exit(1)

logger = logging.getLogger(__name__)



class SecurityRevision:
    """Represents Standard Security Handler Revisions
    and the corresponding key length for the /O and /U entries

    In Revision 5, the /O and /U entries were extended to 48 bytes,
    with three logical parts -- a 32 byte verification hash,
    an 8 byte validation salt, and an 8 byte key salt."""

    revisions = {
        2: 32,  # RC4_BASIC
        3: 32,  # RC4_EXTENDED
        4: 32,  # RC4_OR_AES128
        5: 48,  # AES_R5_256
        6: 48,  # AES_256
    }
```

```
    @classmethod
    def get_key_length(cls, revision):
        """
        Get the key length for a given revision,
        defaults to 48 if no revision is specified.
        """
        return cls.revisions.get(revision, 48)


class PdfHashExtractor:
    """
    Extracts hash and encryption information from a PDF file

    Attributes:
    - `file_name`: PDF file path.
    - `strict`: Boolean that controls whether an error is raised, if a PDF
        has problems e.g. Multiple definitions in encryption dictionary
        for a specific key. Defaults to `False`.
    - `algorithm`: Encryption algorithm used by the standard security handler
    - `length`: The length of the encryption key, in bits. Defaults to 40.
    - `permissions`: User access permissions
    - `revision`: Revision of the standard security handler
    """

    def __init__(self, file_name: str, strict: bool = False):
        self.file_name = file_name

        with open(file_name, "rb") as doc:
            self.pdf = PdfFileReader(doc, strict=strict)
            self.encrypt_dict = self.pdf._get_encryption_params()

            if not self.encrypt_dict:
                raise RuntimeError("File not encrypted")

            self.algorithm: int = self.encrypt_dict.get("/V")
            self.length: int = self.encrypt_dict.get("/Length", 40)
            self.permissions: int = self.encrypt_dict["/P"]
            self.revision: int = self.encrypt_dict["/R"]

    @property
```

```
def document_id(self) -> bytes:
    return self.pdf.document_id[0]

@property
def encrypt_metadata(self) -> str:
    """
    Get a string representation of whether metadata is encrypted.

    Returns "1" if metadata is encrypted, "0" otherwise.
    """
    return str(int(self.pdf.security_handler.encrypt_metadata))

def parse(self) -> str:
    """
    Parse PDF encryption information into a formatted string for John
    """
    passwords = self.get_passwords()
    fields = [
        f"$pdf${self.algorithm}",
        self.revision,
        self.length,
        self.permissions,
        self.encrypt_metadata,
        len(self.document_id),
        self.document_id.hex(),
        passwords,
    ]
    return "*".join(map(str, fields))

def get_passwords(self) -> str:
    """
    Creates a string consisting of the hexidecimal string of the
    /U, /O, /UE and /OE entries and their corresponding byte string length
    """
    passwords = []
    keys = ("udata", "odata", "oeseed", "ueseed")
    max_key_length = SecurityRevision.get_key_length(self.revision)

    for key in keys:
        if data := getattr(self.pdf.security_handler, key):
            data: bytes = data[:max_key_length]
```

```
                passwords.extend([str(len(data)), data.hex()])

        return "*".join(passwords)


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="PDF Hash Extractor")
    parser.add_argument(
        "pdf_files", nargs="+", help="PDF file(s) to extract information from"
    )
    parser.add_argument(
        "-d", "--debug", action="store_true", help="Print the encryption dictionary"
    )
    args = parser.parse_args()

    for filename in args.pdf_files:
        try:
            extractor = PdfHashExtractor(filename)
            pdf_hash = extractor.parse()
            print(pdf_hash)

            if args.debug:
                if extractor.encrypt_dict:
                    print("Encryption Dictionary:")
                    for key, value in extractor.encrypt_dict.items():
                        print(f"{key}: {value}")
                else:
                    print("No encryption dictionary found in the PDF.")

        except PdfReadError as error:
            logger.error("%s : %s", filename, error, exc_info=True)
```

Revision #1
Created 2025-11-25 18:07:10 UTC by David Rizzo
Updated 2025-11-25 18:07:24 UTC by David Rizzo