**Tool for Emulating the PAdES Qualified Electronic Signature**

**Security of Computer Systems**

**Project Report**

Authors:

Bartłomiej, Krawisz, 193319

Stanisław, Nieradko, 193044

Version: 1.0

**Versions**

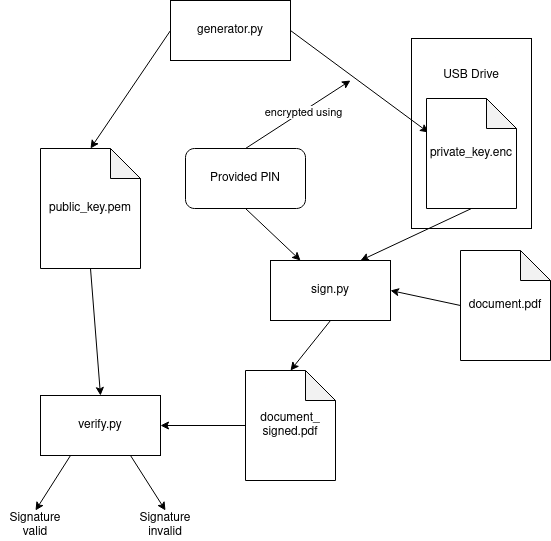
|  |  |  |
| --- | --- | --- |
| Version | Date | Description of changes |
| 1.0 | 26.03.2025 | Creation of the document |

1. **Project**
   1. ***Project’s description***

The project’s goal was to create tool to sign PDF documents to detect any attempts in changing its content. The project was supposed to fully emulate the process, including the hardware toll needed for person identification utilising USB drive for that purpose.

* 1. ***Results***

The project was completed using python in a form of three separate applications: generate.py, sign.py and verify.py. Each application is responsible for different part of the project’s requirements, utilizing common modules for configuration, cryptographic operations and integrations with operating system. The application’s is based on the concept diagram below:



*Fig. 1 – Application’s concept diagram.*

Application is created on and intended to run on Linux based operating systems, due to the way in which it looks up any new USB drivers connected to the system:

|  |
| --- |
| 1. ## @package utils.usb\_utils 2. # USB drive detection utility functions 3. # 4. # This module provides utility functions to detect USB drives with specific files. 5. import getpass 6. import os 7. import threading 8. import time 9. from typing import Callable 10. ## Detect a USB drive with a specific file 11. # @param filename Name of the file to detect 12. # @return Path of the USB drive if found, otherwise None 13. def detect\_usb\_drive\_with\_file(filename: str): 14. base\_paths = ["/media/", f"/run/media/{getpass.getuser()}/"] 15. for base\_path in base\_paths: 16. if os.path.exists(base\_path): 17. # List directories in the base path 18. for subdir in os.listdir(base\_path): 19. file\_path = os.path.join(base\_path, subdir, filename) 20. if os.path.exists(file\_path): 21. return file\_path 22. return None 23. ## USB drive detection daemon 24. # @param callback Callback function to call when USB drive is detected or removed 25. # @param filename Name of the file to detect 26. # @param interval Interval in seconds to check for USB drive 27. # @return Thread object of the daemon, call start() to start the daemon 28. def usb\_detection\_daemon(callback: Callable[[str|None], None], filename: str, interval: float = 1): 29. def daemon(): 30. previous\_usb\_path = None 31. callback(None) # initial call 32. while True: 33. usb\_path = detect\_usb\_drive\_with\_file(filename) 34. if usb\_path == previous\_usb\_path: 35. time.sleep(interval) 36. continue 37. previous\_usb\_path = usb\_path 38. if usb\_path: 39. callback(usb\_path) 40. else: 41. callback(None) 42. time.sleep(interval) 44. t = threading.Thread(target=daemon) 45. t.daemon = True 46. return t |

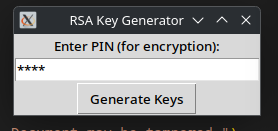
*List. 1 – usb\_utils.py – module responsible for looking up usb drives with encrypted private key*

Other parts of the application are designed to work independently of the underlying OS (logic is written in Python and GUI is created in tkinter).

Application was created by both creators over VSCode’s LiveShare and later published to Bartłomiej Krawisz’s GitHub account.

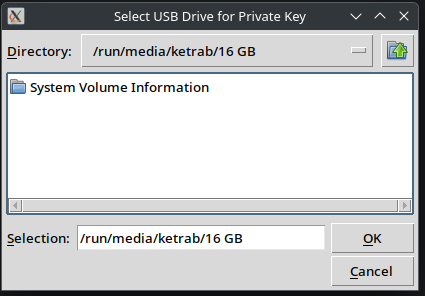
* 1. ***Application’s Usage Flow***

To begin using / testing the application run generator.py application and insert the USB drive that will be used to store application’s secret. The application should display prompt for entering PIN for the secret:



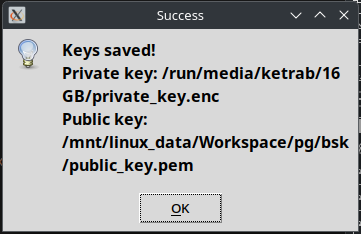
*Fig. 2 – generator.py PIN prompt window*

After providing the PIN, click Generate Keys button and select the USB drive in a file manager window:



*Fig. 3 – File manager window used to select USB drive*

When OK button is clicked the application saves the secret to selected location and displays confirmation window with both public and private keys locations. (Public key is stored in the same directory in which program was run from).



*Fig. 4 – Confirmation window*

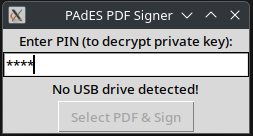
A screenshot of a computer

AI-generated content may be incorrect.

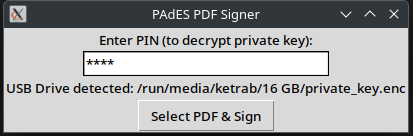
*Fig. 5 – USB drive’s contents after running generate.py*

After preparing the USB drive you can eject it for further usage.

To sign the document, you can use sign.py. When run, the application will display window prompting for PIN and will display whether USB drive is detected or not.

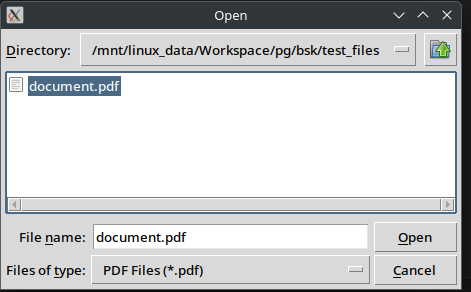


*Fig. 6 – Window without any USB drive detected*

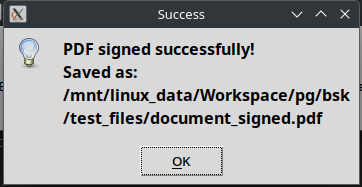


*Fig. 7 – Window with USB drive with secret detected*

When USB drive is detected, and correct PIN is provided the user can click the button below and select the PDF document to sign. When correct PDF document is selected and button Open is clicked the application will display the message window informing us about the successful PDF document signing and the location of signed PDF (should be next to original PDF file with \_signed suffix).

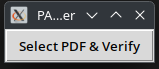


*Fig. 8 – File Manager’s window used to select the PDF document to sign*

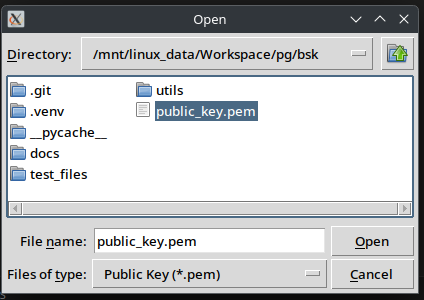


*Fig. 9 – Success message after successfully signing the PDF document*

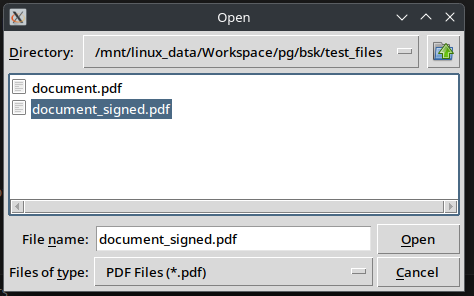
To test whether the signed PDF document has been tampered with, run verify.py, select public key alongside PDF to check and message will appear with requested information.



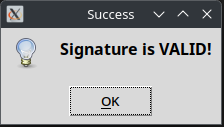
*Fig. 10 – verify.py window*



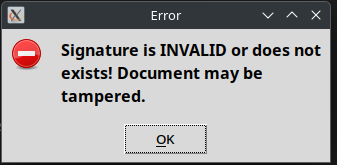
*Fig. 11 – File Manager’s window with public key selection*



*Fig. 12 – File Manager’s window with PDF document selection*



*Fig. 13 – Success window informing us of temperance-free PDF document*



*Fig. 14 – Failure window informing us about PDF document temperance*

* 1. ***Important application’s parts***

To sign the PDF documents, application:

* Generates RSA keys pair
* Encrypts the private key using provided PIN (it will be required to sign the documents)
* When signing the document: the PDF’s content hash is created, encrypted using private key and stored at the end of the signed document

To verify the PDF document, application:

* Reads the public key and uses it to decrypt the stored hash in the PDF document
* Hashes the PDF document without the signature and compares them. When difference is detected it means that the document has been tampered with.

|  |
| --- |
| 1. ## @package utils.crypt\_utils 2. # Cryptography utility functions 3. # 4. # This module provides utility functions for RSA key generation and encryption/decryption of private keys. 5. # 6. # It uses the PyCryptodome library for cryptographic operations. 7. from Crypto.PublicKey import RSA 8. from Crypto.Cipher import AES 9. from Crypto.Hash import SHA256 10. ## Length of the RSA key 11. RSA\_KEY\_LENGTH = 4096 12. ## Generate RSA key pair 13. # @return Tuple of private key and public key in bytes 14. def generate\_rsa\_keys(): 15. key = RSA.generate(RSA\_KEY\_LENGTH) 16. private\_key = key.export\_key() 17. public\_key = key.publickey().export\_key() 18. return private\_key, public\_key 19. ## Encrypt the private key using a PIN 20. # @param private\_key Private key in bytes 21. # @param pin PIN to encrypt the private key 22. # @return Encrypted private key in bytes 23. def encrypt\_private\_key(private\_key: bytes, pin: str): 24. hash\_obj = SHA256.new(pin.encode('utf-8')) 25. aes\_key = hash\_obj.digest() 27. cipher = AES.new(aes\_key, AES.MODE\_EAX) 28. ciphertext, tag = cipher.encrypt\_and\_digest(private\_key) 30. return cipher.nonce + tag + ciphertext 31. ## Decrypt the private key using a PIN 32. # @param encrypted\_key Encrypted private key in bytes 33. # @param pin PIN to decrypt the private key 34. # @return Decrypted private key as RSA key object 35. def decrypt\_private\_key(encrypted\_key: bytes, pin: str): 36. hash\_obj = SHA256.new(pin.encode('utf-8')) 37. aes\_key = hash\_obj.digest() 39. nonce, tag, ciphertext = encrypted\_key[:16], encrypted\_key[16:32], encrypted\_key[32:] 40. cipher = AES.new(aes\_key, AES.MODE\_EAX, nonce=nonce) 42. try: 43. private\_key = cipher.decrypt\_and\_verify(ciphertext, tag) 44. return RSA.import\_key(private\_key) 45. except ValueError: 46. return None |

*List. 2 – crypt\_utils.py – module responsible for all cryptographic operations*

|  |
| --- |
| 1. ## @package utils.pdf\_signing\_utils 2. # PDF signing utility functions 3. # 4. # This module provides utility functions for signing and verifying PDF files using RSA digital signatures. 5. from Crypto.PublicKey import RSA 6. import hashlib 7. ## Length of the RSA signature in bytes 8. SIGNATURE\_LENGTH = 512 9. ## Length of the hash in bytes 10. HASH\_LENGTH = 256 11. ## Sign a PDF file using an RSA private key 12. # @param pdf\_path Path of the PDF file to sign 13. # @param private\_key RSA private key object 14. # @param signed\_pdf\_path Path to save the signed PDF file 15. # @return Path of the signed PDF file 16. def sign\_pdf(pdf\_path: str, private\_key: RSA.RsaKey, signed\_pdf\_path: str | None = None): 17. if signed\_pdf\_path is None: 18. signed\_pdf\_path = pdf\_path.replace('.pdf', '\_signed.pdf') 19. with open(pdf\_path, 'rb') as f: 20. pdf\_data = f.read() 21. pdf\_hash = hashlib.sha256(pdf\_data).digest() 22. signature = pow(int.from\_bytes(pdf\_hash, byteorder='big'), private\_key.d, private\_key.n) 23. signature\_bytes = signature.to\_bytes(SIGNATURE\_LENGTH, byteorder='big') 24. with open(signed\_pdf\_path, 'wb') as f: 25. f.write(pdf\_data + signature\_bytes) 26. return signed\_pdf\_path 27. ## Verify the signature of a signed PDF file using an RSA public key 28. # @param pdf\_path Path of the signed PDF file 29. # @param public\_key RSA public key object 30. # @return True if the signature is valid, False otherwise 31. def verify\_signature(pdf\_path: str, public\_key: RSA.RsaKey): 32. with open(pdf\_path, 'rb') as f: 33. content = f.read() 34. pdf\_data, signature = content[:-SIGNATURE\_LENGTH], content[-SIGNATURE\_LENGTH:] 35. pdf\_hash = hashlib.sha256(pdf\_data).digest() 36. decrypted\_hash = pow(int.from\_bytes(signature, byteorder='big'), public\_key.e, public\_key.n) 37. try: 38. decrypted\_hash\_bytes = decrypted\_hash.to\_bytes(HASH\_LENGTH//8, byteorder='big') 39. except OverflowError: 40. return False 41. return pdf\_hash == decrypted\_hash\_bytes |

*List. 3 – pdf\_signing\_utils.py – module responsible for signing PDF files*

Applications frontend is written using tkinter. As an example, the generate.py is attached below:

|  |
| --- |
| 1. ## @package generate 2. # RSA key generation user GUI application 3. # 4. # This script generates RSA key pair and saves the private key on a USB drive and public key on the local machine. 5. import os 6. import tkinter as tk 7. from tkinter import filedialog, messagebox 8. from consts import PRIVATE\_KEY\_FILENAME, PUBLIC\_KEY\_FILENAME 9. from utils.crypt\_utils import generate\_rsa\_keys, encrypt\_private\_key 10. ## Main application class for RSA key generation 11. class GenerateApp: 12. ## Constructor 13. def \_\_init\_\_(self): 14. # GUI setup 15. self.\_\_root = tk.Tk() 16. self.\_\_root.title("RSA Key Generator") 17. tk.Label(self.\_\_root, text="Enter PIN (for encryption):").pack() 18. self.\_\_pin\_entry = tk.Entry(self.\_\_root, show="\*", width=30) 19. self.\_\_pin\_entry.pack() 20. self.\_\_generate\_button = tk.Button(self.\_\_root, text="Generate Keys", command=self.\_\_save\_keys) 21. self.\_\_generate\_button.pack() 22. ## Start the application 23. def start(self): 24. self.\_\_root.mainloop() 25. def \_\_save\_keys(self): 26. pin = self.\_\_pin\_entry.get() 27. if not pin: 28. messagebox.showerror("Error", "PIN cannot be empty!") 29. return 31. usb\_path = filedialog.askdirectory(title="Select USB Drive for Private Key") 32. if not usb\_path: 33. return 34. private\_key, public\_key = generate\_rsa\_keys() # Generate keys 35. encrypted\_private\_key = encrypt\_private\_key(private\_key, pin) 37. # Save encrypted private key on USB drive 38. private\_key\_path = os.path.join(usb\_path, PRIVATE\_KEY\_FILENAME) 39. with open(private\_key\_path, "wb") as f: 40. f.write(encrypted\_private\_key) 41. # Save public key on local machine 42. public\_key\_path = os.path.join(os.getcwd(), PUBLIC\_KEY\_FILENAME) 43. with open(public\_key\_path, "wb") as f: 44. f.write(public\_key) 46. messagebox.showinfo("Success", f"Keys saved!\nPrivate key: {private\_key\_path}\nPublic key: {public\_key\_path}") 47. # Clear the PIN entry 48. self.\_\_pin\_entry.delete(0, tk.END) 49. if \_\_name\_\_ == "\_\_main\_\_": 50. app = GenerateApp() 51. app.start() |

*List. 4 – generate.py – application’s frontend*

* 1. ***Summary***

Application has been successfully presented and verified by the teacher. Application’s source code is available on GitHub <https://github.com/ketrab2003/pades-emulation> alongside the documentation generated using Doxygen <https://ketrab2003.github.io/pades-emulation/>.

1. **Bibliography**

* <https://en.wikipedia.org/wiki/PAdES>
* Project’s Instruction