Security of Computer Systems

Project Report

Authors:

Iwo Czartowski, 193066 Kacper Witczak, 193373 _____

Versions

Version	Date	Description of changes
1.0	02.06.2025	Initial full project submission

Project – Control Term

Description

At the control term, the project focused on creating a GUI application (gui_app.py) for RSA key generation. The app allows the user to generate RSA private and public keys, which are then saved locally. This stage served as the foundation for cryptographic operations later implemented.

Content

- Implementation of the KeyGenerator class for RSA key pair creation.
- Development of a simple GUI interface enabling key generation and saving.

Results

The application successfully generates RSA keys and saves the private and public keys in PEM format to the local disk. The keys are compatible with cryptography libraries and ready for further use.

Summary

The control term proved the feasibility of generating and saving RSA key pairs. The GUI enabled user-friendly interaction with cryptographic key management, laying groundwork for further security functions.

Project – Final Term

Description

The final version extends the control term application by adding secure storage options, PDF signing, and signature verification functionalities. Key management was enhanced to allow saving keys to USB drives. Additionally, the project implements PDF file hashing, digital signing, and signature validation to ensure document authenticity and integrity.

Code Description

A list of implemented modules and their key features:

- 1. KeyGenerator:
 - Generates RSA private and public keys.
 - Exposes keys in PEM format.
- 2. KeyStorage:
 - Saves private keys encrypted with salt and IV.
 - Supports saving to disk or to USB drives detected by USBDriveDetector.
 - Reads keys from disk or USB.
- 3. USBDriveDetector:
 - Detects connected USB drives on Windows systems.
 - Lists available USB drives and their volume names.
 - Finds private key files (.pem) on connected USB drives.
- 4. PdfSigner:
 - Calculates SHA-256 hash of PDF content.
 - Signs the PDF hash with a private RSA key.
 - Appends the signature to the PDF file.
- 5. PdfVerifier:
 - Extracts the appended signature from the PDF.
 - Calculates the SHA-256 hash excluding the signature.
 - Verifies the signature with the RSA public key.

List. 1 – Code listing [2].

Results

 The application now supports key storage both on local disk and USB drives with automated USB detection.

- PDF documents can be digitally signed by hashing the content and applying the RSA private key signature.
- Signed PDFs are appended with the signature to allow later verification.
- Verification module confirms the authenticity of the signed PDFs by verifying the signature with the corresponding public key.

Summary

The project demonstrates a complete workflow for asymmetric cryptographic key management, secure key storage, and PDF document signing/verification. It can be used as a base for secure document management and authentication systems. The integration of USB drive detection improves usability by facilitating portable key storage.

Literature

[1] Article.

[2] Online Doxygen documentation, https://www.doxygen.nl/manual/lists.html, (accessed on 01.02.2025).

PDF signing and verifying app

Generated by Doxygen 1.14.0

1 Namespace Index	1
1.1 Namespace List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 gui_app Namespace Reference	7
4.1.1 Function Documentation	7
4.1.1.1 main()	7
4.2 key_decryption Namespace Reference	7
4.3 key_encryption Namespace Reference	8
4.4 key_generator Namespace Reference	8
4.5 key_storage Namespace Reference	8
4.6 pdf_gui_app Namespace Reference	8
4.6.1 Variable Documentation	8
4.6.1.1 app	8
4.6.1.2 root	8
4.7 pdf_signer Namespace Reference	9
4.8 pdf_verifier Namespace Reference	
4.9 usb_detector Namespace Reference	
5 Class Documentation	11
5.1 key_decryption.KeyDecryptor Class Reference	11
5.1.1 Detailed Description	
5.1.2 Constructor & Destructor Documentation	11
5.1.2.1init()	
5.1.3 Member Function Documentation	
5.1.3.1 decrypt_private_key()	12
5.1.3.2 derive_key()	12
5.1.4 Member Data Documentation	13
5.1.4.1 encrypted_data	13
5.1.4.2 pin	13
5.2 key_encryption.KeyEncryptor Class Reference	13
5.2.1 Detailed Description	
5.2.2 Constructor & Destructor Documentation	
5.2.2.1init()	14
5.2.3 Member Function Documentation	
5.2.3.1 derive_key()	
5.2.3.2 encrypt_private_key()	
5.2.4 Member Data Documentation	

5.2.4.1 iv	15
5.2.4.2 pin	15
5.2.4.3 private_key	15
5.2.4.4 salt	16
5.3 gui_app.KeyGenerationApp Class Reference	16
5.3.1 Detailed Description	16
5.3.2 Constructor & Destructor Documentation	17
5.3.2.1init()	17
5.3.3 Member Function Documentation	18
5.3.3.1 check_usb_auto()	18
5.3.3.2 generate_keys()	18
5.3.3.3 start_auto_detect()	19
5.3.4 Member Data Documentation	19
5.3.4.1 check_usb_auto	19
5.3.4.2 generate_button	19
5.3.4.3 key_frame	20
5.3.4.4 last_usb_state	20
5.3.4.5 pin_entry	20
5.3.4.6 pin_frame	20
5.3.4.7 pin_label	20
5.3.4.8 public_key_text	20
5.3.4.9 root	20
5.3.4.10 status_frame	21
5.3.4.11 status_label	21
5.3.4.12 status_value	21
5.3.4.13 usb_detector	21
5.3.4.14 usb_frame	21
5.3.4.15 usb_status_label	21
5.3.4.16 usb_status_value	21
5.4 key_generator.KeyGenerator Class Reference	22
5.4.1 Detailed Description	22
5.4.2 Constructor & Destructor Documentation	22
5.4.2.1init()	22
5.4.3 Member Function Documentation	23
5.4.3.1 generate_keys()	23
5.4.3.2 get_public_key_pem()	23
5.4.4 Member Data Documentation	23
5.4.4.1 key_size	23
5.4.4.2 private_key	23
5.4.4.3 public_key	24
5.5 key_storage.KeyStorage Class Reference	24
5.5.1 Detailed Description	24

5.5.2 Member Function Documentation	. 24
5.5.2.1 read_key_from_disk()	. 24
5.5.2.2 save_key()	. 25
5.5.2.3 save_key_to_usb()	. 25
5.5.2.4 save_public_key()	. 26
5.6 pdf_signer.PdfSigner Class Reference	. 26
5.6.1 Detailed Description	. 26
5.6.2 Constructor & Destructor Documentation	. 27
5.6.2.1init()	. 27
5.6.3 Member Function Documentation	. 27
5.6.3.1 _calculate_hash()	. 27
5.6.3.2 sign_pdf()	. 27
5.6.4 Member Data Documentation	. 28
5.6.4.1 pdf_path	. 28
5.7 pdf_gui_app.PdfSignerCheckerApp Class Reference	. 28
5.7.1 Detailed Description	. 29
5.7.2 Constructor & Destructor Documentation	. 30
5.7.2.1init()	. 30
5.7.3 Member Function Documentation	. 31
5.7.3.1 browse_file()	. 31
5.7.3.2 browse_private_key()	. 32
5.7.3.3 browse_public_key()	. 32
5.7.3.4 check_usb_auto()	. 32
5.7.3.5 sign_pdf()	. 33
5.7.3.6 start_usb_detection()	. 33
5.7.3.7 toggle_key_source()	. 33
5.7.3.8 verify_pdf()	. 33
5.7.4 Member Data Documentation	. 34
5.7.4.1 browse_button	. 34
5.7.4.2 browse_private_button	. 34
5.7.4.3 browse_public_button	. 34
5.7.4.4 check_usb_auto	. 34
5.7.4.5 file_frame	. 34
5.7.4.6 file_label	. 34
5.7.4.7 file_path_entry	. 34
5.7.4.8 file_path_var	. 35
5.7.4.9 last_usb_state	. 35
5.7.4.10 op_frame	. 35
5.7.4.11 pin_entry	. 35
5.7.4.12 pin_label	. 35
5.7.4.13 private_key_entry	. 35
5.7.4.14 private_key_frame	. 35

5./.4.15 private_key_label	 . 36
5.7.4.16 private_key_path_var	 . 36
5.7.4.17 public_key_entry	 . 36
5.7.4.18 public_key_frame	 . 36
5.7.4.19 public_key_label	 . 36
5.7.4.20 public_key_path_var	 . 36
5.7.4.21 results_frame	 . 36
5.7.4.22 results_text	 . 37
5.7.4.23 root	 . 37
5.7.4.24 sign_button	 . 37
5.7.4.25 status_frame	 . 37
5.7.4.26 status_label	 . 37
5.7.4.27 status_value	 . 37
5.7.4.28 usb_detector	 . 37
5.7.4.29 usb_frame	 . 38
5.7.4.30 usb_status_label	 . 38
5.7.4.31 usb_status_value	 . 38
5.7.4.32 use_usb_key_check	 . 38
5.7.4.33 use_usb_key_var	 . 38
5.7.4.34 verify_button	 . 38
5.8 pdf_verifier.PdfVerifier Class Reference	 . 39
5.8.1 Detailed Description	 . 39
5.8.2 Constructor & Destructor Documentation	 . 39
5.8.2.1init()	 . 39
5.8.3 Member Function Documentation	 . 40
5.8.3.1 _calculate_hash()	 . 40
5.8.3.2 extract_signature()	 . 40
5.8.3.3 verify_signature()	 . 40
5.8.4 Member Data Documentation	 . 41
5.8.4.1 pdf_path	 . 41
5.9 usb_detector.USBDriveDetector Class Reference	 . 41
5.9.1 Detailed Description	 . 42
5.9.2 Constructor & Destructor Documentation	 . 42
5.9.2.1init()	 . 42
5.9.3 Member Function Documentation	 . 42
5.9.3.1 get_drive_path()	 . 42
5.9.3.2 get_private_key_path()	 . 43
5.9.3.3 is_drive_connected()	 . 43
5.9.3.4 list_available_usb_drives()	 . 44
5.9.4 Member Data Documentation	 . 44
5.9.4.1 connected drive	. 44

6 File Documentation	45
6.1 crypto/key_decryption.py File Reference	45
6.1.1 Detailed Description	45
6.2 key_decryption.py	46
6.3 crypto/key_encryption.py File Reference	46
6.3.1 Detailed Description	47
6.4 key_encryption.py	47
6.5 crypto/key_generator.py File Reference	48
6.5.1 Detailed Description	48
6.6 key_generator.py	48
6.7 crypto/key_storage.py File Reference	49
6.7.1 Detailed Description	49
6.8 key_storage.py	50
6.9 crypto/pdf_signer.py File Reference	51
6.9.1 Detailed Description	51
6.10 pdf_signer.py	51
6.11 crypto/pdf_verifier.py File Reference	52
6.11.1 Detailed Description	52
6.12 pdf_verifier.py	52
6.13 gui_app.py File Reference	53
6.14 gui_app.py	54
6.15 pdf_gui_app.py File Reference	56
6.15.1 Detailed Description	56
6.16 pdf_gui_app.py	57
6.17 usb/usb_detector.py File Reference	59
6.18 usb_detector.py	59
Index	61

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

gui_app									 															7
key_decryptio	n								 															7
key_encryptio	n								 															8
key_generator	r.								 															8
key_storage									 															8
pdf_gui_app									 															8
pdf_signer									 															9
pdf_verifier									 															9
usb detector									 															9

2 Namespace Index

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ey_decryption.KeyDecryptor	11
ey_encryption.KeyEncryptor	13
ui_app.KeyGenerationApp	16
ey_generator.KeyGenerator	22
ey_storage.KeyStorage	24
df_signer.PdfSigner	26
df_gui_app.PdfSignerCheckerApp	28
df_verifier.PdfVerifier	39
sb detector.USBDriveDetector	41

4 Class Index

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

gui_app.py	53
pdf_gui_app.py	
GUI application for signing and verifying PDF documents	56
crypto/key_decryption.py	
Decrypts an encrypted private RSA key using a PIN and AES-CBC	45
crypto/key_encryption.py	
Encrypts an RSA private key using a PIN-derived AES key	46
crypto/key_generator.py	
Module for generating RSA key pairs using the cryptography library	48
crypto/key_storage.py	
Module providing static methods for storing and reading cryptographic keys, including support	
for USB drives detection and file operations	49
crypto/pdf_signer.py	
Provides functionality for hashing and signing PDF files using RSA private keys	51
crypto/pdf_verifier.py	
Provides functionality to verify digital signatures appended to PDF files	52
usb/usb detector.py	59

6 File Index

Chapter 4

Namespace Documentation

4.1 gui app Namespace Reference

Classes

class KeyGenerationApp

Functions

• main ()

4.1.1 Function Documentation

4.1.1.1 main()

```
gui_app.main ()
Entry point for the application.
Initializes Tkinter root and launches the key generation GUI.
```

Definition at line 175 of file gui_app.py.

```
00175 def main():
00176     """
00177     Entry point for the application.
00178     Initializes Tkinter root and launches the key generation GUI.
00179     """
00180     root = tk.Tk()
00181     app = KeyGenerationApp(root)
00182     root.mainloop()
00183
00184
```

4.2 key_decryption Namespace Reference

Classes

class KeyDecryptor

4.3 key_encryption Namespace Reference

Classes

class KeyEncryptor

4.4 key_generator Namespace Reference

Classes

class KeyGenerator

4.5 key storage Namespace Reference

Classes

class KeyStorage

4.6 pdf_gui_app Namespace Reference

Classes

• class PdfSignerCheckerApp

Variables

- **root** = tk.Tk()
- app = PdfSignerCheckerApp(root)

4.6.1 Variable Documentation

4.6.1.1 app

```
pdf_gui_app.app = PdfSignerCheckerApp(root)
```

Definition at line 199 of file pdf_gui_app.py.

4.6.1.2 root

```
pdf_gui_app.root = tk.Tk()
```

Definition at line 198 of file pdf_gui_app.py.

4.7 pdf_signer Namespace Reference

Classes

• class PdfSigner

4.8 pdf_verifier Namespace Reference

Classes

• class PdfVerifier

4.9 usb_detector Namespace Reference

Classes

• class USBDriveDetector

Chapter 5

Class Documentation

5.1 key_decryption.KeyDecryptor Class Reference

Public Member Functions

- __init__ (self, encrypted_data, pin)
- derive_key (self, salt)
- decrypt_private_key (self)

Public Attributes

- encrypted_data = encrypted_data
- pin = pin.encode()

5.1.1 Detailed Description

```
Responsible for decrypting a private key encrypted with AES, using a password-based key derived from a PIN.
```

Definition at line 19 of file key_decryption.py.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 __init__()

Definition at line 24 of file key_decryption.py.

5.1.3 Member Function Documentation

5.1.3.1 decrypt private key()

Definition at line 49 of file key decryption.py.

```
def decrypt_private_key(self):
00051
               Decrypts the encrypted private key using AES-CBC.
00052
               :return: A deserialized RSA private key object.
:raises ValueError: If decryption or deserialization fails.
00053
00054
00055
00056
               salt = self.encrypted_data[:SALT_LENGTH]
00057
               iv = self.encrypted_data[SALT_LENGTH:SALT_LENGTH + IV_LENGTH]
00058
               encrypted_private_pem = self.encrypted_data[SALT_LENGTH + IV_LENGTH:]
00059
00060
               key = self.derive_key(salt)
               cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
00061
00062
               decryptor = cipher.decryptor()
00063
00064
               decrypted_pem_padded = decryptor.update(encrypted_private_pem) + decryptor.finalize()
00065
00066
               pad_len = decrypted_pem_padded[-1]
               decrypted_pem = decrypted_pem_padded[:-pad_len]
00067
00068
00069
               private_key = serialization.load_pem_private_key(decrypted_pem, password=None)
00070
               return private_key
```

5.1.3.2 derive key()

Definition at line 34 of file key_decryption.py.

```
00034
          def derive_key(self, salt):
00035
00036
              Derives a symmetric key using PBKDF2.
00037
00038
              :param salt: The salt used in key derivation.
00039
              :return: Derived key (bytes).
00040
00041
              kdf = PBKDF2HMAC(
00042
                 algorithm=hashes.SHA256(),
00043
                  length=KEY_LENGTH,
00044
                  salt=salt,
00045
                  iterations=PBKDF2_ITERATIONS,
00046
00047
              return kdf.derive(self.pin)
00048
```

5.1.4 Member Data Documentation

5.1.4.1 encrypted_data

key_decryption.KeyDecryptor.encrypted_data = encrypted_data

Definition at line 31 of file key_decryption.py.

5.1.4.2 pin

```
key_decryption.KeyDecryptor.pin = pin.encode()
```

Definition at line 32 of file key_decryption.py.

The documentation for this class was generated from the following file:

· crypto/key_decryption.py

5.2 key_encryption.KeyEncryptor Class Reference

Public Member Functions

- __init__ (self, private_key, pin)
- derive_key (self)
- encrypt_private_key (self)

Public Attributes

- private_key = private_key
- pin = pin.encode()
- salt = os.urandom(SALT_LENGTH)
- iv = os.urandom(IV_LENGTH)

5.2.1 Detailed Description

Class for encrypting an RSA private key using AES-CBC with a derived key from a PIN.

Definition at line 20 of file key_encryption.py.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 init ()

Definition at line 24 of file key_encryption.py.

```
00024
            def __init__(self, private_key, pin):
00025
00026
                 Constructor.
00027
00028
                 :param private_key: RSA private key object to be encrypted.
                 :param pin: The user-provided PIN used to derive the AES key.
00029
00030
                self.private_key = private_key
self.pin = pin.encode()
self.salt = os.urandom(SALT_LENGTH)
00031
00032
00033
00034
                 self.iv = os.urandom(IV_LENGTH)
00035
```

5.2.3 Member Function Documentation

5.2.3.1 derive_key()

Definition at line 36 of file key_encryption.py.

```
00036
          def derive_key(self):
00037
00038
              Derives a symmetric AES key using PBKDF2 and the PIN.
00039
              :return: Derived AES key (bytes).
00040
00041
00042
              kdf = PBKDF2HMAC(
00043
                  algorithm=hashes.SHA256(),
00044
                  length=KEY_LENGTH,
00045
                  salt=self.salt,
                  iterations=PBKDF2_ITERATIONS,
00046
00047
00048
              return kdf.derive(self.pin)
00049
```

5.2.3.2 encrypt_private_key()

```
\verb"key_encryption.KeyEncryptor.encrypt_private_key" (
                 self)
Encrypts the private key with AES-CBC.
:return: A tuple containing (salt, IV, encrypted PEM bytes).
Definition at line 50 of file key_encryption.py.
          def encrypt_private_key(self):
00050
00051
00052
               Encrypts the private key with AES-CBC.
00053
               :return: A tuple containing (salt, IV, encrypted PEM bytes). """
00054
00055
               key = self.derive_key()
00056
               \verb|cipher| = \verb|Cipher| (\verb|algorithms.AES| (\verb|key|)|, \verb|modes.CBC| (\verb|self.iv|)|)
00057
00058
              encryptor = cipher.encryptor()
00059
00060
              private_pem = self.private_key.private_bytes(
                   encoding=serialization.Encoding.PEM,
00061
00062
                   format=serialization.PrivateFormat.PKCS8,
00063
                   \verb"encryption_algorithm" = \verb"serialization.NoEncryption" ()
00064
              )
00065
00066
               pad_len = 16 - (len(private_pem) % 16)
00067
               private_pem_padded = private_pem + bytes([pad_len] * pad_len)
00068
```

encrypted_private_pem = encryptor.update(private_pem_padded) + encryptor.finalize()

5.2.4 Member Data Documentation

5.2.4.1 iv

00069

00070 00071

```
key_encryption.KeyEncryptor.iv = os.urandom(IV_LENGTH)
```

return self.salt, self.iv, encrypted_private_pem

Definition at line 34 of file key_encryption.py.

5.2.4.2 pin

```
key_encryption.KeyEncryptor.pin = pin.encode()
```

Definition at line 32 of file key_encryption.py.

5.2.4.3 private key

```
key_encryption.KeyEncryptor.private_key = private_key
```

Definition at line 31 of file key_encryption.py.

5.2.4.4 salt

```
key_encryption.KeyEncryptor.salt = os.urandom(SALT_LENGTH)
```

Definition at line 33 of file key encryption.py.

The documentation for this class was generated from the following file:

· crypto/key_encryption.py

5.3 gui_app.KeyGenerationApp Class Reference

Public Member Functions

- __init__ (self, root)
- start_auto_detect (self)
- · check usb auto (self)
- generate_keys (self)

Public Attributes

- root = root
- usb_frame = tk.Frame(root, padx=10, pady=10)
- usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
- usb_status_value = tk.Label(self.usb_frame, text="Nowy...")
- pin frame = tk.Frame(root, padx=10, pady=10)
- pin label = tk.Label(self.pin frame, text="Wpisz PIN:")
- pin_entry = tk.Entry(self.pin_frame, width=20)
- generate_button
- status_frame = tk.Frame(root, padx=10, pady=5)
- status_label = tk.Label(self.status_frame, text="Status:")
- status value = tk.Label(self.status frame, text="Gotowe")
- key_frame = tk.LabelFrame(root, text="Wygenerowany klucz publiczny", padx=10, pady=10)
- public_key_text = scrolledtext.ScrolledText(self.key_frame, wrap=tk.WORD)
- usb_detector = USBDriveDetector()
- bool last_usb_state = None
- · check_usb_auto

5.3.1 Detailed Description

 ${\tt GUI}$ application for RSA key pair generation and storage on USB drive.

The user inputs a PIN to encrypt the generated private key before saving it to USB. The public key is displayed and saved locally. The application detects USB connection status to allow or block key generation accordingly.

Definition at line 25 of file gui app.py.

5.3.2 Constructor & Destructor Documentation

```
5.3.2.1 init ()
gui_app.KeyGenerationApp.__init__ (
                self,
                root)
Initializes the GUI layout, USB detection, and event bindings.
:param root: Tk root window.
Definition at line 34 of file gui_app.py.
         def __init__(self, root):
00036
              Initializes the GUI layout, USB detection, and event bindings.
00037
              :param root: Tk root window.
00038
00039
00040
              self.root = root
00041
              root.title("Generator Kluczy")
00042
              root.geometry("600x500")
00043
              root.resizable(False, False)
00044
              # USB
00045
00046
              self.usb_frame = tk.Frame(root, padx=10, pady=10)
00047
              self.usb frame.pack(fill=tk.X)
00048
00049
              self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
```

self.usb_status_label.pack(side=tk.LEFT)

self.pin_frame.pack(fill=tk.X)

self.pin_label.pack(side=tk.LEFT)

self.usb_status_value.pack(side=tk.LEFT, padx=10)

self.pin_frame = tk.Frame(root, padx=10, pady=10)

self.pin_entry = tk.Entry(self.pin_frame, width=20)

self.pin_entry.pack(side=tk.LEFT, padx=10)

self.usb_status_value = tk.Label(self.usb_frame, text="Nowy...")

self.pin_label = tk.Label(self.pin_frame, text="Wpisz PIN:")

00050

00051

00052 00053

00054 00055 00056

00057

00058 00059

00060

00061 00062

00063

5.3.3 Member Function Documentation

5.3.3.1 check_usb_auto()

```
gui_app.KeyGenerationApp.check_usb_auto ( self) Checks the USB drive connection status every second. Updates the USB status label and enables/disables the generate button.
```

Definition at line 105 of file gui_app.py.

```
def check_usb_auto(self):
00106
00107
              Checks the USB drive connection status every second.
              Updates the USB status label and enables/disables the generate button.
00108
00109
00110
              current_state = self.usb_detector.is_drive_connected()
00111
00112
              if current_state != self.last_usb_state:
00113
                  if current_state:
00114
                      self.usb_status_value.config(text="Podłączony", fg="green")
00115
                      self.generate_button.config(state=tk.NORMAL)
00116
                  else:
00117
                      self.usb_status_value.config(text="Nie podłączony", fg="red")
00118
                      self.generate_button.config(state=tk.DISABLED)
00119
00120
              self.last_usb_state = current_state
00121
00122
              self.root.after(1000, self.check usb auto)
00123
```

5.3.3.2 generate_keys()

```
gui_app.KeyGenerationApp.generate_keys ( self)

Generates an RSA key pair, encrypts the private key with the entered PIN, saves the encrypted private key to the USB drive, and saves the public key locally.
```

Definition at line 124 of file gui_app.py.

Displays status messages and errors accordingly.

```
00124
           def generate_keys(self):
00125
00126
               Generates an RSA key pair, encrypts the private key with the entered PIN,
00127
               saves the encrypted private key to the USB drive, and saves the public key locally.
00128
00129
               Displays status messages and errors accordingly.
00130
00131
               pin = self.pin_entry.get()
00132
                if not pin:
                   messagebox.showerror("Błąd", "Wpisz PIN!")
00133
00134
00135
00136
               if self.last_usb_state == False:
00137
                   messagebox.showerror("Błąd", "USB nie jest podłączony!")
00138
00139
00140
               self.status value.config(text="Generowanie kluczy...", fg="blue")
00141
               self.root.update()
00142
00143
00144
                    key_generator = KeyGenerator(DEFAULT_RSA_KEY_SIZE)
                   key_generator.generate_keys()
private_key = key_generator.private_key
public_key_pem = key_generator.get_public_key_pem()
00145
00146
00147
00148
00149
                    key_encryptor = KeyEncryptor(private_key, pin)
```

```
00150
                  salt, iv, encrypted_private_pem = key_encryptor.encrypt_private_key()
00151
00152
00153
                      usb_file_path = KeyStorage.save_key_to_usb(PRIVATE_KEY_FILE, salt, iv,
     encrypted_private_pem)
00154
                      self.status_value.config(text=f"Klucz prywatny zapisany na USB", fg="green")
                      messagebox.showinfo("Sukces", f"Klucz prywatny zapisany na USB jako: {usb_file_path}")
00155
00156
00157
                     self.status_value.config(text="Nie udało się zapisać klucza prywatnego na USB!",
      fg="red")
00158
                      messagebox.showerror("Błąd", f"Nie udało się zapisać klucza prywatnego na USB:
     {str(e)}")
00159
00160
00161
                  KeyStorage.save_public_key(PUBLIC_KEY_FILE, public_key_pem)
00162
                  self.public_key_text.config(state=tk.NORMAL)
00163
                  self.public_key_text.delete(1.0, tk.END)
self.public_key_text.insert(tk.END, public_key_pem.decode('utf-8'))
00164
00165
00166
                  self.public_key_text.config(state=tk.DISABLED)
00167
00168
                  self.status_value.config(text="Klucze wygenerowane poprawnie!", fg="green")
00169
              except Exception as e:
00170
00171
                  self.status_value.config(text="Bład podczas generowania kluczy!", fg="red")
00172
                  messagebox.showerror("Błąd", f"Błąd podczas generowania kluczy!: {str(e)}")
00173
00174
```

5.3.3.3 start_auto_detect()

```
\label{eq:gui_app.KeyGenerationApp.start_auto_detect} gui\_app.KeyGenerationApp.start\_auto\_detect \; ( \\ self)
```

Starts the periodic USB connection detection loop.

Definition at line 99 of file gui app.py.

```
00099 def start_auto_detect(self):
00100 """
00101 Starts the periodic USB connection detection loop.
00102 """
00103 self.check_usb_auto()
```

5.3.4 Member Data Documentation

5.3.4.1 check_usb_auto

```
gui_app.KeyGenerationApp.check_usb_auto
```

Definition at line 122 of file gui_app.py.

5.3.4.2 generate button

gui_app.KeyGenerationApp.generate_button

Initial value:

Definition at line 66 of file gui_app.py.

5.3.4.3 key_frame

```
gui_app.KeyGenerationApp.key_frame = tk.LabelFrame(root, text="Wygenerowany klucz publiczny",
padx=10, pady=10)
```

Definition at line 87 of file gui_app.py.

5.3.4.4 last_usb_state

```
bool gui_app.KeyGenerationApp.last_usb_state = None
```

Definition at line 96 of file gui_app.py.

5.3.4.5 pin_entry

```
gui_app.KeyGenerationApp.pin_entry = tk.Entry(self.pin_frame, width=20)
```

Definition at line 62 of file gui_app.py.

5.3.4.6 pin_frame

```
gui_app.KeyGenerationApp.pin_frame = tk.Frame(root, padx=10, pady=10)
```

Definition at line 56 of file gui_app.py.

5.3.4.7 pin_label

```
gui_app.KeyGenerationApp.pin_label = tk.Label(self.pin_frame, text="Wpisz PIN:")
```

Definition at line 59 of file gui app.py.

5.3.4.8 public_key_text

 $\label{eq:gui_app.KeyGenerationApp.public_key_text} = scrolledtext.ScrolledText(self.key_frame, wrap=tk. \leftarrow \texttt{WORD})$

Definition at line 90 of file gui app.py.

5.3.4.9 root

```
gui_app.KeyGenerationApp.root = root
```

Definition at line 40 of file gui_app.py.

5.3.4.10 status_frame

```
gui_app.KeyGenerationApp.status_frame = tk.Frame(root, padx=10, pady=5)
```

Definition at line 77 of file gui_app.py.

5.3.4.11 status label

```
gui_app.KeyGenerationApp.status_label = tk.Label(self.status_frame, text="Status:")
```

Definition at line 80 of file gui_app.py.

5.3.4.12 status_value

```
gui_app.KeyGenerationApp.status_value = tk.Label(self.status_frame, text="Gotowe")
```

Definition at line 83 of file gui_app.py.

5.3.4.13 usb_detector

```
gui_app.KeyGenerationApp.usb_detector = USBDriveDetector()
```

Definition at line 95 of file gui_app.py.

5.3.4.14 usb_frame

```
gui_app.KeyGenerationApp.usb_frame = tk.Frame(root, padx=10, pady=10)
```

Definition at line 46 of file gui_app.py.

5.3.4.15 usb_status_label

```
gui_app.KeyGenerationApp.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
```

Definition at line 49 of file gui_app.py.

5.3.4.16 usb_status_value

```
gui_app.KeyGenerationApp.usb_status_value = tk.Label(self.usb_frame, text="Nowy...")
```

Definition at line 52 of file gui_app.py.

The documentation for this class was generated from the following file:

• gui_app.py

5.4 key_generator.KeyGenerator Class Reference

Public Member Functions

```
• init (self, key size=DEFAULT RSA KEY SIZE)
```

- · generate keys (self)
- get_public_key_pem (self)

Public Attributes

```
• key_size = key_size
```

- private_key = None
- public_key = None

5.4.1 Detailed Description

```
Generates RSA private and public key pairs.

Attributes:
    key_size (int): The size of the RSA key in bits.
    private_key (rsa.RSAPrivateKey): Generated RSA private key.
    public_key (rsa.RSAPublicKey): Corresponding public key.
```

Definition at line 16 of file key_generator.py.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 __init__()

Definition at line 26 of file key_generator.py.

5.4.3 Member Function Documentation

5.4.3.1 generate_keys()

```
key_generator.KeyGenerator.generate_keys ( self) Generates a new RSA private key and extracts the public key. Uses the default public exponent from the configuration.
```

Definition at line 36 of file key_generator.py.

```
00036
          def generate_keys(self):
00037
00038
              Generates a new RSA private key and extracts the public key.
00039
              Uses the default public exponent from the configuration.
00040
00041
              self.private_key = rsa.generate_private_key(
00042
                  public_exponent=DEFAULT_PUBLIC_EXPONENT,
00043
                  key_size=self.key_size
00044
00045
              self.public_key = self.private_key.public_key()
00046
```

5.4.3.2 get_public_key_pem()

Definition at line 47 of file key generator.py.

```
00047
          def get_public_key_pem(self):
00048
00049
              Returns the public key in PEM-encoded format.
00050
              :return: Bytes containing the PEM-formatted public key. """
00051
00052
00053
              return self.public_key.public_bytes(
00054
                  encoding=serialization.Encoding.PEM,
00055
                  format=serialization.PublicFormat.SubjectPublicKeyInfo
00056
```

5.4.4 Member Data Documentation

5.4.4.1 key_size

```
key_generator.KeyGenerator.key_size = key_size
```

Definition at line 32 of file key_generator.py.

5.4.4.2 private_key

```
key_generator.KeyGenerator.private_key = None
```

Definition at line 33 of file key_generator.py.

5.4.4.3 public_key

```
key_generator.KeyGenerator.public_key = None
```

Definition at line 34 of file key_generator.py.

The documentation for this class was generated from the following file:

crypto/key_generator.py

5.5 key storage. KeyStorage Class Reference

Static Public Member Functions

- save_key (file_name, salt, iv, encrypted_private_pem)
- save public key (file name, public pem)
- save_key_to_usb (file_name, salt, iv, encrypted_private_pem)
- read key from disk (file name, usb detector=None)

5.5.1 Detailed Description

```
Provides static methods to save and read cryptographic keys, supporting both local filesystem and USB drives.
```

Definition at line 13 of file key_storage.py.

5.5.2 Member Function Documentation

5.5.2.1 read_key_from_disk()

Definition at line 66 of file key_storage.py.

```
00066
          def read_key_from_disk(file_name, usb_detector=None):
00068
              Reads key data from USB drive if connected, else from local disk.
00069
              Handles file-not-found and USB errors gracefully.
00070
00071
              :param file_name: Name or path of the key file to read.
00072
              :param usb_detector: Optional USBDriveDetector instance to check USB state.
              :return: Bytes read from the key file.
00074
00075
              if usb_detector is not None and usb_detector.is_drive_connected():
00076
00077
                      usb_file_path = usb_detector.get_drive_path(file_name)
00078
                      print(f"Odczytywanie z USB: {usb_file_path}")
00079
                      with open(usb_file_path, "rb") as f:
08000
                          return f.read()
00081
                  except (FileNotFoundError, ValueError):
00082
                     print(f"Błąd podczas odczytywania")
00083
              else:
00084
                  print(f"Odczytuje z pliku: {file_name}")
00085
                  with open(file_name, "rb") as f:
00086
                      return f.read()
```

5.5.2.2 save_key()

```
key_storage.KeyStorage.save_key (
               file_name,
               salt.
               iv.
               encrypted_private_pem) [static]
Saves encrypted private key data (salt, IV, encrypted key) to a file on disk.
:param file_name: Path to the file to save the key.
:param salt: Salt bytes used for encryption.
:param iv: Initialization vector bytes used for encryption.
:param encrypted_private_pem: Encrypted private key bytes.
Definition at line 20 of file key storage.py.
00020
         def save_key(file_name, salt, iv, encrypted_private_pem):
00021
00022
              Saves encrypted private key data (salt, IV, encrypted key) to a file on disk.
00023
00024
              :param file_name: Path to the file to save the key.
              :param salt: Salt bytes used for encryption.
00025
00026
              :param iv: Initialization vector bytes used for encryption.
              :param encrypted_private_pem: Encrypted private key bytes.
00027
00028
00029
              with open(file_name, "wb") as f:
00030
                  f.write(salt + iv + encrypted_private_pem)
00031
5.5.2.3 save key to usb()
{\tt key\_storage.KeyStorage.save\_key\_to\_usb} \ (
               file_name,
               salt,
               iv.
               encrypted_private_pem) [static]
Saves the encrypted private key to a USB drive.
Checks for USB drive presence and raises ValueError if not found.
:param file_name: Name of the file to be saved on USB.
:param salt: Salt bytes used for encryption.
:param iv: Initialization vector bytes used for encryption.
:param encrypted_private_pem: Encrypted private key bytes.
:return: Full path of the saved key file on USB.
:raises ValueError: If no USB drive is connected.
Definition at line 44 of file key_storage.py.
00044
          def save_key_to_usb(file_name, salt, iv, encrypted_private_pem):
00045
00046
              Saves the encrypted private key to a USB drive.
00047
              Checks for USB drive presence and raises ValueError if not found.
00048
              :param file_name: Name of the file to be saved on USB.
00049
00050
              :param salt: Salt bytes used for encryption.
              :param iv: Initialization vector bytes used for encryption.
00051
00052
              :param encrypted_private_pem: Encrypted private key bytes.
00053
              :return: Full path of the saved key file on USB.
              :raises ValueError: If no USB drive is connected.
00054
00055
             usb_detector = USBDriveDetector()
00056
00057
              if not usb_detector.is_drive_connected():
00058
                 raise ValueError("Brak podłączonego USB!")
00059
             usb_file_path = usb_detector.get_drive_path(file_name)
with open(usb_file_path, "wb") as f:
00060
00061
                 f.write(salt + iv + encrypted_private_pem)
00062
00063
             return usb_file_path
```

00064

5.5.2.4 save_public_key()

Definition at line 33 of file key_storage.py.

```
00033 def save_public_key(file_name, public_pem):
00034 """

00035 Saves the public key in PEM format to a file on disk.
00036
00037 :param file_name: Path to the file to save the public key.
00038 :param public_pem: Public key bytes in PEM format.
00039 """
00040 with open(file_name, "wb") as f:
00041 f.write(public_pem)
```

The documentation for this class was generated from the following file:

· crypto/key_storage.py

5.6 pdf_signer.PdfSigner Class Reference

Public Member Functions

- __init__ (self, pdf_path)
- sign pdf (self, private key)

Public Attributes

• pdf path = pdf path

Protected Member Functions

• _calculate_hash (self)

5.6.1 Detailed Description

Handles signing of PDF files by computing their SHA-256 hash and creating a signature using an RSA private key.

Definition at line 14 of file pdf_signer.py.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 __init__()

Definition at line 20 of file pdf_signer.py.

5.6.3 Member Function Documentation

5.6.3.1 _calculate_hash()

Definition at line 28 of file pdf signer.py.

```
00028
           def _calculate_hash(self):
00029
                Calculates the SHA-256 hash of the PDF file content.
00030
00031
00032
                :return: The SHA-256 hash digest bytes of the PDF file.
00033
00034
               hash_sha256 = hashlib.sha256()
00035
                with open(self.pdf_path, "rb") as pdf_file:
                   # Read the file in chunks to efficiently handle large files
for chunk in iter(lambda: pdf_file.read(4096), b"):
00036
00037
                         hash_sha256.update(chunk)
00038
00039
               return hash_sha256.digest()
00040
```

5.6.3.2 sign_pdf()

Definition at line 41 of file pdf_signer.py.

```
00041
           def sign_pdf(self, private_key):
00042
00043
                Signs the PDF file by appending a signature generated using the provided private key.
00044
               The signature is created over the SHA-256 hash of the file's content.
00045
00046
                :param private_key: RSA private key object used for signing.
00047
               :return: The signature bytes that were appended to the PDF.
:raises FileNotFoundError: If the PDF file does not exist.
00048
00049
00050
                if not os.path.exists(self.pdf_path):
00051
                    raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")
00052
00053
               pdf_hash = self._calculate_hash()
00054
00055
                signature = private_key.sign(
00056
                   pdf_hash,
00057
                    padding.PKCS1v15(),
00058
                    hashes.SHA256()
00059
               # Append the signature bytes to the end of the PDF file with open(self.pdf_path, "ab") as pdf_file:
00060
00061
00062
                    pdf_file.write(signature)
00063
                return signature
```

5.6.4 Member Data Documentation

5.6.4.1 pdf_path

```
pdf_signer.PdfSigner.pdf_path = pdf_path
```

Definition at line 26 of file pdf_signer.py.

The documentation for this class was generated from the following file:

crypto/pdf_signer.py

5.7 pdf_gui_app.PdfSignerCheckerApp Class Reference

Public Member Functions

- __init__ (self, root)
- toggle_key_source (self)
- browse_public_key (self)
- browse_private_key (self)
- start_usb_detection (self)
- check_usb_auto (self)
- browse_file (self)
- sign_pdf (self)
- · verify pdf (self)

Public Attributes

- root = root
- usb_frame = tk.Frame(root, padx=10, pady=10)
- usb status label = tk.Label(self.usb frame, text="USB Status:")
- usb_status_value = tk.Label(self.usb_frame, text="Checking...")
- file frame = tk.Frame(root, padx=10, pady=10)
- file label = tk.Label(self.file frame, text="PDF File:")
- file_path_var = tk.StringVar()
- file path entry = tk.Entry(self.file frame, textvariable=self.file path var, width=40)
- browse_button = tk.Button(self.file_frame, text="Browse", command=self.browse_file)
- public key frame = tk.Frame(root, padx=10, pady=5)
- public key label = tk.Label(self.public key frame, text="Public Key:")
- public_key_path_var = tk.StringVar(value=PUBLIC_KEY_FILE)
- public_key_entry = tk.Entry(self.public_key_frame, textvariable=self.public_key_path_var, width=40)
- browse_public_button = tk.Button(self.public_key_frame, text="Browse", command=self.browse_public_key)
- private_key_frame = tk.Frame(root, padx=10, pady=5)
- private_key_label = tk.Label(self.private_key_frame, text="Private Key:")
- use_usb_key_var = tk.BooleanVar(value=True)
- · use usb key check
- private key path var = tk.StringVar()
- private_key_entry = tk.Entry(self.private_key_frame, textvariable=self.private_key_path_var, width=30, state=tk.DISABLED)
- browse_private_button = tk.Button(self.private_key_frame, text="Browse", command=self.browse_private_
 key, state=tk.DISABLED)
- op frame = tk.Frame(root, padx=10, pady=10)
- pin label = tk.Label(self.op frame, text="PIN:")
- pin entry = tk.Entry(self.op frame, width=10, show="*")
- sign button = tk.Button(self.op frame, text="Sign PDF", command=self.sign pdf, bg="#4CAF50", fg="white")
- verify_button = tk.Button(self.op_frame, text="Verify PDF", command=self.verify_pdf, bg="#2196F3", fg="white")
- status_frame = tk.Frame(root, padx=10, pady=10)
- status label = tk.Label(self.status frame, text="Status:")
- status value = tk.Label(self.status frame, text="Ready")
- results frame = tk.LabelFrame(root, text="Results", padx=10, pady=10)
- results_text = scrolledtext.ScrolledText(self.results_frame, wrap=tk.WORD)
- usb_detector = USBDriveDetector()
- last usb state = None
- · check usb auto

5.7.1 Detailed Description

GUI application for PDF signing and signature verification.

Definition at line 24 of file pdf_gui_app.py.

5.7.2 Constructor & Destructor Documentation

```
5.7.2.1 __init__()
```

```
pdf_gui_app.PdfSignerCheckerApp.__init__ (
                self.
                root)
Initializes the GUI layout and internal logic.
:param root: Tk root window.
Definition at line 28 of file pdf_gui_app.py.
          def __init__(self, root):
00030
              Initializes the GUI layout and internal logic.
00031
              :param root: Tk root window.
00032
00033
00034
              self.root = root
00035
              root.title("PDF Integrity App")
00036
              root.geometry("600x550")
00037
              root.resizable(False, False)
00038
              # === USB Status Frame ===
00039
00040
              self.usb_frame = tk.Frame(root, padx=10, pady=10)
00041
              self.usb frame.pack(fill=tk.X)
00042
00043
              self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
00044
              self.usb_status_label.pack(side=tk.LEFT)
00045
00046
              self.usb status value = tk.Label(self.usb frame, text="Checking...")
00047
              self.usb_status_value.pack(side=tk.LEFT, padx=10)
00048
00049
              # === File Selection Frame ===
00050
              self.file_frame = tk.Frame(root, padx=10, pady=10)
00051
              self.file_frame.pack(fill=tk.X)
00052
00053
              self.file_label = tk.Label(self.file_frame, text="PDF File:")
00054
              self.file_label.pack(side=tk.LEFT)
00055
00056
              self.file_path_var = tk.StringVar()
00057
              self.file_path_entry = tk.Entry(self.file_frame, textvariable=self.file_path_var, width=40)
00058
              self.file_path_entry.pack(side=tk.LEFT, padx=10)
00059
00060
              self.browse_button = tk.Button(self.file_frame, text="Browse", command=self.browse_file)
00061
              self.browse_button.pack(side=tk.LEFT)
00062
00063
              # === Public Key Selection ===
              self.public_key_frame = tk.Frame(root, padx=10, pady=5)
00064
00065
              self.public_key_frame.pack(fill=tk.X)
00066
00067
              self.public_key_label = tk.Label(self.public_key_frame, text="Public Key:")
00068
              self.public_key_label.pack(side=tk.LEFT)
00069
              self.public_key_path_var = tk.StringVar(value=PUBLIC_KEY_FILE)
00070
00071
              self.public_key_entry = tk.Entry(self.public_key_frame, textvariable=self.public_key_path_var,
      width=40)
00072
              self.public_key_entry.pack(side=tk.LEFT, padx=10)
00073
00074
              self.browse_public_button = tk.Button(self.public_key_frame, text="Browse",
      command=self.browse_public_key)
00075
              self.browse_public_button.pack(side=tk.LEFT)
00076
00077
              # === Private Key Selection =
00078
              self.private_key_frame = tk.Frame(root, padx=10, pady=5)
00079
              self.private_key_frame.pack(fill=tk.X)
08000
00081
              self.private_key_label = tk.Label(self.private_key_frame, text="Private Key:")
00082
              self.private_key_label.pack(side=tk.LEFT)
00083
00084
              self.use_usb_key_var = tk.BooleanVar(value=True)
00085
              self.use_usb_key_check = tk.Checkbutton(
00086
                  self.private_key_frame, text="Use USB key", variable=self.use_usb_key_var,
      command=self.toggle_key_source
00087
00088
              self.use_usb_key_check.pack(side=tk.LEFT)
00089
```

```
00090
              self.private_key_path_var = tk.StringVar()
              self.private_key_entry = tk.Entry(self.private_key_frame,
00091
      textvariable=self.private_key_path_var, width=30, state=tk.DISABLED)
00092
             self.private_key_entry.pack(side=tk.LEFT, padx=10)
00093
00094
              self.browse private button = tk.Button(self.private kev frame, text="Browse",
     command=self.browse_private_key, state=tk.DISABLED)
00095
             self.browse_private_button.pack(side=tk.LEFT)
00096
00097
              # === Operations Frame ===
00098
              self.op_frame = tk.Frame(root, padx=10, pady=10)
00099
             self.op_frame.pack(fill=tk.X)
00100
             self.pin_label = tk.Label(self.op_frame, text="PIN:")
00101
00102
             self.pin_label.pack(side=tk.LEFT)
00103
              self.pin_entry = tk.Entry(self.op_frame, width=10, show="*")
00104
00105
             self.pin_entry.pack(side=tk.LEFT, padx=10)
00106
00107
              self.sign_button = tk.Button(self.op_frame, text="Sign PDF", command=self.sign_pdf,
     bg="#4CAF50", fg="white")
00108
             self.sign_button.pack(side=tk.LEFT, padx=10)
00109
     self.verify_button = tk.Button(self.op_frame, text="Verify PDF", command=self.verify_pdf,
bg="#2196F3", fg="white")
00110
00111
             self.verify_button.pack(side=tk.LEFT)
00112
00113
              # === Status Frame ===
              self.status_frame = tk.Frame(root, padx=10, pady=10)
00114
00115
              self.status_frame.pack(fill=tk.X)
00116
00117
              self.status_label = tk.Label(self.status_frame, text="Status:")
00118
             self.status_label.pack(side=tk.LEFT)
00119
00120
              self.status_value = tk.Label(self.status_frame, text="Ready")
00121
              self.status_value.pack(side=tk.LEFT, padx=10)
00122
             # === Results Display ===
00124
              self.results_frame = tk.LabelFrame(root, text="Results", padx=10, pady=10)
00125
             self.results_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
00126
              self.results_text = scrolledtext.ScrolledText(self.results_frame, wrap=tk.WORD)
00127
00128
             self.results text.pack(fill=tk.BOTH, expand=True)
00129
00130
              # USB Detection Setup
00131
             self.usb_detector = USBDriveDetector()
00132
              self.last_usb_state = None
00133
              self.start_usb_detection()
00134
```

5.7.3 Member Function Documentation

5.7.3.1 browse file()

```
pdf_gui_app.PdfSignerCheckerApp.browse_file (
               self)
Opens file dialog to select a PDF file.
Definition at line 179 of file pdf gui app.py.
00179
         def browse file (self):
               """Opens file dialog to select a PDF file."""
00180
00181
              file_path = filedialog.askopenfilename(filetypes=[("PDF files", "*.pdf"), ("All files",
     "*.*")])
00182
              if file_path:
00183
                 self.file_path_var.set(file_path)
00184
                 self.status_value.config(text=f"Selected: {os.path.basename(file_path)}")
00185
```

5.7.3.2 browse_private_key()

```
{\tt pdf\_gui\_app.PdfSignerCheckerApp.browse\_private\_key \ (} \\ self)
```

Opens file dialog to select an encrypted private key PEM file.

Definition at line 152 of file pdf_gui_app.py.

5.7.3.3 browse_public_key()

```
\label{eq:pdf_gui_app.PdfSignerCheckerApp.browse_public_key (} self)
```

Opens file dialog to select a public key PEM file.

Definition at line 146 of file pdf_gui_app.py.

5.7.3.4 check_usb_auto()

Checks if a USB drive is connected and updates UI accordingly.

Definition at line 162 of file pdf_gui_app.py.

```
00162
00163
               """Checks if a USB drive is connected and updates UI accordingly."""
00164
               current_state = self.usb_detector.is_drive_connected()
00165
00166
               if current_state != self.last_usb_state:
00167
                   if current_state:
00168
                       self.usb_status_value.config(text="Connected", fg="green")
00169
                       if self.use_usb_key_var.get():
00170
                           self.sign_button.config(state=tk.NORMAL)
00171
                   else:
                       self.usb_status_value.config(text="Not connected", fg="red")
if self.use_usb_key_var.get():
00172
00173
00174
                           self.sign_button.config(state=tk.DISABLED)
00175
00176
               self.last\_usb\_state = current\_state
               self.root.after(1000, self.check_usb_auto)
00177
00178
```

5.7.3.5 sign_pdf()

Decrypts the private key using the PIN and signs the selected PDF.

Definition at line 186 of file pdf_gui_app.py.

```
00186    def sign_pdf(self):
00187          """Decrypts the private key using the PIN and signs the selected PDF."""
00188          # [function body not repeated - unchanged logic, already clear]
00189          ...
```

5.7.3.6 start usb detection()

```
{\tt pdf\_gui\_app.PdfSignerCheckerApp.start\_usb\_detection} \  ( \\ self)
```

Starts periodic USB drive detection loop.

Definition at line 158 of file pdf_gui_app.py.

```
00158 def start_usb_detection(self):
00159 """Starts periodic USB drive detection loop."""
00160 self.check_usb_auto()
00161
```

5.7.3.7 toggle_key_source()

```
\label{pdf_gui_app.PdfSignerCheckerApp.toggle_key_source (} self)
```

Enables or disables manual key file input depending on USB checkbox state.

Definition at line 135 of file pdf_gui_app.py.

```
def toggle_key_source(self):
00135
00136
00137
              Enables or disables manual key file input depending on USB checkbox state.
00138
00139
              if self.use_usb_key_var.get():
00140
                  self.private_key_entry.config(state=tk.DISABLED)
00141
                  self.browse_private_button.config(state=tk.DISABLED)
00142
              else:
00143
                 self.private_key_entry.config(state=tk.NORMAL)
00144
                  self.browse_private_button.config(state=tk.NORMAL)
00145
```

5.7.3.8 verify pdf()

Verifies the digital signature in the selected PDF using a public key.

Definition at line 191 of file pdf_gui_app.py.

5.7.4 Member Data Documentation

5.7.4.1 browse button

pdf_gui_app.PdfSignerCheckerApp.browse_button = tk.Button(self.file_frame, text="Browse",
command=self.browse_file)

Definition at line 60 of file pdf_gui_app.py.

5.7.4.2 browse_private_button

```
pdf_gui_app.PdfSignerCheckerApp.browse_private_button = tk.Button(self.private_key_frame,
text="Browse", command=self.browse_private_key, state=tk.DISABLED)
```

Definition at line 94 of file pdf_gui_app.py.

5.7.4.3 browse_public_button

pdf_gui_app.PdfSignerCheckerApp.browse_public_button = tk.Button(self.public_key_frame, text="Browse",
command=self.browse_public_key)

Definition at line 74 of file pdf_gui_app.py.

5.7.4.4 check_usb_auto

pdf_gui_app.PdfSignerCheckerApp.check_usb_auto

Definition at line 177 of file pdf gui app.py.

5.7.4.5 file_frame

```
pdf_gui_app.PdfSignerCheckerApp.file_frame = tk.Frame(root, padx=10, pady=10)
```

Definition at line 50 of file pdf_gui_app.py.

5.7.4.6 file label

```
pdf_gui_app.PdfSignerCheckerApp.file_label = tk.Label(self.file_frame, text="PDF File:")
```

Definition at line 53 of file pdf_gui_app.py.

5.7.4.7 file_path_entry

pdf_gui_app.PdfSignerCheckerApp.file_path_entry = tk.Entry(self.file_frame, textvariable=self. ← file_path_var, width=40)

Definition at line 57 of file pdf_gui_app.py.

5.7.4.8 file_path_var

```
pdf_gui_app.PdfSignerCheckerApp.file_path_var = tk.StringVar()
```

Definition at line 56 of file pdf_gui_app.py.

5.7.4.9 last_usb_state

```
pdf_gui_app.PdfSignerCheckerApp.last_usb_state = None
```

Definition at line 132 of file pdf_gui_app.py.

5.7.4.10 op_frame

```
pdf_gui_app.PdfSignerCheckerApp.op_frame = tk.Frame(root, padx=10, pady=10)
```

Definition at line 98 of file pdf gui app.py.

5.7.4.11 pin_entry

```
pdf_gui_app.PdfSignerCheckerApp.pin_entry = tk.Entry(self.op_frame, width=10, show="*")
```

Definition at line 104 of file pdf_gui_app.py.

5.7.4.12 pin_label

```
pdf_gui_app.PdfSignerCheckerApp.pin_label = tk.Label(self.op_frame, text="PIN:")
```

Definition at line 101 of file pdf gui app.py.

5.7.4.13 private_key_entry

```
pdf\_gui\_app.PdfSignerCheckerApp.private\_key\_entry = tk.Entry(self.private\_key\_frame, textvariable=self. \\ \leftarrow private\_key\_path\_var, width=30, state=tk.DISABLED)
```

Definition at line 91 of file pdf_gui_app.py.

5.7.4.14 private_key_frame

```
pdf_gui_app.PdfSignerCheckerApp.private_key_frame = tk.Frame(root, padx=10, pady=5)
```

Definition at line 78 of file pdf_gui_app.py.

5.7.4.15 private_key_label

pdf_gui_app.PdfSignerCheckerApp.private_key_label = tk.Label(self.private_key_frame, text="Private
Key:")

Definition at line 81 of file pdf_gui_app.py.

5.7.4.16 private_key_path_var

```
pdf_gui_app.PdfSignerCheckerApp.private_key_path_var = tk.StringVar()
```

Definition at line 90 of file pdf_gui_app.py.

5.7.4.17 public_key_entry

pdf_gui_app.PdfSignerCheckerApp.public_key_entry = tk.Entry(self.public_key_frame, textvariable=self. ← public_key_path_var, width=40)

Definition at line 71 of file pdf_gui_app.py.

5.7.4.18 public_key_frame

```
pdf_gui_app.PdfSignerCheckerApp.public_key_frame = tk.Frame(root, padx=10, pady=5)
```

Definition at line 64 of file pdf gui app.py.

5.7.4.19 public key label

pdf_gui_app.PdfSignerCheckerApp.public_key_label = tk.Label(self.public_key_frame, text="Public
Key:")

Definition at line 67 of file pdf_gui_app.py.

5.7.4.20 public_key_path_var

```
pdf_gui_app.PdfSignerCheckerApp.public_key_path_var = tk.StringVar(value=PUBLIC_KEY_FILE)
```

Definition at line 70 of file pdf_gui_app.py.

5.7.4.21 results_frame

pdf_gui_app.PdfSignerCheckerApp.results_frame = tk.LabelFrame(root, text="Results", padx=10,
pady=10)

Definition at line 124 of file pdf_gui_app.py.

5.7.4.22 results_text

```
pdf_gui_app.PdfSignerCheckerApp.results_text = scrolledtext.ScrolledText(self.results_frame,
wrap=tk.WORD)
```

Definition at line 127 of file pdf_gui_app.py.

5.7.4.23 root

```
pdf_gui_app.PdfSignerCheckerApp.root = root
```

Definition at line 34 of file pdf_gui_app.py.

5.7.4.24 sign_button

```
pdf_gui_app.PdfSignerCheckerApp.sign_button = tk.Button(self.op_frame, text="Sign PDF", command=self.\leftarrow sign_pdf, bg="#4CAF50", fg="white")
```

Definition at line 107 of file pdf_gui_app.py.

5.7.4.25 status_frame

```
pdf_gui_app.PdfSignerCheckerApp.status_frame = tk.Frame(root, padx=10, pady=10)
```

Definition at line 114 of file pdf_gui_app.py.

5.7.4.26 status_label

```
pdf_gui_app.PdfSignerCheckerApp.status_label = tk.Label(self.status_frame, text="Status:")
```

Definition at line 117 of file pdf_gui_app.py.

5.7.4.27 status_value

```
pdf_gui_app.PdfSignerCheckerApp.status_value = tk.Label(self.status_frame, text="Ready")
```

Definition at line 120 of file pdf gui app.py.

5.7.4.28 usb_detector

```
pdf_gui_app.PdfSignerCheckerApp.usb_detector = USBDriveDetector()
```

Definition at line 131 of file pdf_gui_app.py.

5.7.4.29 usb_frame

```
pdf_gui_app.PdfSignerCheckerApp.usb_frame = tk.Frame(root, padx=10, pady=10)
```

Definition at line 40 of file pdf gui app.py.

5.7.4.30 usb_status_label

```
pdf_gui_app.PdfSignerCheckerApp.usb_status_label = tk.Label(self.usb_frame, text="USB Status↔:")
```

Definition at line 43 of file pdf_gui_app.py.

5.7.4.31 usb_status_value

```
pdf_gui_app.PdfSignerCheckerApp.usb_status_value = tk.Label(self.usb_frame, text="Checking...")
```

Definition at line 46 of file pdf_gui_app.py.

5.7.4.32 use_usb_key_check

pdf_gui_app.PdfSignerCheckerApp.use_usb_key_check

Initial value:

```
= tk.Checkbutton(
          self.private_key_frame, text="Use USB key", variable=self.use_usb_key_var,
          command=self.toggle_key_source
          )
```

Definition at line 85 of file pdf_gui_app.py.

5.7.4.33 use_usb_key_var

```
pdf_gui_app.PdfSignerCheckerApp.use_usb_key_var = tk.BooleanVar(value=True)
```

Definition at line 84 of file pdf_gui_app.py.

5.7.4.34 verify_button

```
pdf_gui_app.PdfSignerCheckerApp.verify_button = tk.Button(self.op_frame, text="Verify PDF",
command=self.verify_pdf, bg="#2196F3", fg="white")
```

Definition at line 110 of file pdf_gui_app.py.

The documentation for this class was generated from the following file:

```
· pdf_gui_app.py
```

5.8 pdf_verifier.PdfVerifier Class Reference

Public Member Functions

- __init__ (self, pdf_path)
- extract signature (self)
- verify_signature (self, public_key)

Public Attributes

• pdf_path = pdf_path

Protected Member Functions

· _calculate_hash (self)

5.8.1 Detailed Description

Verifies the digital signature of a PDF file that was appended after the file content.

Definition at line 16 of file pdf verifier.py.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 __init__()

Definition at line 21 of file pdf_verifier.py.

5.8.3 Member Function Documentation

5.8.3.1 _calculate_hash()

Definition at line 29 of file pdf_verifier.py.

```
def _calculate_hash(self):
00031
              Calculates the SHA-256 hash of the PDF file content excluding the last 512 bytes,
00032
              which are assumed to be the signature.
00033
              :return: The SHA-256 hash digest bytes of the PDF content (excluding signature).
00034
00035
00036
              hash\_sha256 = hashlib.sha256()
00037
              with open(self.pdf_path, "rb") as pdf_file:
00038
                  content = pdf_file.read()
00039
                  # Exclude the last 512 bytes (signature size for 4096-bit RSA key)
                  hash_sha256.update(content[:-512])
00040
00041
              return hash sha256.digest()
00042
```

5.8.3.2 extract_signature()

Definition at line 43 of file pdf_verifier.py.

```
00043
          def extract_signature(self):
00044
00045
              Extracts the signature bytes appended at the end of the PDF file.
00046
              :return: The signature bytes extracted from the PDF.
00047
00048
00049
              with open(self.pdf_path, "rb") as pdf_file:
00050
                 content = pdf_file.read()
                  return content[-512:]
00051
00052
```

5.8.3.3 verify signature()

Definition at line 53 of file pdf_verifier.py.

```
00053
          def verify_signature(self, public_key):
00054
00055
               Verifies the signature appended to the PDF file using the provided RSA public key.
00056
00057
               :param public_key: RSA public key object to verify the signature.
               :return: Tuple (bool, str) indicating if the signature is valid and a message.
:raises FileNotFoundError: If the PDF file does not exist.
00058
00059
00060
00061
               if not os.path.exists(self.pdf_path):
00062
                   raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")
00063
00064
               signature = self.extract_signature()
00065
               pdf_hash = self._calculate_hash()
00066
00067
               try:
00068
                   public_key.verify(
00069
                        signature,
                        pdf_hash,
00070
                        padding.PKCS1v15(),
00071
00072
                        hashes.SHA256()
00074
                   return True, "Signature is valid"
00075
               except InvalidSignature:
00076
                   return False, "Invalid signature"
               except Exception as e:
    return False, f"Verification error: {str(e)}"
00077
00078
```

5.8.4 Member Data Documentation

5.8.4.1 pdf_path

```
pdf_verifier.PdfVerifier.pdf_path = pdf_path
```

Definition at line 27 of file pdf_verifier.py.

The documentation for this class was generated from the following file:

· crypto/pdf verifier.py

5.9 usb_detector.USBDriveDetector Class Reference

Public Member Functions

- __init__ (self)
- is_drive_connected (self)
- get_drive_path (self, filename)
- get_private_key_path (self)

Static Public Member Functions

• list_available_usb_drives ()

Protected Attributes

• _connected_drive = None

5.9.1 Detailed Description

Detects USB/removable drives and provides utility methods for interacting with them.

Definition at line 15 of file usb_detector.py.

5.9.2 Constructor & Destructor Documentation

```
5.9.2.1 __init__()
```

```
 \begin{tabular}{ll} usb\_detector. USBDriveDetector. \_\_init\_\_ ( \\ self) \end{tabular}
```

Definition at line 20 of file usb_detector.py.

```
00020 def __init__(self):

00021 self._connected_drive = None

00022
```

5.9.3 Member Function Documentation

5.9.3.1 get_drive_path()

Definition at line 47 of file usb detector.py.

```
def get_drive_path(self, filename):
00047
00048
00049
               Constructs the full path to a file located on the connected USB drive.
00051
               :param filename: Name of the file on the USB drive.
               return: Full path string.
:raises ValueError: If no USB drive is connected.
00052
00053
00054
00055
              if not self._connected_drive:
00056
                   raise ValueError("Brak podłączonego USB!")
00057
               return os.path.join(self._connected_drive, filename)
00058
```

5.9.3.2 get_private_key_path()

Definition at line 79 of file usb detector.py.

```
00079
          def get_private_key_path(self):
00080
              Searches recursively for the first '.pem' file on the connected USB drive.
00081
00082
00083
              :return: Full path to the found .pem file.
00084
              :raises ValueError: If no USB drive is connected.
00085
00086
              if not self._connected_drive:
00087
                  raise ValueError("Brak podłączonego USB!")
00088
00089
              for root, dirs, files in os.walk(self._connected_drive):
00090
                  for file in files:
00091
                      if file.endswith('.pem'):
00092
                          return os.path.join(root, file)
```

5.9.3.3 is_drive_connected()

Definition at line 23 of file usb_detector.py.

```
00023
           {\tt def} is_drive_connected(self):
00024
00025
                Checks if there is any removable or USB drive currently connected.
00026
                return: True if at least one removable USB drive is connected; False otherwise.
00027
00028
00029
                try:
00030
                     removable_drives = []
                     for partition in psutil.disk_partitions():
    # Check if the drive is removable or USB by looking at partition options
    if 'removable' in partition.opts.lower() or 'usb' in partition.opts.lower():
00031
00032
00033
00034
                              removable_drives.append(partition.device)
00035
00036
                     if not removable_drives:
00037
                          return False
00038
00039
                     # Save the first detected removable drive for future use
00040
                     self._connected_drive = removable_drives[0]
00041
                    return True
00042
00043
                except Exception as e:
                    print(f"Błąd podczas sprawdzania czy USB podłączony: {e}")
00044
00045
                     return False
00046
```

5.9.3.4 list_available_usb_drives()

```
usb_detector.USBDriveDetector.list_available_usb_drives () [static]
Lists all currently connected removable or USB drives with their volume names.
:return: List of tuples (drive_letter, volume_name).
```

Definition at line 60 of file usb_detector.py.

```
00060
            def list_available_usb_drives():
00061
00062
                 Lists all currently connected removable or USB drives with their volume names.
00063
00064
                 :return: List of tuples (drive_letter, volume_name).
00065
                 result = []
00066
                 for partition in psutil.disk_partitions():
    if 'removable' in partition.opts.lower() or 'usb' in partition.opts.lower():
00067
00068
00069
00070
                               drive = partition.device
                               volume_info = win32api.GetVolumeInformation(drive)
volume_name = volume_info[0]
result.append((drive, volume_name))
00071
00072
00073
00074
                           except Exception:
00075
                               # Ignore drives that cannot provide volume information
00076
00077
                 return result
00078
```

5.9.4 Member Data Documentation

5.9.4.1 _connected_drive

```
usb_detector.USBDriveDetector._connected_drive = None [protected]
```

Definition at line 21 of file usb_detector.py.

The documentation for this class was generated from the following file:

usb/usb_detector.py

Chapter 6

File Documentation

6.1 crypto/key_decryption.py File Reference

Decrypts an encrypted private RSA key using a PIN and AES-CBC.

Classes

• class key_decryption.KeyDecryptor

Namespaces

• namespace key_decryption

6.1.1 Detailed Description

Decrypts an encrypted private RSA key using a PIN and AES-CBC.

This module provides the KeyDecryptor class, which handles decryption of private key data encrypted with AES- \leftarrow CBC. The symmetric key is derived using PBKDF2 with the user's PIN and a salt.

Author

Kacper Witczak

Iwo Czartowski

Definition in file key_decryption.py.

6.2 key_decryption.py

Go to the documentation of this file.

```
00012
00013 from cryptography.hazmat.primitives import hashes, serialization
{\tt 00014~from~cryptography.hazmat.primitives.kdf.pbkdf2~import~PBKDF2HMAC}
00015 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
00016 from config import KEY_LENGTH, PBKDF2_ITERATIONS, SALT_LENGTH, IV_LENGTH
00018
00019 class KeyDecryptor:
00020
          Responsible for decrypting a private key encrypted with AES, using a password-based key derived from a PIN. \,
00021
00022
00023
00024
          def __init__(self, encrypted_data, pin):
00025
00026
              Constructor.
00027
              :param encrypted_data: Full encrypted content (salt + IV + encrypted key).
00028
              :param pin: The PIN string used for key derivation.
00029
00030
00031
              self.encrypted_data = encrypted_data
00032
              self.pin = pin.encode()
00033
          def derive_key(self, salt):
00034
00035
00036
              Derives a symmetric key using PBKDF2.
00037
00038
              :param salt: The salt used in key derivation.
              :return: Derived key (bytes).
00039
00040
00041
              kdf = PBKDF2HMAC(
00042
                  algorithm=hashes.SHA256(),
00043
                  length=KEY_LENGTH,
00044
                  salt=salt,
                  iterations=PBKDF2_ITERATIONS,
00045
00046
00047
              return kdf.derive(self.pin)
00048
00049
          def decrypt_private_key(self):
00050
00051
              Decrypts the encrypted private key using AES-CBC.
00052
00053
              :return: A deserialized RSA private key object.
              :raises ValueError: If decryption or deserialization fails.
00054
00055
00056
              salt = self.encrypted_data[:SALT_LENGTH]
00057
              iv = self.encrypted_data[SALT_LENGTH:SALT_LENGTH + IV_LENGTH]
00058
              encrypted_private_pem = self.encrypted_data[SALT_LENGTH + IV_LENGTH:]
00059
00060
              key = self.derive_key(salt)
00061
              cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
00062
              decryptor = cipher.decryptor()
00063
00064
              decrypted_pem_padded = decryptor.update(encrypted_private_pem) + decryptor.finalize()
00065
00066
              pad_len = decrypted_pem_padded[-1]
              decrypted_pem = decrypted_pem_padded[:-pad_len]
00068
00069
              private_key = serialization.load_pem_private_key(decrypted_pem, password=None)
00070
              return private_key
```

6.3 crypto/key_encryption.py File Reference

Encrypts an RSA private key using a PIN-derived AES key.

Classes

class key encryption.KeyEncryptor

6.4 key_encryption.py 47

Namespaces

• namespace key_encryption

6.3.1 Detailed Description

Encrypts an RSA private key using a PIN-derived AES key.

This module defines the KeyEncryptor class, which is responsible for encrypting an RSA private key using AES in CBC mode. The AES key is derived using PBKDF2 with a salt and a user-provided PIN.

Author

Kacper Witczak Iwo Czartowski

Definition in file key_encryption.py.

6.4 key_encryption.py

```
00001
00012
00013 from cryptography.hazmat.primitives import hashes, serialization
00014 from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
00015 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
00016 import os
00017 from config import SALT_LENGTH, IV_LENGTH, KEY_LENGTH, PBKDF2_ITERATIONS
00018
00019
00020 class KeyEncryptor:
00021
00022
          Class for encrypting an RSA private key using AES-CBC with a derived key from a PIN.
00023
00024
          def __init__(self, private_key, pin):
00025
00026
              Constructor.
00027
00028
              :param private_key: RSA private key object to be encrypted.
              :param pin: The user-provided PIN used to derive the AES key.
00029
00030
00031
              self.private_key = private_key
              self.pin = pin.encode()
self.salt = os.urandom(SALT_LENGTH)
00032
00033
00034
              self.iv = os.urandom(IV_LENGTH)
00035
          def derive_key(self):
00036
00037
00038
              Derives a symmetric AES key using PBKDF2 and the PIN.
00039
00040
              :return: Derived AES key (bytes).
00041
              kdf = PBKDF2HMAC(
00042
00043
                  algorithm=hashes.SHA256(),
00044
                  length=KEY_LENGTH,
00045
00046
                  iterations=PBKDF2_ITERATIONS,
00047
00048
              return kdf.derive(self.pin)
00049
          def encrypt_private_key(self):
00051
00052
              Encrypts the private key with AES-CBC.
00053
              :return: A tuple containing (salt, IV, encrypted PEM bytes).
00054
00055
00056
              key = self.derive_key()
00057
              cipher = Cipher(algorithms.AES(key), modes.CBC(self.iv))
```

```
encryptor = cipher.encryptor()
00059
00060
              private_pem = self.private_key.private_bytes(
00061
                  encoding=serialization.Encoding.PEM,
00062
                  format=serialization.PrivateFormat.PKCS8,
00063
                  encryption_algorithm=serialization.NoEncryption()
00064
00065
00066
              pad_len = 16 - (len(private_pem) % 16)
00067
              private_pem_padded = private_pem + bytes([pad_len] * pad_len)
00068
00069
              encrypted_private_pem = encryptor.update(private_pem_padded) + encryptor.finalize()
00070
00071
              return self.salt, self.iv, encrypted_private_pem
```

6.5 crypto/key generator.py File Reference

Module for generating RSA key pairs using the cryptography library.

Classes

· class key_generator.KeyGenerator

Namespaces

· namespace key_generator

6.5.1 Detailed Description

Module for generating RSA key pairs using the cryptography library.

Provides the KeyGenerator class to create RSA private and public keys, with configurable key size and default public exponent.

The public key can be exported in PEM format suitable for storage or distribution.

Definition in file key_generator.py.

6.6 key_generator.py

```
00001
00010
00011 from cryptography.hazmat.primitives.asymmetric import {\tt rsa}
00012 from cryptography.hazmat.primitives import serialization
00013 from config import DEFAULT_RSA_KEY_SIZE, DEFAULT_PUBLIC_EXPONENT
00014
00015
00016 class KeyGenerator:
00017
00018
          Generates RSA private and public key pairs.
00020
          Attributes:
00021
              key_size (int): The size of the RSA key in bits.
00022
              private_key (rsa.RSAPrivateKey): Generated RSA private key.
          public_key (rsa.RSAPublicKey): Corresponding public key.
00023
00024
00025
00026
          def __init__(self, key_size=DEFAULT_RSA_KEY_SIZE):
```

```
00027
00028
              Initializes the KeyGenerator with a specified key size.
00029
              :param key_size: Size of RSA keys in bits (default is from config).
00030
00031
              self.key_size = key_size
00032
              self.private_key = None
00034
              self.public_key = None
00035
00036
         def generate_keys(self):
00037
00038
              Generates a new RSA private key and extracts the public key.
00039
              Uses the default public exponent from the configuration.
00040
00041
              self.private_key = rsa.generate_private_key(
00042
                  public_exponent=DEFAULT_PUBLIC_EXPONENT,
00043
                  key_size=self.key_size
00044
00045
              self.public_key = self.private_key.public_key()
00046
00047
         def get_public_key_pem(self):
00048
              Returns the public key in PEM-encoded format.
00049
00050
00051
              :return: Bytes containing the PEM-formatted public key.
00052
00053
              return self.public_key.public_bytes(
00054
                  encoding=serialization.Encoding.PEM,
00055
                  {\tt format=serialization.PublicFormat.SubjectPublicKeyInfo}
00056
```

6.7 crypto/key_storage.py File Reference

Module providing static methods for storing and reading cryptographic keys, including support for USB drives detection and file operations.

Classes

· class key storage.KeyStorage

Namespaces

· namespace key storage

6.7.1 Detailed Description

Module providing static methods for storing and reading cryptographic keys, including support for USB drives detection and file operations.

KeyStorage handles saving private and public keys both on local disk and USB drives, and reading keys back with error handling and USB presence checks.

Definition in file key_storage.py.

6.8 key storage.py

```
00009
00010 from usb.usb_detector import USBDriveDetector
00011
00012
00013 class KeyStorage:
00014
00015
           Provides static methods to save and read cryptographic keys,
00016
           supporting both local filesystem and USB drives.
00017
00018
00019
           @staticmethod
00020
           def save_key(file_name, salt, iv, encrypted_private_pem):
00021
00022
               Saves encrypted private key data (salt, IV, encrypted key) to a file on disk.
00023
               :param file_name: Path to the file to save the key.
:param salt: Salt bytes used for encryption.
00024
00025
00026
               :param iv: Initialization vector bytes used for encryption.
               :param encrypted_private_pem: Encrypted private key bytes.
00027
00028
               with open(file_name, "wb") as f:
    f.write(salt + iv + encrypted_private_pem)
00029
00030
00031
00032
           @staticmethod
           def save_public_key(file_name, public_pem):
00034
00035
               Saves the public key in PEM format to a file on disk.
00036
               :param file_name: Path to the file to save the public key.
00037
00038
               :param public_pem: Public key bytes in PEM format.
00039
00040
               with open(file_name, "wb") as f:
00041
                   f.write(public_pem)
00042
00043
           @staticmethod
           def save_key_to_usb(file_name, salt, iv, encrypted_private_pem):
00044
00045
00046
               Saves the encrypted private key to a USB drive.
00047
               Checks for USB drive presence and raises ValueError if not found.
00048
               :param file_name: Name of the file to be saved on USB.
00049
00050
               :param salt: Salt bytes used for encryption.
               :param iv: Initialization vector bytes used for encryption.
00051
00052
               :param encrypted_private_pem: Encrypted private key bytes.
00053
               :return: Full path of the saved key file on USB.
               :raises ValueError: If no USB drive is connected.
00054
00055
               usb_detector = USBDriveDetector()
00056
00057
               if not usb_detector.is_drive_connected():
                   raise ValueError("Brak podłączonego USB!")
00058
00059
               usb_file_path = usb_detector.get_drive_path(file_name)
with open(usb_file_path, "wb") as f:
    f.write(salt + iv + encrypted_private_pem)
00060
00061
00062
00063
               return usb_file_path
00064
00065
           @staticmethod
           def read_key_from_disk(file_name, usb_detector=None):
00066
00067
00068
               Reads key data from USB drive if connected, else from local disk.
00069
               Handles file-not-found and USB errors gracefully.
00071
               :param file_name: Name or path of the key file to read.
00072
               :param usb_detector: Optional USBDriveDetector instance to check USB state.
               :return: Bytes read from the key file.
00073
00074
00075
               if usb detector is not None and usb detector.is drive connected():
00076
                        usb_file_path = usb_detector.get_drive_path(file_name)
00077
00078
                        print(f"Odczytywanie z USB: {usb_file_path}")
00079
                        with open(usb_file_path, "rb") as f:
00080
                            return f.read()
00081
                   except (FileNotFoundError, ValueError):
00082
                       print(f"Błąd podczas odczytywania")
00083
                   print(f"Odczytuje z pliku: {file_name}")
with open(file_name, "rb") as f:
00084
00085
00086
                        return f.read()
```

6.9 crypto/pdf signer.py File Reference

Provides functionality for hashing and signing PDF files using RSA private keys.

Classes

· class pdf signer.PdfSigner

Namespaces

· namespace pdf signer

6.9.1 Detailed Description

Provides functionality for hashing and signing PDF files using RSA private keys.

The PdfSigner class calculates SHA-256 hash of a given PDF file and appends a cryptographic signature generated with a private RSA key to the end of the file.

Definition in file pdf_signer.py.

6.10 pdf_signer.py

```
00001
00009 from cryptography.hazmat.primitives import hashes
00010 from cryptography.hazmat.primitives.asymmetric import padding
00011 import hashlib
00012 import os
00013
00014 class PdfSigner:
00015
00016
         Handles signing of PDF files by computing their SHA-256 hash
          and creating a signature using an RSA private key.
00017
00018
00019
          def __init__(self, pdf_path):
00021
00022
              Initializes the signer with the path to the PDF file.
00023
              :param pdf_path: Path to the PDF file to be signed.
00024
00025
00026
              self.pdf_path = pdf_path
00027
00028
         def _calculate_hash(self):
00029
00030
              Calculates the SHA-256 hash of the PDF file content.
00031
00032
              :return: The SHA-256 hash digest bytes of the PDF file.
00033
00034
              hash\_sha256 = hashlib.sha256()
00035
              with open(self.pdf_path, "rb") as pdf_file:
00036
                  # Read the file in chunks to efficiently handle large files
                  for chunk in iter(lambda: pdf_file.read(4096), b"):
00037
                      hash_sha256.update(chunk)
00038
00039
              return hash_sha256.digest()
00040
         def sign_pdf(self, private_key):
00041
00042
00043
              Signs the PDF file by appending a signature generated using the provided private key.
00044
              The signature is created over the SHA-256 hash of the file's content.
00045
00046
              :param private_key: RSA private key object used for signing.
```

```
:return: The signature bytes that were appended to the PDF.
               :raises FileNotFoundError: If the PDF file does not exist.
00048
00049
              if not os.path.exists(self.pdf_path):
    raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")
00050
00051
00052
              pdf_hash = self._calculate_hash()
00054
00055
               signature = private_key.sign(
                   pdf_hash,
00056
                   padding.PKCS1v15(),
00057
00058
                   hashes.SHA256()
00059
00060
               # Append the signature bytes to the end of the PDF file
00061
               with open(self.pdf_path, "ab") as pdf_file:
                   pdf_file.write(signature)
00062
00063
               return signature
```

6.11 crypto/pdf_verifier.py File Reference

Provides functionality to verify digital signatures appended to PDF files.

Classes

· class pdf verifier.PdfVerifier

Namespaces

· namespace pdf_verifier

6.11.1 Detailed Description

Provides functionality to verify digital signatures appended to PDF files.

The PdfVerifier class extracts a signature appended to the end of a PDF file, computes the SHA-256 hash of the PDF content excluding the signature, and verifies the signature using a given RSA public key.

Definition in file pdf_verifier.py.

6.12 pdf_verifier.py

```
00001
00009
00010 from cryptography.hazmat.primitives import hashes
00011 from cryptography.hazmat.primitives.asymmetric import padding
00012 from cryptography.exceptions import InvalidSignature
00013 import hashlib
00014 import os
00015
00016 class PdfVerifier:
00017
00018
          Verifies the digital signature of a PDF file that was appended after the file content.
00019
00020
00021
          def __init__(self, pdf_path):
00022
00023
              Initializes the verifier with the path to the PDF file to verify.
00024
00025
              :param pdf_path: Path to the signed PDF file.
```

```
00026
00027
              self.pdf_path = pdf_path
00028
00029
          def _calculate_hash(self):
00030
00031
              Calculates the SHA-256 hash of the PDF file content excluding the last 512 bytes,
              which are assumed to be the signature.
00033
              :return: The SHA-256 hash digest bytes of the PDF content (excluding signature). \tt^{\tt mnn}
00034
00035
00036
              hash_sha256 = hashlib.sha256()
              with open(self.pdf_path, "rb") as pdf_file:
00037
                  content = pdf_file.read()
# Exclude the last 512 bytes (signature size for 4096-bit RSA key)
00038
00039
00040
                  hash_sha256.update(content[:-512])
00041
              return hash_sha256.digest()
00042
00043
          def extract_signature(self):
00045
              Extracts the signature bytes appended at the end of the PDF file.
00046
               :return: The signature bytes extracted from the PDF.
00047
00048
              with open(self.pdf_path, "rb") as pdf_file:
    content = pdf_file.read()
00049
00050
00051
                  return content[-512:]
00052
00053
          def verify_signature(self, public_key):
00054
00055
              Verifies the signature appended to the PDF file using the provided RSA public key.
00056
00057
              :param public_key: RSA public key object to verify the signature.
00058
               return: Tuple (bool, str) indicating if the signature is valid and a message.
00059
               :raises FileNotFoundError: If the PDF file does not exist.
00060
00061
              if not os.path.exists(self.pdf_path):
00062
                  raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")
00063
00064
              signature = self.extract_signature()
00065
              pdf_hash = self._calculate_hash()
00066
00067
00068
                  public_key.verify(
00069
                       signature,
00070
                      pdf_hash,
00071
                       padding.PKCS1v15(),
00072
                      hashes.SHA256()
00073
00074
                  return True, "Signature is valid"
00075
              except InvalidSignature:
                  return False, "Invalid signature"
00077
              except Exception as e:
00078
                  return False, f"Verification error: {str(e)}"
```

6.13 gui_app.py File Reference

Classes

· class gui_app.KeyGenerationApp

Namespaces

namespace gui app

Functions

• gui_app.main ()

6.14 gui app.py

```
00001
00015
00016 import tkinter as tk
00017 from tkinter import messagebox, scrolledtext
00018 from usb.usb_detector import USBDriveDetector
00019 from crypto.key_generator import KeyGenerator
00020 from crypto.key_encryption import KeyEncryptor
00021 from crypto.key_storage import KeyStorage
00022 from config import PRIVATE_KEY_FILE, PUBLIC_KEY_FILE, DEFAULT_RSA_KEY_SIZE
00023
00024
00025 class KeyGenerationApp:
00026
00027
          GUI application for RSA key pair generation and storage on USB drive.
00028
00029
          The user inputs a PIN to encrypt the generated private key before saving it to USB.
00030
          The public key is displayed and saved locally. The application detects USB connection
00031
          status to allow or block key generation accordingly.
00032
00033
00034
          def __init__(self, root):
00035
              Initializes the GUI layout, USB detection, and event bindings.
00036
00037
              :param root: Tk root window.
00038
00039
              self.root = root
00040
              root.title("Generator Kluczy")
root.geometry("600x500")
00041
00042
00043
              root.resizable(False, False)
00044
00045
00046
              self.usb_frame = tk.Frame(root, padx=10, pady=10)
00047
              self.usb_frame.pack(fill=tk.X)
00048
              self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
00049
00050
              self.usb_status_label.pack(side=tk.LEFT)
00051
00052
              self.usb_status_value = tk.Label(self.usb_frame, text="Nowy...")
00053
              self.usb_status_value.pack(side=tk.LEFT, padx=10)
00054
00055
              # PIN
              self.pin_frame = tk.Frame(root, padx=10, pady=10)
00056
00057
              self.pin_frame.pack(fill=tk.X)
00058
00059
              self.pin_label = tk.Label(self.pin_frame, text="Wpisz PIN:")
00060
              self.pin_label.pack(side=tk.LEFT)
00061
00062
              self.pin_entry = tk.Entry(self.pin_frame, width=20)
              self.pin_entry.pack(side=tk.LEFT, padx=10)
00063
00064
00065
              # Generuj klucze
00066
              self.generate_button = tk.Button(
00067
                  self.pin_frame,
text="Generuj klucze",
00068
00069
                  command=self.generate_keys,
00070
                  bg="#4CAF50",
00071
                  fg="white",
00072
                  state=tk.DISABLED
00073
00074
              self.generate button.pack(side=tk.RIGHT)
00075
00076
00077
              self.status_frame = tk.Frame(root, padx=10, pady=5)
00078
              self.status_frame.pack(fill=tk.X)
00079
              self.status label = tk.Label(self.status frame, text="Status:")
00080
00081
              self.status label.pack(side=tk.LEFT)
00082
00083
              self.status_value = tk.Label(self.status_frame, text="Gotowe")
00084
              self.status_value.pack(side=tk.LEFT, padx=10)
00085
              # Key
00086
              self.key_frame = tk.LabelFrame(root, text="Wygenerowany klucz publiczny", padx=10, pady=10)
00087
00088
              self.key_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
00089
              self.public_key_text = scrolledtext.ScrolledText(self.key_frame, wrap=tk.WORD)
00090
00091
              self.public_key_text.pack(fill=tk.BOTH, expand=True)
00092
              self.public_key_text.config(state=tk.DISABLED)
00093
00094
00095
              self.usb_detector = USBDriveDetector()
```

6.14 gui_app.py 55

```
00096
              self.last_usb_state = None
00097
              self.start_auto_detect()
00098
00099
          def start_auto_detect(self):
00100
              Starts the periodic USB connection detection loop.
00101
00102
00103
              self.check_usb_auto()
00104
00105
          def check_usb_auto(self):
00106
00107
              Checks the USB drive connection status every second.
00108
              Updates the USB status label and enables/disables the generate button.
00109
00110
              current_state = self.usb_detector.is_drive_connected()
00111
              if current state != self.last usb state:
00112
00113
                  if current state:
00114
                      self.usb_status_value.config(text="Podłaczony", fg="green")
00115
                      self.generate_button.config(state=tk.NORMAL)
00116
00117
                      self.usb_status_value.config(text="Nie podłączony", fg="red")
                      self.generate_button.config(state=tk.DISABLED)
00118
00119
00120
              self.last_usb_state = current_state
00121
00122
              self.root.after(1000, self.check_usb_auto)
00123
00124
          def generate_keys(self):
00125
00126
              Generates an RSA key pair, encrypts the private key with the entered PIN,
00127
              saves the encrypted private key to the USB drive, and saves the public key locally.
00128
00129
              Displays status messages and errors accordingly.
00130
              pin = self.pin_entry.get()
00131
00132
              if not pin:
00133
                 messagebox.showerror("Błąd", "Wpisz PIN!")
00134
00135
              if self.last_usb_state == False:
00136
                  messagebox.showerror("Błąd", "USB nie jest podłączony!")
00137
00138
00139
00140
              self.status_value.config(text="Generowanie kluczy...", fg="blue")
00141
              self.root.update()
00142
00143
                  kev generator = KevGenerator(DEFAULT RSA KEY SIZE)
00144
00145
                  kev generator.generate kevs()
                  private_key = key_generator.private_key
00146
00147
                  public_key_pem = key_generator.get_public_key_pem()
00148
00149
                  key_encryptor = KeyEncryptor(private_key, pin)
00150
                  salt, iv, encrypted_private_pem = key_encryptor.encrypt_private_key()
00151
00152
                      usb_file_path = KeyStorage.save_key_to_usb(PRIVATE_KEY_FILE, salt, iv,
00153
      encrypted_private_pem)
00154
                      self.status_value.config(text=f"Klucz prywatny zapisany na USB", fg="green")
00155
                      messagebox.showinfo("Sukces", f"Klucz prywatny zapisany na USB jako: {usb_file_path}")
00156
                  except ValueError as e:
00157
                      self.status_value.config(text="Nie udało się zapisać klucza prywatnego na USB!",
      fg="red")
00158
                      messagebox.showerror("Błąd", f"Nie udało się zapisać klucza prywatnego na USB:
      {str(e)}")
00159
                      return
00160
00161
                  KeyStorage.save_public_key(PUBLIC_KEY_FILE, public_key_pem)
00162
00163
                  self.public_key_text.config(state=tk.NORMAL)
00164
                  self.public_key_text.delete(1.0, tk.END)
00165
                  self.public_key_text.insert(tk.END, public_key_pem.decode('utf-8'))
00166
                  self.public_key_text.config(state=tk.DISABLED)
00167
                  self.status_value.config(text="Klucze wygenerowane poprawnie!", fg="green")
00168
00169
00170
              except Exception as e:
                  self.status_value.config(text="Błąd podczas generowania kluczy!", fg="red")
00171
                  messagebox.showerror("Błąd", f"Błąd podczas generowania kluczy!: {str(e)}")
00172
00173
00174
00175 def main():
00177
          Entry point for the application.
00178
          Initializes Tkinter root and launches the key generation GUI.
00179
```

```
00180     root = tk.Tk()
00181     app = KeyGenerationApp(root)
00182     root.mainloop()
00183
00184
00185     if __name__ == "__main__":
00186          main()
```

6.15 pdf_gui_app.py File Reference

GUI application for signing and verifying PDF documents.

Classes

• class pdf_gui_app.PdfSignerCheckerApp

Namespaces

• namespace pdf_gui_app

Variables

- pdf_gui_app.root = tk.Tk()
- pdf_gui_app.app = PdfSignerCheckerApp(root)

6.15.1 Detailed Description

GUI application for signing and verifying PDF documents.

This module provides a Tkinter-based graphical interface to sign and verify PDF files using RSA keys. It supports key loading from USB or manual selection, and interacts with cryptographic utilities for key decryption, signing, and verification.

Author

Kacper Witczak Iwo Czartowski

Definition in file pdf_gui_app.py.

6.16 pdf_gui_app.py 57

6.16 pdf gui app.py

```
00012
00013 import tkinter as tk
00014 from tkinter import messagebox, scrolledtext, filedialog
00015 import os
00016 from usb.usb_detector import USBDriveDetector
00017 from crypto.pdf_signer import PdfSigner
00018 from crypto.pdf_verifier import PdfVerifier
00019 from crypto.key_decryption import KeyDecryptor
{\tt 00020\ from\ cryptography.hazmat.primitives\ import\ serialization}
00021 from config import PRIVATE_KEY_FILE, PUBLIC_KEY_FILE
00022
00023
00024 class PdfSignerCheckerApp:
00025
00026
          GUI application for PDF signing and signature verification.
00027
00028
          def __init__(self, root):
00029
00030
              Initializes the GUI layout and internal logic.
00031
               :param root: Tk root window.
00032
00033
00034
              self.root = root
00035
              root.title("PDF Integrity App")
00036
               root.geometry("600x550")
00037
              root.resizable(False, False)
00038
00039
              # === USB Status Frame ===
00040
              self.usb_frame = tk.Frame(root, padx=10, pady=10)
00041
              self.usb frame.pack(fill=tk.X)
00042
00043
              self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
00044
              self.usb_status_label.pack(side=tk.LEFT)
00045
              self.usb_status_value = tk.Label(self.usb_frame, text="Checking...")
00046
00047
              self.usb_status_value.pack(side=tk.LEFT, padx=10)
00048
00049
               # === File Selection Frame ===
00050
              self.file_frame = tk.Frame(root, padx=10, pady=10)
00051
              self.file_frame.pack(fill=tk.X)
00052
00053
              self.file_label = tk.Label(self.file_frame, text="PDF File:")
00054
              self.file_label.pack(side=tk.LEFT)
00055
              self.file_path_var = tk.StringVar()
self.file_path_entry = tk.Entry(self.file_frame, textvariable=self.file_path_var, width=40)
00056
00057
00058
              self.file_path_entry.pack(side=tk.LEFT, padx=10)
00059
00060
              self.browse_button = tk.Button(self.file_frame, text="Browse", command=self.browse_file)
00061
              self.browse_button.pack(side=tk.LEFT)
00062
00063
               # === Public Key Selection ===
              self.public_key_frame = tk.Frame(root, padx=10, pady=5)
00064
00065
              self.public_key_frame.pack(fill=tk.X)
00066
00067
              self.public_key_label = tk.Label(self.public_key_frame, text="Public Key:")
00068
              self.public_key_label.pack(side=tk.LEFT)
00069
              self.public_key_path_var = tk.StringVar(value=PUBLIC_KEY_FILE)
self.public_key_entry = tk.Entry(self.public_key_frame, textvariable=self.public_key_path_var,
00070
00071
      width=40)
00072
              self.public_key_entry.pack(side=tk.LEFT, padx=10)
00073
00074
              self.browse_public_button = tk.Button(self.public_key_frame, text="Browse",
      command=self.browse_public_key)
00075
              self.browse_public_button.pack(side=tk.LEFT)
00076
00077
               # === Private Key Selection =
00078
              self.private_key_frame = tk.Frame(root, padx=10, pady=5)
00079
              self.private_key_frame.pack(fill=tk.X)
00080
00081
               self.private_key_label = tk.Label(self.private_key_frame, text="Private Key:")
00082
              self.private_key_label.pack(side=tk.LEFT)
00083
00084
               self.use_usb_key_var = tk.BooleanVar(value=True)
              self.use_usb_key_check = tk.Checkbutton(
00085
00086
                   self.private_key_frame, text="Use USB key", variable=self.use_usb_key_var,
      command=self.toggle_key_source
00087
00088
              self.use_usb_key_check.pack(side=tk.LEFT)
00089
```

```
self.private_key_path_var = tk.StringVar()
              self.private_key_entry = tk.Entry(self.private_key_frame)
      textvariable=self.private_key_path_var, width=30, state=tk.DISABLED)
00092
              self.private_key_entry.pack(side=tk.LEFT, padx=10)
00093
00094
              self.browse private button = tk.Button(self.private key frame, text="Browse",
     command=self.browse_private_key, state=tk.DISABLED)
00095
              self.browse_private_button.pack(side=tk.LEFT)
00096
00097
              # === Operations Frame ===
              self.op_frame = tk.Frame(root, padx=10, pady=10)
00098
00099
              self.op frame.pack(fill=tk.X)
00100
              self.pin_label = tk.Label(self.op_frame, text="PIN:")
00101
00102
              self.pin_label.pack(side=tk.LEFT)
00103
              self.pin_entry = tk.Entry(self.op_frame, width=10, show="*")
self.pin_entry.pack(side=tk.LEFT, padx=10)
00104
00105
00106
00107
              self.sign_button = tk.Button(self.op_frame, text="Sign PDF", command=self.sign_pdf,
     bg="#4CAF50", fg="white")
00108
              self.sign_button.pack(side=tk.LEFT, padx=10)
00109
              self.verify_button = tk.Button(self.op_frame, text="Verify PDF", command=self.verify_pdf,
00110
     bq="#2196F3", fq="white")
00111
             self.verify_button.pack(side=tk.LEFT)
00112
00113
              # === Status Frame ===
              self.status_frame = tk.Frame(root, padx=10, pady=10)
00114
00115
              self.status_frame.pack(fill=tk.X)
00116
              self.status_label = tk.Label(self.status_frame, text="Status:")
00118
              self.status_label.pack(side=tk.LEFT)
00119
00120
              self.status_value = tk.Label(self.status_frame, text="Ready")
00121
              self.status_value.pack(side=tk.LEFT, padx=10)
00122
              # === Results Display ===
00124
              self.results_frame = tk.LabelFrame(root, text="Results", padx=10, pady=10)
00125
              self.results_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
00126
              self.results_text = scrolledtext.ScrolledText(self.results_frame, wrap=tk.WORD)
00128
              self.results_text.pack(fill=tk.BOTH, expand=True)
00129
00130
              # USB Detection Setup
00131
              self.usb_detector = USBDriveDetector()
00132
              self.last_usb_state = None
00133
              self.start usb detection()
00134
00135
          def toggle_key_source(self):
00136
00137
              Enables or disables manual key file input depending on USB checkbox state.
00138
00139
              if self.use_usb_key_var.get():
                  self.private_key_entry.config(state=tk.DISABLED)
00140
                  self.browse_private_button.config(state=tk.DISABLED)
00141
00143
                  self.private_key_entry.config(state=tk.NORMAL)
00144
                  self.browse_private_button.config(state=tk.NORMAL)
00145
00146
         def browse_public_key(self):
              """Opens file dialog to select a public key PEM file."""
00147
00148
              file_path = filedialog.askopenfilename(filetypes=[("PEM files", "*.pem"), ("All files",
     "*.*")])
00149
              if file_path:
00150
                  self.public_key_path_var.set(file_path)
00151
00152
         def browse private kev(self):
              """Opens file dialog to select an encrypted private key PEM file."""
00153
00154
              file_path = filedialog.askopenfilename(filetypes=[("PEM files", "*.pem"), ("All files",
     "*.*")])
00155
              if file_path:
00156
                  self.private_key_path_var.set(file_path)
00157
00158
         def start usb detection(self):
00159
              """Starts periodic USB drive detection loop."""
00160
              self.check_usb_auto()
00161
00162
         def check_usb_auto(self):
               """Checks if a USB drive is connected and updates UI accordingly."""
00163
              current state = self.usb detector.is drive connected()
00164
00165
00166
              if current_state != self.last_usb_state:
00167
                  if current_state:
00168
                      self.usb_status_value.config(text="Connected", fg="green")
00169
                      if self.use_usb_key_var.get():
00170
                          self.sign_button.config(state=tk.NORMAL)
```

```
00172
                       self.usb_status_value.config(text="Not connected", fg="red")
00173
                       if self.use_usb_key_var.get():
00174
                           self.sign_button.config(state=tk.DISABLED)
00175
              self.last_usb_state = current_state
00176
              self.root.after(1000, self.check_usb_auto)
00178
          def browse_file(self):
    """Opens file dialog to select a PDF file."""
00179
00180
               file_path = filedialog.askopenfilename(filetypes=[("PDF files", "*.pdf"), ("All files",
00181
      "*.*")])
00182
               if file_path:
00183
                   self.file_path_var.set(file_path)
00184
                   self.status_value.config(text=f"Selected: {os.path.basename(file_path)}")
00185
          def sign_pdf(self):
    """Decrypts the private key using the PIN and signs the selected PDF."""
00186
00187
              # [function body not repeated - unchanged logic, already clear]
00188
00189
00190
00191
          def verify_pdf(self):
               """Verifies the digital signature in the selected PDF using a public key."""
00192
              # [function body not repeated - unchanged logic, already clear]
00193
00194
00195
00196
00197 if __name__ == "__main__":
00198     root = tk.Tk()
          app = PdfSignerCheckerApp(root)
00199
          root.mainloop()
00200
```

6.17 usb/usb_detector.py File Reference

Classes

· class usb_detector.USBDriveDetector

Namespaces

• namespace usb_detector

6.18 usb_detector.py

```
00001
00009
00010 import os \,
00011 import psutil
00012 from win32 import win32api
00014
00015 class USBDriveDetector:
00016
          Detects USB/removable drives and provides utility methods for interacting with them.
00017
00018
00019
00020
                _init___(self):
00021
              self._connected_drive = None
00022
          def is_drive_connected(self):
00023
00024
00025
              Checks if there is any removable or USB drive currently connected.
00026
              :return: True if at least one removable USB drive is connected; False otherwise. """
00027
00028
00029
              try:
00030
                  removable drives = []
00031
                  for partition in psutil.disk_partitions():
00032
                       # Check if the drive is removable or USB by looking at partition options
```

```
if 'removable' in partition.opts.lower() or 'usb' in partition.opts.lower():
00034
                            removable_drives.append(partition.device)
00035
00036
                   if not removable_drives:
00037
                       return False
00038
                   # Save the first detected removable drive for future use
00040
                   self._connected_drive = removable_drives[0]
                   return True
00041
00042
00043
               except Exception as e:
                   print(f"Błąd podczas sprawdzania czy USB podłączony: {e}")
00044
00045
                   return False
00046
          def get_drive_path(self, filename):
00047
00048
               Constructs the full path to a file located on the connected USB drive.
00049
00050
00051
               :param filename: Name of the file on the USB drive.
00052
               :return: Full path string.
               raises ValueError: If no USB drive is connected.
00053
00054
00055
               if not self._connected_drive:
                  raise ValueError("Brak podłączonego USB!")
00056
00057
               return os.path.join(self._connected_drive, filename)
00058
00059
00060
          def list_available_usb_drives():
00061
               Lists all currently connected removable or USB drives with their volume names.
00062
00063
00064
               :return: List of tuples (drive_letter, volume_name).
00065
00066
               result = []
               for partition in psutil.disk_partitions():
    if 'removable' in partition.opts.lower() or 'usb' in partition.opts.lower():
00067
00068
00069
00070
                           drive = partition.device
                           volume_info = win32api.GetVolumeInformation(drive)
volume_name = volume_info[0]
00071
00072
00073
                           result.append((drive, volume_name))
00074
                       except Exception:
00075
                           # Ignore drives that cannot provide volume information
00076
00077
               return result
00078
00079
          def get_private_key_path(self):
00080
00081
               Searches recursively for the first '.pem' file on the connected USB drive.
00082
00083
               :return: Full path to the found .pem file.
               raises ValueError: If no USB drive is connected.
00084
00085
               if not self._connected_drive:
    raise ValueError("Brak podłączonego USB!")
00086
00087
00088
00089
               for root, dirs, files in os.walk(self._connected_drive):
00090
                   for file in files:
00091
                       if file.endswith('.pem'):
00092
                            return os.path.join(root, file)
```

Index

init	extract_signature
gui_app.KeyGenerationApp, 17	pdf_verifier.PdfVerifier, 40
key_decryption.KeyDecryptor, 11	
key_encryption.KeyEncryptor, 14	file_frame
key_generator.KeyGenerator, 22	pdf_gui_app.PdfSignerCheckerApp, 34
pdf_gui_app.PdfSignerCheckerApp, 30	file_label
pdf_signer.PdfSigner, 27	pdf_gui_app.PdfSignerCheckerApp, 34
pdf_verifier.PdfVerifier, 39	file_path_entry
usb_detector.USBDriveDetector, 42	pdf_gui_app.PdfSignerCheckerApp, 34
_calculate_hash	file_path_var
pdf_signer.PdfSigner, 27	pdf_gui_app.PdfSignerCheckerApp, 34
pdf_verifier.PdfVerifier, 40	
_connected_drive	generate_button
usb_detector.USBDriveDetector, 44	gui_app.KeyGenerationApp, 19
	generate_keys
арр	gui_app.KeyGenerationApp, 18
pdf_gui_app, 8	key_generator.KeyGenerator, 23
	get_drive_path
browse_button	usb_detector.USBDriveDetector, 42
pdf_gui_app.PdfSignerCheckerApp, 34	get_private_key_path
browse_file	usb_detector.USBDriveDetector, 42
pdf_gui_app.PdfSignerCheckerApp, 31	get_public_key_pem
browse_private_button	key_generator.KeyGenerator, 23
pdf_gui_app.PdfSignerCheckerApp, 34	gui_app, 7
browse_private_key	main, 7
pdf_gui_app.PdfSignerCheckerApp, 31	gui_app.KeyGenerationApp, 16
browse_public_button	init, 17
pdf_gui_app.PdfSignerCheckerApp, 34	check_usb_auto, 18, 19
browse_public_key	generate_button, 19
pdf_gui_app.PdfSignerCheckerApp, 32	generate_keys, 18
	key_frame, 19
check_usb_auto	last_usb_state, 20
gui_app.KeyGenerationApp, 18, 19	pin_entry, 20
pdf_gui_app.PdfSignerCheckerApp, 32, 34	pin_frame, 20
crypto/key_decryption.py, 45, 46	pin_label, 20
crypto/key_encryption.py, 46, 47	public_key_text, 20
crypto/key_generator.py, 48	root, 20
crypto/key_storage.py, 49, 50	start_auto_detect, 19
crypto/pdf_signer.py, 51	status_frame, 20
crypto/pdf_verifier.py, 52	status_label, 21
dearunt private key	status_value, 21
decrypt_private_key	usb_detector, 21
key_decryption.KeyDecryptor, 12	usb_frame, 21
derive_key	usb_status_label, 21
key_decryption.KeyDecryptor, 12	usb_status_value, 21
key_encryption.KeyEncryptor, 14	gui_app.py, 53
encrypt_private_key	to differ accorded
key_encryption.KeyEncryptor, 14	is_drive_connected
encrypted_data	usb_detector.USBDriveDetector, 43
key_decryption.KeyDecryptor, 13	iv

62 INDEX

key_encryption.KeyEncryptor, 15	browse_public_key, 32
	check_usb_auto, 32, 34
key_decryption, 7	file_frame, 34
key_decryption.KeyDecryptor, 11	file_label, 34
init, 11	file_path_entry, 34
decrypt_private_key, 12	file_path_var, 34
derive_key, 12	last_usb_state, 35
encrypted_data, 13	op_frame, 35
pin, 13	pin_entry, 35
key_encryption, 8	pin label, 35
key_encryption.KeyEncryptor, 13	private_key_entry, 35
init, 14	private_key_frame, 35
derive_key, 14	
encrypt_private_key, 14	private_key_label, 35
	private_key_path_var, 36
iv, 15	public_key_entry, 36
pin, 15	public_key_frame, 36
private_key, 15	public_key_label, 36
salt, 15	public_key_path_var, 36
key_frame	results_frame, 36
gui_app.KeyGenerationApp, 19	results_text, 36
key_generator, 8	root, 37
key_generator.KeyGenerator, 22	sign_button, 37
init, 22	sign_pdf, 32
generate_keys, 23	start_usb_detection, 33
get_public_key_pem, 23	status_frame, 37
key_size, 23	status_label, 37
private_key, 23	status_value, 37
public_key, 23	
key_size	toggle_key_source, 33
key_generator.KeyGenerator, 23	usb_detector, 37
	usb_frame, 37
key_storage, 8	usb_status_label, 38
key_storage.KeyStorage, 24	usb_status_value, 38
read_key_from_disk, 24	use_usb_key_check, 38
save_key, 24	use_usb_key_var, 38
save_key_to_usb, 25	verify_button, 38
save_public_key, 25	verify_pdf, 33
	pdf_gui_app.py, 56
last_usb_state	pdf_path
gui_app.KeyGenerationApp, 20	pdf signer.PdfSigner, 28
pdf_gui_app.PdfSignerCheckerApp, 35	pdf_verifier.PdfVerifier, 41
list_available_usb_drives	pdf_signer, 9
usb_detector.USBDriveDetector, 43	pdf_signer.PdfSigner, 26
	init, 27
main	nn, 27 _calculate_hash, 27
gui_app, 7	
	pdf_path, 28
op_frame	sign_pdf, 27
pdf_gui_app.PdfSignerCheckerApp, 35	pdf_verifier, 9
	pdf_verifier.PdfVerifier, 39
pdf_gui_app, 8	init, 39
app, 8	_calculate_hash, 40
root, 8	extract_signature, 40
pdf_gui_app.PdfSignerCheckerApp, 28	pdf_path, 41
init, 30	verify_signature, 40
browse_button, 34	pin
browse_file, 31	key_decryption.KeyDecryptor, 1
browse_private_button, 34	key_encryption.KeyEncryptor, 1
	pin_entry
browse_private_key, 31	gui_app.KeyGenerationApp, 20
browse_public_button, 34	gui_app.royaerierationApp, 20

INDEX 63

pdf_gui_app.PdfSignerCheckerApp, 35 pin_frame	gui_app.KeyGenerationApp, 20 pdf_gui_app.PdfSignerCheckerApp, 37
gui_app.KeyGenerationApp, 20	status label
pin_label	gui_app.KeyGenerationApp, 21
gui_app.KeyGenerationApp, 20	pdf_gui_app.PdfSignerCheckerApp, 37
pdf_gui_app.PdfSignerCheckerApp, 35	status value
private_key	gui_app.KeyGenerationApp, 21
key_encryption.KeyEncryptor, 15	pdf_gui_app.PdfSignerCheckerApp, 37
key_generator.KeyGenerator, 23	
private_key_entry	toggle_key_source
pdf_gui_app.PdfSignerCheckerApp, 35	pdf_gui_app.PdfSignerCheckerApp, 33
private_key_frame	
pdf_gui_app.PdfSignerCheckerApp, 35	usb/usb_detector.py, 59
private_key_label	usb_detector, 9
pdf_gui_app.PdfSignerCheckerApp, 35	gui_app.KeyGenerationApp, 21
private_key_path_var	pdf_gui_app.PdfSignerCheckerApp, 37
pdf_gui_app.PdfSignerCheckerApp, 36	usb_detector.USBDriveDetector, 41
public_key	init, 42
key_generator.KeyGenerator, 23	_connected_drive, 44
public_key_entry	get_drive_path, 42
pdf_gui_app.PdfSignerCheckerApp, 36	get_private_key_path, 42
public_key_frame	is_drive_connected, 43
pdf_gui_app.PdfSignerCheckerApp, 36	list_available_usb_drives, 43
public_key_label	usb_frame
pdf_gui_app.PdfSignerCheckerApp, 36	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 37
public_key_path_var	usb_status_label
pdf_gui_app.PdfSignerCheckerApp, 36	gui_app.KeyGenerationApp, 21
public_key_text	pdf_gui_app.PdfSignerCheckerApp, 38
gui_app.KeyGenerationApp, 20	
was all leave from saliste	usb_status_value
read_key_from_disk	gui_app.KeyGenerationApp, 21
key_storage.KeyStorage, 24	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38
key_storage.KeyStorage, 24 results_frame	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 38
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf pdf_gui_app.PdfSignerCheckerApp, 32	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf pdf_gui_app.PdfSignerCheckerApp, 32 pdf_signer.PdfSignerCheckerApp, 32 pdf_signer.PdfSigner, 27	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf pdf_gui_app.PdfSignerCheckerApp, 32 pdf_signer.PdfSigner, 27 start_auto_detect	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf pdf_gui_app.PdfSignerCheckerApp, 32 pdf_signer.PdfSigner, 27 start_auto_detect gui_app.KeyGenerationApp, 19	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf pdf_gui_app.PdfSignerCheckerApp, 32 pdf_signer.PdfSignerCheckerApp, 32 start_auto_detect gui_app.KeyGenerationApp, 19 start_usb_detection	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature
key_storage.KeyStorage, 24 results_frame pdf_gui_app.PdfSignerCheckerApp, 36 results_text pdf_gui_app.PdfSignerCheckerApp, 36 root gui_app.KeyGenerationApp, 20 pdf_gui_app, 8 pdf_gui_app.PdfSignerCheckerApp, 37 salt key_encryption.KeyEncryptor, 15 save_key key_storage.KeyStorage, 24 save_key_to_usb key_storage.KeyStorage, 25 save_public_key key_storage.KeyStorage, 25 sign_button pdf_gui_app.PdfSignerCheckerApp, 37 sign_pdf pdf_gui_app.PdfSignerCheckerApp, 32 pdf_signer.PdfSigner, 27 start_auto_detect gui_app.KeyGenerationApp, 19	gui_app.KeyGenerationApp, 21 pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_check pdf_gui_app.PdfSignerCheckerApp, 38 use_usb_key_var pdf_gui_app.PdfSignerCheckerApp, 38 verify_button pdf_gui_app.PdfSignerCheckerApp, 38 verify_pdf pdf_gui_app.PdfSignerCheckerApp, 33 verify_signature