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

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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__()

```
key_decryption.KeyDecryptor.__init__ (
              self,
              encrypted_data,
              pin)
```

```
Constructor.

:param encrypted_data: Full encrypted content (salt + IV + encrypted key).
:param pin: The PIN string used for key derivation.
```

Definition at line 24 of file key_decryption.py.

```
00024     def __init__(self, encrypted_data, pin):
00025         """
00026         Constructor.
00027 
00028         :param encrypted_data: Full encrypted content (salt + IV + encrypted key).
00029         :param pin: The PIN string used for key derivation.
00030         """
00031         self.encrypted_data = encrypted_data
00032         self.pin = pin.encode()
00033 
```

5.1.3 Member Function Documentation

5.1.3.1 decrypt_private_key()

```
key_decryption.KeyDecryptor.decrypt_private_key (
              self)
```

```
Decrypts the encrypted private key using AES-CBC.

:return: A deserialized RSA private key object.
:raises ValueError: If decryption or deserialization fails.
```

Definition at line 49 of file key_decryption.py.

```
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.
00054         :raises ValueError: If decryption or deserialization fails.
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]
00067         decrypted_pem = decrypted_pem_padded[:-pad_len]
00068 
00069         private_key = serialization.load_pem_private_key(decrypted_pem, password=None)
00070         return private_key
```

5.1.3.2 derive_key()

```
key_decryption.KeyDecryptor.derive_key (
              self,
              salt)
```

```
Derives a symmetric key using PBKDF2.

:param salt: The salt used in key derivation.
:return: Derived key (bytes).
```

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__()

```
key_encryption.KeyEncryptor.__init__ (
              self,
              private_key,
              pin)
```

```
Constructor.

:param private_key: RSA private key object to be encrypted.
:param pin: The user-provided PIN used to derive the AES key.
```

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.
00029         :param pin: The user-provided PIN used to derive the AES key.
00030         """
00031         self.private_key = private_key
00032         self.pin = pin.encode()
00033         self.salt = os.urandom(SALT_LENGTH)
00034         self.iv = os.urandom(IV_LENGTH)
00035
```

5.2.3 Member Function Documentation

5.2.3.1 derive_key()

```
key_encryption.KeyEncryptor.derive_key (
              self)
```

```
Derives a symmetric AES key using PBKDF2 and the PIN.

:return: Derived AES key (bytes).
```

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
00040         :return: Derived AES key (bytes).
00041         """
00042         kdf = PBKDF2HMAC(
00043             algorithm=hashes.SHA256(),
00044             length=KEY_LENGTH,
00045             salt=self.salt,
00046             iterations=PBKDF2_ITERATIONS,
00047         )
00048         return kdf.derive(self.pin)
00049
```

5.2.3.2 encrypt_private_key()

```
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.

```
00050     def encrypt_private_key(self):
00051         """
00052         Encrypts the private key with AES-CBC.
00053 
00054         :return: A tuple containing (salt, IV, encrypted PEM bytes).
00055         """
00056         key = self.derive_key()
00057         cipher = Cipher(algorithms.AES(key), modes.CBC(self.iv))
00058         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
```

5.2.4 Member Data Documentation

5.2.4.1 iv

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

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

```
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):
        """
        Initializes the GUI layout, USB detection, and event bindings.

        :param root: Tk root window.
        """
        self.root = root
        root.title("Generator Kluczy")
        root.geometry("600x500")
        root.resizable(False, False)

        # USB
        self.usb_frame = tk.Frame(root, padx=10, pady=10)
        self.usb_frame.pack(fill=tk.X)

        self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
        self.usb_status_label.pack(side=tk.LEFT)

        self.usb_status_value = tk.Label(self.usb_frame, text="Nowy...")
        self.usb_status_value.pack(side=tk.LEFT, padx=10)

        # PIN
        self.pin_frame = tk.Frame(root, padx=10, pady=10)
        self.pin_frame.pack(fill=tk.X)

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

        self.pin_entry = tk.Entry(self.pin_frame, width=20)
        self.pin_entry.pack(side=tk.LEFT, padx=10)

        # Generuj klucze
        self.generate_button = tk.Button(
            self.pin_frame,
            text="Generuj klucze",
            command=self.generate_keys,
            bg="#4CAF50",
            fg="white",
            state=tk.DISABLED
        )
        self.generate_button.pack(side=tk.RIGHT)

        # Status
        self.status_frame = tk.Frame(root, padx=10, pady=5)
        self.status_frame.pack(fill=tk.X)

        self.status_label = tk.Label(self.status_frame, text="Status:")
        self.status_label.pack(side=tk.LEFT)

        self.status_value = tk.Label(self.status_frame, text="Gotowe")
        self.status_value.pack(side=tk.LEFT, padx=10)

        # Key
        self.key_frame = tk.LabelFrame(root, text="Wygenerowany klucz publiczny", padx=10, pady=10)
        self.key_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.public_key_text = scrolledtext.ScrolledText(self.key_frame, wrap=tk.WORD)
        self.public_key_text.pack(fill=tk.BOTH, expand=True)
        self.public_key_text.config(state=tk.DISABLED)

        # Init
        self.usb_detector = USBDriveDetector()
        self.last_usb_state = None
        self.start_auto_detect()

```

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):
        """
        Checks the USB drive connection status every second.
        Updates the USB status label and enables/disables the generate button.
        """
        current_state = self.usb_detector.is_drive_connected()

        if current_state != self.last_usb_state:
            if current_state:
                self.usb_status_value.config(text="Podłączony", fg="green")
                self.generate_button.config(state=tk.NORMAL)
            else:
                self.usb_status_value.config(text="Nie podłączony", fg="red")
                self.generate_button.config(state=tk.DISABLED)

        self.last_usb_state = current_state

        self.root.after(1000, self.check_usb_auto)

```

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.

Displays status messages and errors accordingly.
```

Definition at line 124 of file gui_app.py.

```
    def 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.

        Displays status messages and errors accordingly.
        """
        pin = self.pin_entry.get()
        if not pin:
            messagebox.showerror("Błąd", "Wpisz PIN!")
            return

        if self.last_usb_state == False:
            messagebox.showerror("Błąd", "USB nie jest podłączony!")
            return

        self.status_value.config(text="Generowanie kluczy...", fg="blue")
        self.root.update()

        try:
            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()

            key_encryptor = KeyEncryptor(private_key, pin)
```

```
00150             salt, iv, encrypted_private_pem = key_encryptor.encrypt_private_key()
00151 
00152             try:
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")
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 
00168             self.status_value.config(text="Klucze wygenerowane poprawnie!", fg="green")
00169 
00170         except Exception as e:
00171             self.status_value.config(text="Błąd 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()

```
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()
00104 
```

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:

```
=  tk.Button(
            self.pin_frame,
            text="Generuj klucze",
            command=self.generate_keys,
            bg="#4CAF50",
            fg="white",
            state=tk.DISABLED
        )
```

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

```
gui_app.KeyGenerationApp.public_key_text = scrolledtext.ScrolledText(self.key_frame, wrap=tk.↩
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__()

```
key_generator.KeyGenerator.__init__ (
              self,
              key_size = DEFAULT_RSA_KEY_SIZE)
```

```
Initializes the KeyGenerator with a specified key size.

:param key_size: Size of RSA keys in bits (default is from config).
```

Definition at line 26 of file key_generator.py.

```
00026     def __init__(self, key_size=DEFAULT_RSA_KEY_SIZE):
00027         """
00028         Initializes the KeyGenerator with a specified key size.
00029 
00030         :param key_size: Size of RSA keys in bits (default is from config).
00031         """
00032         self.key_size = key_size
00033         self.private_key = None
00034         self.public_key = None
00035 
```

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()

```
key_generator.KeyGenerator.get_public_key_pem (
              self)
```

```
Returns the public key in PEM-encoded format.

:return: Bytes containing the PEM-formatted public key.
```

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 
00051         :return: Bytes containing the PEM-formatted public key.
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()

```
key_storage.KeyStorage.read_key_from_disk (
              file_name,
              usb_detector = None)  [static]
```

```
Reads key data from USB drive if connected, else from local disk.
Handles file-not-found and USB errors gracefully.

:param file_name: Name or path of the key file to read.
:param usb_detector: Optional USBDriveDetector instance to check USB state.
:return: Bytes read from the key file.
```

Definition at line 66 of file key_storage.py.

```
def read_key_from_disk(file_name, usb_detector=None):
    """
    Reads key data from USB drive if connected, else from local disk.
    Handles file-not-found and USB errors gracefully.

    :param file_name: Name or path of the key file to read.
    :param usb_detector: Optional USBDriveDetector instance to check USB state.
    :return: Bytes read from the key file.
    """
    if usb_detector is not None and usb_detector.is_drive_connected():
        try:
            usb_file_path = usb_detector.get_drive_path(file_name)
            print(f"Odczytywanie z USB: {usb_file_path}")
            with open(usb_file_path, "rb") as f:
                return f.read()
        except (FileNotFoundError, ValueError):
            print(f"Błąd podczas odczytywania")
    else:
        print(f"Odczytuje z pliku: {file_name}")
        with open(file_name, "rb") as f:
            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.
00025         :param salt: Salt bytes used for encryption.
00026         :param iv: Initialization vector bytes used for encryption.
00027         :param encrypted_private_pem: Encrypted private key bytes.
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()

```
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 
00049         :param file_name: Name of the file to be saved on USB.
00050         :param salt: Salt bytes used for encryption.
00051         :param iv: Initialization vector bytes used for encryption.
00052         :param encrypted_private_pem: Encrypted private key bytes.
00053         :return: Full path of the saved key file on USB.
00054         :raises ValueError: If no USB drive is connected.
00055         """
00056         usb_detector = USBDriveDetector()
00057         if not usb_detector.is_drive_connected():
00058             raise ValueError("Brak podłączonego USB!")
00059 
00060         usb_file_path = usb_detector.get_drive_path(file_name)
00061         with open(usb_file_path, "wb") as f:
00062             f.write(salt + iv + encrypted_private_pem)
00063         return usb_file_path
00064 
```

5.5.2.4 save_public_key()

```
key_storage.KeyStorage.save_public_key (
              file_name,
              public_pem)  [static]
```

```
Saves the public key in PEM format to a file on disk.

:param file_name: Path to the file to save the public key.
:param public_pem: Public key bytes in PEM format.
```

Definition at line 33 of file key_storage.py.

```
    def save_public_key(file_name, public_pem):
        """
        Saves the public key in PEM format to a file on disk.

        :param file_name: Path to the file to save the public key.
        :param public_pem: Public key bytes in PEM format.
        """
        with open(file_name, "wb") as f:
            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__()

```
pdf_signer.PdfSigner.__init__ (
               self,
               pdf_path)
```

```
Initializes the signer with the path to the PDF file.

:param pdf_path: Path to the PDF file to be signed.
```

Definition at line 20 of file pdf_signer.py.

```
00020     def __init__(self, pdf_path):
00021         """
00022         Initializes the signer with the path to the PDF file.
00023 
00024         :param pdf_path: Path to the PDF file to be signed.
00025         """
00026         self.pdf_path = pdf_path
00027 
```

5.6.3 Member Function Documentation

5.6.3.1 _calculate_hash()

```
pdf_signer.PdfSigner._calculate_hash (
               self)  [protected]
```

```
Calculates the SHA-256 hash of the PDF file content.

:return: The SHA-256 hash digest bytes of the PDF file.
```

Definition at line 28 of file pdf_signer.py.

```
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
00037             for chunk in iter(lambda: pdf_file.read(4096), b”):
00038                 hash_sha256.update(chunk)
00039         return hash_sha256.digest()
00040 
```

5.6.3.2 sign_pdf()

```
pdf_signer.PdfSigner.sign_pdf (
               self,
               private_key)
```

```
Signs the PDF file by appending a signature generated using the provided private key.
The signature is created over the SHA-256 hash of the file’s content.

: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.
```

Definition at line 41 of file pdf_signer.py.

```python
    def sign_pdf(self, private_key):
        """
        Signs the PDF file by appending a signature generated using the provided private key.
        The signature is created over the SHA-256 hash of the file’s content.

        :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.
        """
        if not os.path.exists(self.pdf_path):
            raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")

        pdf_hash = self._calculate_hash()

        signature = private_key.sign(
            pdf_hash,
            padding.PKCS1v15(),
            hashes.SHA256()
        )
        # Append the signature bytes to the end of the PDF file
        with open(self.pdf_path, "ab") as pdf_file:
            pdf_file.write(signature)
        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.

```
00028     def __init__(self, root):
00029         """
00030         Initializes the GUI layout and internal logic.
00031 
00032         :param root: Tk root window.
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 
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 ===
00064         self.public_key_frame = tk.Frame(root, padx=10, pady=5)
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 
00070         self.public_key_path_var = tk.StringVar(value=PUBLIC_KEY_FILE)
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)
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)
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 
```

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

        self.browse_private_button = tk.Button(self.private_key_frame, text="Browse", command=self.browse_private_key, state=tk.DISABLED)
        self.browse_private_button.pack(side=tk.LEFT)

        # === Operations Frame ===
        self.op_frame = tk.Frame(root, padx=10, pady=10)
        self.op_frame.pack(fill=tk.X)

        self.pin_label = tk.Label(self.op_frame, text="PIN:")
        self.pin_label.pack(side=tk.LEFT)

        self.pin_entry = tk.Entry(self.op_frame, width=10, show="*")
        self.pin_entry.pack(side=tk.LEFT, padx=10)

        self.sign_button = tk.Button(self.op_frame, text="Sign PDF", command=self.sign_pdf, bg="#4CAF50", fg="white")
        self.sign_button.pack(side=tk.LEFT, padx=10)

        self.verify_button = tk.Button(self.op_frame, text="Verify PDF", command=self.verify_pdf, bg="#2196F3", fg="white")
        self.verify_button.pack(side=tk.LEFT)

        # === Status Frame ===
        self.status_frame = tk.Frame(root, padx=10, pady=10)
        self.status_frame.pack(fill=tk.X)

        self.status_label = tk.Label(self.status_frame, text="Status:")
        self.status_label.pack(side=tk.LEFT)

        self.status_value = tk.Label(self.status_frame, text="Ready")
        self.status_value.pack(side=tk.LEFT, padx=10)

        # === Results Display ===
        self.results_frame = tk.LabelFrame(root, text="Results", padx=10, pady=10)
        self.results_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.results_text = scrolledtext.ScrolledText(self.results_frame, wrap=tk.WORD)
        self.results_text.pack(fill=tk.BOTH, expand=True)

        # USB Detection Setup
        self.usb_detector = USBDriveDetector()
        self.last_usb_state = None
        self.start_usb_detection()

```

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):
00180         """Opens file dialog to select a PDF file."""
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()

```
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.

```
00152     def browse_private_key(self):
00153         """Opens file dialog to select an encrypted private key PEM file."""
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 
```

5.7.3.3 browse_public_key()

```
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.

```
00146     def browse_public_key(self):
00147         """Opens file dialog to select a public key PEM file."""
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 
```

5.7.3.4 check_usb_auto()

```
pdf_gui_app.PdfSignerCheckerApp.check_usb_auto (
              self)
```

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

Definition at line 162 of file pdf_gui_app.py.

```
00162     def check_usb_auto(self):
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:
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 
00176         self.last_usb_state = current_state
00177         self.root.after(1000, self.check_usb_auto)
00178 
```

5.7.3.5 sign_pdf()

```
pdf_gui_app.PdfSignerCheckerApp.sign_pdf (
              self)
```

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

Definition at line 186 of file pdf_gui_app.py.

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

```

5.7.3.6 start_usb_detection()

```
pdf_gui_app.PdfSignerCheckerApp.start_usb_detection (
              self)
```

Starts periodic USB drive detection loop.

Definition at line 158 of file pdf_gui_app.py.

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

```

5.7.3.7 toggle_key_source()

```
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):
    """
    Enables or disables manual key file input depending on USB checkbox state.
    """
    if self.use_usb_key_var.get():
        self.private_key_entry.config(state=tk.DISABLED)
        self.browse_private_button.config(state=tk.DISABLED)
    else:
        self.private_key_entry.config(state=tk.NORMAL)
        self.browse_private_button.config(state=tk.NORMAL)

```

5.7.3.8 verify_pdf()

```
pdf_gui_app.PdfSignerCheckerApp.verify_pdf (
              self)
```

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

Definition at line 191 of file pdf_gui_app.py.

```
def verify_pdf(self):
    """Verifies the digital signature in the selected PDF using a public key."""
    # [function body not repeated - unchanged logic, already clear]
    ...


```

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.↩
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.←
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__()

```
pdf_verifier.PdfVerifier.__init__ (
                self,
                pdf_path)
```

```
Initializes the verifier with the path to the PDF file to verify.

:param pdf_path: Path to the signed PDF file.
```

Definition at line 21 of file pdf_verifier.py.

```
    def __init__(self, pdf_path):
        """
        Initializes the verifier with the path to the PDF file to verify.

        :param pdf_path: Path to the signed PDF file.
        """
        self.pdf_path = pdf_path

```

5.8.3 Member Function Documentation

5.8.3.1 _calculate_hash()

```
pdf_verifier.PdfVerifier._calculate_hash (
              self)  [protected]
```

```
Calculates the SHA-256 hash of the PDF file content excluding the last 512 bytes,
which are assumed to be the signature.

:return: The SHA-256 hash digest bytes of the PDF content (excluding signature).
```

Definition at line 29 of file pdf_verifier.py.

```
00029     def _calculate_hash(self):
00030         """
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 
00034         :return: The SHA-256 hash digest bytes of the PDF content (excluding signature).
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)
00040             hash_sha256.update(content[:-512])
00041         return hash_sha256.digest()
00042 
```

5.8.3.2 extract_signature()

```
pdf_verifier.PdfVerifier.extract_signature (
              self)
```

```
Extracts the signature bytes appended at the end of the PDF file.

:return: The signature bytes extracted from the PDF.
```

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 
00047         :return: The signature bytes extracted from the PDF.
00048         """
00049         with open(self.pdf_path, "rb") as pdf_file:
00050             content = pdf_file.read()
00051             return content[-512:]
00052 
```

5.8.3.3 verify_signature()

```
pdf_verifier.PdfVerifier.verify_signature (
              self,
              public_key)
```

```
Verifies the signature appended to the PDF file using the provided RSA public key.

: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.
```

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.
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         try:
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:
00076             return False, "Invalid signature"
00077         except Exception as e:
00078             return False, f"Verification error: {str(e)}"
```

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__()

```
usb_detector.USBDriveDetector.__init__ (
              self)
```

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()

```
usb_detector.USBDriveDetector.get_drive_path (
              self,
              filename)
```

```
Constructs the full path to a file located on the connected USB drive.

:param filename: Name of the file on the USB drive.
:return: Full path string.
:raises ValueError: If no USB drive is connected.
```

Definition at line 47 of file usb_detector.py.

```
00047     def get_drive_path(self, filename):
00048         """
00049         Constructs the full path to a file located on the connected USB drive.
00050
00051         :param filename: Name of the file on the USB drive.
00052         :return: Full path string.
00053         :raises ValueError: If no USB drive is connected.
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()

```
usb_detector.USBDriveDetector.get_private_key_path (
              self)
```

```
Searches recursively for the first '.pem' file on the connected USB drive.

:return: Full path to the found .pem file.
:raises ValueError: If no USB drive is connected.
```

Definition at line 79 of file usb_detector.py.

```
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.
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()

```
usb_detector.USBDriveDetector.is_drive_connected (
              self)
```

```
Checks if there is any removable or USB drive currently connected.

:return: True if at least one removable USB drive is connected; False otherwise.
```

Definition at line 23 of file usb_detector.py.

```
00023     def is_drive_connected(self):
00024         """
00025         Checks if there is any removable or USB drive currently connected.
00026 
00027         :return: True if at least one removable USB drive is connected; False otherwise.
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
00033                 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 
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:
00044             print(f"Błąd podczas sprawdzania czy USB podłączony: {e}")
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         """
00066         result = []
00067         for partition in psutil.disk_partitions():
00068             if ’removable’ in partition.opts.lower() or ’usb’ in partition.opts.lower():
00069                 try:
00070                     drive = partition.device
00071                     volume_info = win32api.GetVolumeInformation(drive)
00072                     volume_name = volume_info[0]
00073                     result.append((drive, volume_name))
00074                 except Exception:
00075                     # Ignore drives that cannot provide volume information
00076                     pass
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-↩ 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.

```


from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from config import KEY_LENGTH, PBKDF2_ITERATIONS, SALT_LENGTH, IV_LENGTH


class KeyDecryptor:
    """
    Responsible for decrypting a private key encrypted with AES,
    using a password-based key derived from a PIN.
    """
    def __init__(self, encrypted_data, pin):
        """
        Constructor.

        :param encrypted_data: Full encrypted content (salt + IV + encrypted key).
        :param pin: The PIN string used for key derivation.
        """
        self.encrypted_data = encrypted_data
        self.pin = pin.encode()

    def derive_key(self, salt):
        """
        Derives a symmetric key using PBKDF2.

        :param salt: The salt used in key derivation.
        :return: Derived key (bytes).
        """
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=KEY_LENGTH,
            salt=salt,
            iterations=PBKDF2_ITERATIONS,
        )
        return kdf.derive(self.pin)

    def decrypt_private_key(self):
        """
        Decrypts the encrypted private key using AES-CBC.

        :return: A deserialized RSA private key object.
        :raises ValueError: If decryption or deserialization fails.
        """
        salt = self.encrypted_data[:SALT_LENGTH]
        iv = self.encrypted_data[SALT_LENGTH:SALT_LENGTH + IV_LENGTH]
        encrypted_private_pem = self.encrypted_data[SALT_LENGTH + IV_LENGTH:]

        key = self.derive_key(salt)
        cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
        decryptor = cipher.decryptor()

        decrypted_pem_padded = decryptor.update(encrypted_private_pem) + decryptor.finalize()

        pad_len = decrypted_pem_padded[-1]
        decrypted_pem = decrypted_pem_padded[:-pad_len]

        private_key = serialization.load_pem_private_key(decrypted_pem, password=None)
        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

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

Go to the documentation of this file.

```python


from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os
from config import SALT_LENGTH, IV_LENGTH, KEY_LENGTH, PBKDF2_ITERATIONS


class KeyEncryptor:
    """
    Class for encrypting an RSA private key using AES-CBC with a derived key from a PIN.
    """
    def __init__(self, private_key, pin):
        """
        Constructor.

        :param private_key: RSA private key object to be encrypted.
        :param pin: The user-provided PIN used to derive the AES key.
        """
        self.private_key = private_key
        self.pin = pin.encode()
        self.salt = os.urandom(SALT_LENGTH)
        self.iv = os.urandom(IV_LENGTH)

    def derive_key(self):
        """
        Derives a symmetric AES key using PBKDF2 and the PIN.

        :return: Derived AES key (bytes).
        """
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=KEY_LENGTH,
            salt=self.salt,
            iterations=PBKDF2_ITERATIONS,
        )
        return kdf.derive(self.pin)

    def encrypt_private_key(self):
        """
        Encrypts the private key with AES-CBC.

        :return: A tuple containing (salt, IV, encrypted PEM bytes).
        """
        key = self.derive_key()
        cipher = Cipher(algorithms.AES(key), modes.CBC(self.iv))
        encryptor = cipher.encryptor()

        private_pem = self.private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption()
        )

        pad_len = 16 - (len(private_pem) % 16)
        private_pem_padded = private_pem + bytes([pad_len] * pad_len)

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

        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

Go to the documentation of this file.

```

from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
from config import DEFAULT_RSA_KEY_SIZE, DEFAULT_PUBLIC_EXPONENT


class KeyGenerator:
    """
    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.
    """

    def __init__(self, key_size=DEFAULT_RSA_KEY_SIZE):
        """
        Initializes the KeyGenerator with a specified key size.

        :param key_size: Size of RSA keys in bits (default is from config).
        """
        self.key_size = key_size
        self.private_key = None
        self.public_key = None

    def generate_keys(self):
        """
        Generates a new RSA private key and extracts the public key.
        Uses the default public exponent from the configuration.
        """
        self.private_key = rsa.generate_private_key(
            public_exponent=DEFAULT_PUBLIC_EXPONENT,
            key_size=self.key_size
        )
        self.public_key = self.private_key.public_key()

    def get_public_key_pem(self):
        """
        Returns the public key in PEM-encoded format.

        :return: Bytes containing the PEM-formatted public key.
        """
        return self.public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
```

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

Go to the documentation of this file.

```


from usb.usb_detector import USBDriveDetector


class KeyStorage:
    """
    Provides static methods to save and read cryptographic keys,
    supporting both local filesystem and USB drives.
    """

    @staticmethod
    def save_key(file_name, salt, iv, encrypted_private_pem):
        """
        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.
        """
        with open(file_name, "wb") as f:
            f.write(salt + iv + encrypted_private_pem)

    @staticmethod
    def save_public_key(file_name, public_pem):
        """
        Saves the public key in PEM format to a file on disk.

        :param file_name: Path to the file to save the public key.
        :param public_pem: Public key bytes in PEM format.
        """
        with open(file_name, "wb") as f:
            f.write(public_pem)

    @staticmethod
    def save_key_to_usb(file_name, salt, iv, encrypted_private_pem):
        """
        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.
        """
        usb_detector = USBDriveDetector()
        if not usb_detector.is_drive_connected():
            raise ValueError("Brak podłączonego USB!")

        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)
        return usb_file_path

    @staticmethod
    def read_key_from_disk(file_name, usb_detector=None):
        """
        Reads key data from USB drive if connected, else from local disk.
        Handles file-not-found and USB errors gracefully.

        :param file_name: Name or path of the key file to read.
        :param usb_detector: Optional USBDriveDetector instance to check USB state.
        :return: Bytes read from the key file.
        """
        if usb_detector is not None and usb_detector.is_drive_connected():
            try:
                usb_file_path = usb_detector.get_drive_path(file_name)
                print(f"Odczytywanie z USB: {usb_file_path}")
                with open(usb_file_path, "rb") as f:
                    return f.read()
            except (FileNotFoundError, ValueError):
                print(f"Błąd podczas odczytywania")
        else:
            print(f"Odczytuje z pliku: {file_name}")
            with open(file_name, "rb") as f:
                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

Go to the documentation of this file.

```python

from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
import hashlib
import os

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

    def __init__(self, pdf_path):
        """
        Initializes the signer with the path to the PDF file.

        :param pdf_path: Path to the PDF file to be signed.
        """
        self.pdf_path = pdf_path

    def _calculate_hash(self):
        """
        Calculates the SHA-256 hash of the PDF file content.

        :return: The SHA-256 hash digest bytes of the PDF file.
        """
        hash_sha256 = hashlib.sha256()
        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”):
                hash_sha256.update(chunk)
        return hash_sha256.digest()

    def sign_pdf(self, private_key):
        """
        Signs the PDF file by appending a signature generated using the provided private key.
        The signature is created over the SHA-256 hash of the file’s content.

        :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.
        """
        if not os.path.exists(self.pdf_path):
            raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")

        pdf_hash = self._calculate_hash()

        signature = private_key.sign(
            pdf_hash,
            padding.PKCS1v15(),
            hashes.SHA256()
        )
        # Append the signature bytes to the end of the PDF file
        with open(self.pdf_path, "ab") as pdf_file:
            pdf_file.write(signature)
        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

Go to the documentation of this file.

```


from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.exceptions import InvalidSignature
import hashlib
import os

class PdfVerifier:
    """
    Verifies the digital signature of a PDF file that was appended after the file content.
    """

    def __init__(self, pdf_path):
        """
        Initializes the verifier with the path to the PDF file to verify.

        :param pdf_path: Path to the signed PDF file.
        """
        self.pdf_path = pdf_path

    def _calculate_hash(self):
        """
        Calculates the SHA-256 hash of the PDF file content excluding the last 512 bytes,
        which are assumed to be the signature.

        :return: The SHA-256 hash digest bytes of the PDF content (excluding signature).
        """
        hash_sha256 = hashlib.sha256()
        with open(self.pdf_path, "rb") as pdf_file:
            content = pdf_file.read()
            # Exclude the last 512 bytes (signature size for 4096-bit RSA key)
            hash_sha256.update(content[:-512])
        return hash_sha256.digest()

    def extract_signature(self):
        """
        Extracts the signature bytes appended at the end of the PDF file.

        :return: The signature bytes extracted from the PDF.
        """
        with open(self.pdf_path, "rb") as pdf_file:
            content = pdf_file.read()
            return content[-512:]

    def verify_signature(self, public_key):
        """
        Verifies the signature appended to the PDF file using the provided RSA public key.

        :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.
        """
        if not os.path.exists(self.pdf_path):
            raise FileNotFoundError(f"PDF file not found: {self.pdf_path}")

        signature = self.extract_signature()
        pdf_hash = self._calculate_hash()

        try:
            public_key.verify(
                signature,
                pdf_hash,
                padding.PKCS1v15(),
                hashes.SHA256()
            )
            return True, "Signature is valid"
        except InvalidSignature:
            return False, "Invalid signature"
        except Exception as e:
            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

Go to the documentation of this file.

```


import tkinter as tk
from tkinter import messagebox, scrolledtext
from usb.usb_detector import USBDriveDetector
from crypto.key_generator import KeyGenerator
from crypto.key_encryption import KeyEncryptor
from crypto.key_storage import KeyStorage
from config import PRIVATE_KEY_FILE, PUBLIC_KEY_FILE, DEFAULT_RSA_KEY_SIZE


class KeyGenerationApp:
    """
    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.
    """

    def __init__(self, root):
        """
        Initializes the GUI layout, USB detection, and event bindings.

        :param root: Tk root window.
        """
        self.root = root
        root.title("Generator Kluczy")
        root.geometry("600x500")
        root.resizable(False, False)

        # USB
        self.usb_frame = tk.Frame(root, padx=10, pady=10)
        self.usb_frame.pack(fill=tk.X)

        self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
        self.usb_status_label.pack(side=tk.LEFT)

        self.usb_status_value = tk.Label(self.usb_frame, text="Nowy...")
        self.usb_status_value.pack(side=tk.LEFT, padx=10)

        # PIN
        self.pin_frame = tk.Frame(root, padx=10, pady=10)
        self.pin_frame.pack(fill=tk.X)

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

        self.pin_entry = tk.Entry(self.pin_frame, width=20)
        self.pin_entry.pack(side=tk.LEFT, padx=10)

        # Generuj klucze
        self.generate_button = tk.Button(
            self.pin_frame,
            text="Generuj klucze",
            command=self.generate_keys,
            bg="#4CAF50",
            fg="white",
            state=tk.DISABLED
        )
        self.generate_button.pack(side=tk.RIGHT)

        # Status
        self.status_frame = tk.Frame(root, padx=10, pady=5)
        self.status_frame.pack(fill=tk.X)

        self.status_label = tk.Label(self.status_frame, text="Status:")
        self.status_label.pack(side=tk.LEFT)

        self.status_value = tk.Label(self.status_frame, text="Gotowe")
        self.status_value.pack(side=tk.LEFT, padx=10)

        # Key
        self.key_frame = tk.LabelFrame(root, text="Wygenerowany klucz publiczny", padx=10, pady=10)
        self.key_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.public_key_text = scrolledtext.ScrolledText(self.key_frame, wrap=tk.WORD)
        self.public_key_text.pack(fill=tk.BOTH, expand=True)
        self.public_key_text.config(state=tk.DISABLED)

        # Init
        self.usb_detector = USBDriveDetector()
        self.last_usb_state = None
        self.start_auto_detect()

    def start_auto_detect(self):
        """
        Starts the periodic USB connection detection loop.
        """
        self.check_usb_auto()

    def check_usb_auto(self):
        """
        Checks the USB drive connection status every second.
        Updates the USB status label and enables/disables the generate button.
        """
        current_state = self.usb_detector.is_drive_connected()

        if current_state != self.last_usb_state:
            if current_state:
                self.usb_status_value.config(text="Podłączony", fg="green")
                self.generate_button.config(state=tk.NORMAL)
            else:
                self.usb_status_value.config(text="Nie podłączony", fg="red")
                self.generate_button.config(state=tk.DISABLED)

        self.last_usb_state = current_state

        self.root.after(1000, self.check_usb_auto)

    def 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.

        Displays status messages and errors accordingly.
        """
        pin = self.pin_entry.get()
        if not pin:
            messagebox.showerror("Błąd", "Wpisz PIN!")
            return

        if self.last_usb_state == False:
            messagebox.showerror("Błąd", "USB nie jest podłączony!")
            return

        self.status_value.config(text="Generowanie kluczy...", fg="blue")
        self.root.update()

        try:
            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()

            key_encryptor = KeyEncryptor(private_key, pin)
            salt, iv, encrypted_private_pem = key_encryptor.encrypt_private_key()

            try:
                usb_file_path = KeyStorage.save_key_to_usb(PRIVATE_KEY_FILE, salt, iv,
encrypted_private_pem)
                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}")
            except ValueError as e:
                self.status_value.config(text="Nie udało się zapisać klucza prywatnego na USB!",
fg="red")
                messagebox.showerror("Błąd", f"Nie udało się zapisać klucza prywatnego na USB:
{str(e)}")
                return

            KeyStorage.save_public_key(PUBLIC_KEY_FILE, public_key_pem)

            self.public_key_text.config(state=tk.NORMAL)
            self.public_key_text.delete(1.0, tk.END)
            self.public_key_text.insert(tk.END, public_key_pem.decode('utf-8'))
            self.public_key_text.config(state=tk.DISABLED)

            self.status_value.config(text="Klucze wygenerowane poprawnie!", fg="green")

        except Exception as e:
            self.status_value.config(text="Błąd podczas generowania kluczy!", fg="red")
            messagebox.showerror("Błąd", f"Błąd podczas generowania kluczy!: {str(e)}")


def main():
    """
    Entry point for the application.
    Initializes Tkinter root and launches the key generation GUI.
    """
    root = tk.Tk()
    app = KeyGenerationApp(root)
    root.mainloop()


if __name__ == "__main__":
    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

Go to the documentation of this file.

```


import tkinter as tk
from tkinter import messagebox, scrolledtext, filedialog
import os
from usb.usb_detector import USBDriveDetector
from crypto.pdf_signer import PdfSigner
from crypto.pdf_verifier import PdfVerifier
from crypto.key_decryption import KeyDecryptor
from cryptography.hazmat.primitives import serialization
from config import PRIVATE_KEY_FILE, PUBLIC_KEY_FILE


class PdfSignerCheckerApp:
    """
    GUI application for PDF signing and signature verification.
    """
    def __init__(self, root):
        """
        Initializes the GUI layout and internal logic.

        :param root: Tk root window.
        """
        self.root = root
        root.title("PDF Integrity App")
        root.geometry("600x550")
        root.resizable(False, False)

        # === USB Status Frame ===
        self.usb_frame = tk.Frame(root, padx=10, pady=10)
        self.usb_frame.pack(fill=tk.X)

        self.usb_status_label = tk.Label(self.usb_frame, text="USB Status:")
        self.usb_status_label.pack(side=tk.LEFT)

        self.usb_status_value = tk.Label(self.usb_frame, text="Checking...")
        self.usb_status_value.pack(side=tk.LEFT, padx=10)

        # === File Selection Frame ===
        self.file_frame = tk.Frame(root, padx=10, pady=10)
        self.file_frame.pack(fill=tk.X)

        self.file_label = tk.Label(self.file_frame, text="PDF File:")
        self.file_label.pack(side=tk.LEFT)

        self.file_path_var = tk.StringVar()
        self.file_path_entry = tk.Entry(self.file_frame, textvariable=self.file_path_var, width=40)
        self.file_path_entry.pack(side=tk.LEFT, padx=10)

        self.browse_button = tk.Button(self.file_frame, text="Browse", command=self.browse_file)
        self.browse_button.pack(side=tk.LEFT)

        # === Public Key Selection ===
        self.public_key_frame = tk.Frame(root, padx=10, pady=5)
        self.public_key_frame.pack(fill=tk.X)

        self.public_key_label = tk.Label(self.public_key_frame, text="Public Key:")
        self.public_key_label.pack(side=tk.LEFT)

        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, width=40)
        self.public_key_entry.pack(side=tk.LEFT, padx=10)

        self.browse_public_button = tk.Button(self.public_key_frame, text="Browse", command=self.browse_public_key)
        self.browse_public_button.pack(side=tk.LEFT)

        # === Private Key Selection ===
        self.private_key_frame = tk.Frame(root, padx=10, pady=5)
        self.private_key_frame.pack(fill=tk.X)

        self.private_key_label = tk.Label(self.private_key_frame, text="Private Key:")
        self.private_key_label.pack(side=tk.LEFT)

        self.use_usb_key_var = tk.BooleanVar(value=True)
        self.use_usb_key_check = tk.Checkbutton(
            self.private_key_frame, text="Use USB key", variable=self.use_usb_key_var, command=self.toggle_key_source
        )
        self.use_usb_key_check.pack(side=tk.LEFT)

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

        self.browse_private_button = tk.Button(self.private_key_frame, text="Browse", command=self.browse_private_key, state=tk.DISABLED)
        self.browse_private_button.pack(side=tk.LEFT)

        # === Operations Frame ===
        self.op_frame = tk.Frame(root, padx=10, pady=10)
        self.op_frame.pack(fill=tk.X)

        self.pin_label = tk.Label(self.op_frame, text="PIN:")
        self.pin_label.pack(side=tk.LEFT)

        self.pin_entry = tk.Entry(self.op_frame, width=10, show="*")
        self.pin_entry.pack(side=tk.LEFT, padx=10)

        self.sign_button = tk.Button(self.op_frame, text="Sign PDF", command=self.sign_pdf, bg="#4CAF50", fg="white")
        self.sign_button.pack(side=tk.LEFT, padx=10)

        self.verify_button = tk.Button(self.op_frame, text="Verify PDF", command=self.verify_pdf, bg="#2196F3", fg="white")
        self.verify_button.pack(side=tk.LEFT)

        # === Status Frame ===
        self.status_frame = tk.Frame(root, padx=10, pady=10)
        self.status_frame.pack(fill=tk.X)

        self.status_label = tk.Label(self.status_frame, text="Status:")
        self.status_label.pack(side=tk.LEFT)

        self.status_value = tk.Label(self.status_frame, text="Ready")
        self.status_value.pack(side=tk.LEFT, padx=10)

        # === Results Display ===
        self.results_frame = tk.LabelFrame(root, text="Results", padx=10, pady=10)
        self.results_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.results_text = scrolledtext.ScrolledText(self.results_frame, wrap=tk.WORD)
        self.results_text.pack(fill=tk.BOTH, expand=True)

        # USB Detection Setup
        self.usb_detector = USBDriveDetector()
        self.last_usb_state = None
        self.start_usb_detection()

    def toggle_key_source(self):
        """
        Enables or disables manual key file input depending on USB checkbox state.
        """
        if self.use_usb_key_var.get():
            self.private_key_entry.config(state=tk.DISABLED)
            self.browse_private_button.config(state=tk.DISABLED)
        else:
            self.private_key_entry.config(state=tk.NORMAL)
            self.browse_private_button.config(state=tk.NORMAL)

    def browse_public_key(self):
        """Opens file dialog to select a public key PEM file."""
        file_path = filedialog.askopenfilename(filetypes=[("PEM files", "*.pem"), ("All files", "*.*")])
        if file_path:
            self.public_key_path_var.set(file_path)

    def browse_private_key(self):
        """Opens file dialog to select an encrypted private key PEM file."""
        file_path = filedialog.askopenfilename(filetypes=[("PEM files", "*.pem"), ("All files", "*.*")])
        if file_path:
            self.private_key_path_var.set(file_path)

    def start_usb_detection(self):
        """Starts periodic USB drive detection loop."""
        self.check_usb_auto()

    def check_usb_auto(self):
        """Checks if a USB drive is connected and updates UI accordingly."""
        current_state = self.usb_detector.is_drive_connected()

        if current_state != self.last_usb_state:
            if current_state:
                self.usb_status_value.config(text="Connected", fg="green")
                if self.use_usb_key_var.get():
                    self.sign_button.config(state=tk.NORMAL)
            else:
                self.usb_status_value.config(text="Not connected", fg="red")
                if self.use_usb_key_var.get():
                    self.sign_button.config(state=tk.DISABLED)

        self.last_usb_state = current_state
        self.root.after(1000, self.check_usb_auto)

    def browse_file(self):
        """Opens file dialog to select a PDF file."""
        file_path = filedialog.askopenfilename(filetypes=[("PDF files", "*.pdf"), ("All files", "*.*")])
        if file_path:
            self.file_path_var.set(file_path)
            self.status_value.config(text=f"Selected: {os.path.basename(file_path)}")

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

    def verify_pdf(self):
        """Verifies the digital signature in the selected PDF using a public key."""
        # [function body not repeated - unchanged logic, already clear]
        ...


if __name__ == "__main__":
    root = tk.Tk()
    app = PdfSignerCheckerApp(root)
    root.mainloop()
```

6.17 usb/usb_detector.py File Reference

Classes

- class usb_detector.USBDriveDetector

Namespaces

- namespace usb_detector

6.18 usb_detector.py

Go to the documentation of this file.

```


import os
import psutil
from win32 import win32api


class USBDriveDetector:
    """
    Detects USB/removable drives and provides utility methods for interacting with them.
    """

    def __init__(self):
        self._connected_drive = None

    def is_drive_connected(self):
        """
        Checks if there is any removable or USB drive currently connected.

        :return: True if at least one removable USB drive is connected; False otherwise.
        """
        try:
            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():
                    removable_drives.append(partition.device)

            if not removable_drives:
                return False

            # Save the first detected removable drive for future use
            self._connected_drive = removable_drives[0]
            return True

        except Exception as e:
            print(f"Błąd podczas sprawdzania czy USB podłączony: {e}")
            return False

    def get_drive_path(self, filename):
        """
        Constructs the full path to a file located on the connected USB drive.

        :param filename: Name of the file on the USB drive.
        :return: Full path string.
        :raises ValueError: If no USB drive is connected.
        """
        if not self._connected_drive:
            raise ValueError("Brak podłączonego USB!")
        return os.path.join(self._connected_drive, filename)

    @staticmethod
    def list_available_usb_drives():
        """
        Lists all currently connected removable or USB drives with their volume names.

        :return: List of tuples (drive_letter, volume_name).
        """
        result = []
        for partition in psutil.disk_partitions():
            if 'removable' in partition.opts.lower() or 'usb' in partition.opts.lower():
                try:
                    drive = partition.device
                    volume_info = win32api.GetVolumeInformation(drive)
                    volume_name = volume_info[0]
                    result.append((drive, volume_name))
                except Exception:
                    # Ignore drives that cannot provide volume information
                    pass
        return result

    def get_private_key_path(self):
        """
        Searches recursively for the first '.pem' file on the connected USB drive.

        :return: Full path to the found .pem file.
        :raises ValueError: If no USB drive is connected.
        """
        if not self._connected_drive:
            raise ValueError("Brak podłączonego USB!")

        for root, dirs, files in os.walk(self._connected_drive):
            for file in files:
                if file.endswith('.pem'):
                    return os.path.join(root, file)
```

Index