PADES_imitation_app

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Namespace Documentation

3.1 key_generation_main Namespace Reference

Functions

- encrypt_private_key (pk, pin)
- generate_rsa_keys ()
- generate_aes_key (pin, salt)
- update_task_progress (value, text)
- generate_keys ()
- save_public ()
- save_private ()

Variables

- int rsa_bits = 4096
- aes_mode = AES.MODE_GCM
- main_window = tk.Tk()
- background
- style = ttk.Style()
- padding
- font
- thickness
- weight
- text
- row
- column
- padx
- pady
- sticky
- passphrase_entry = ttk.Entry(main_window, show="*")
- gen_button = ttk.Button(main_window, text="Generate Keys", command=generate_keys, style="Custom.↩ TButton")
- columnspan
- progress_bar = ttk.Progressbar(main_window, length=290)
- status_label = ttk.Label(main_window, text="", font=("Arial", 10))
- save_public_button = ttk.Button(main_window, text="Save Public Key", command=save_public, state=tk.↩ DISABLED, style="Custom.TButton")
- save_private_button = ttk.Button(main_window, text="Save Private Key", command=save_private, state=tk.↩ DISABLED, style="Custom.TButton")

3.1.1 Function Documentation

3.1.1.1 encrypt_private_key()

```
key_generation_main.encrypt_private_key (
              pk,
              pin)
```

```
    @brief Encrypts a private RSA key using a PIN.

    @param pk The private RSA key to encrypt.
    @param pin The PIN used to encrypt the key.

    @return The encrypted key data.
```

3.1.1.2 generate_aes_key()

```
key_generation_main.generate_aes_key (
              pin,
              salt)
```

```
    @brief Generates an AES encryption key based on a PIN and salt.

    @param pin The user's PIN used for key derivation.
    @param salt The salt used in the key derivation process.

    @return The derived AES key.
```

3.1.1.3 generate_keys()

```
key_generation_main.generate_keys ()
```

```
    @brief Initiates the RSA and AES key generation process

    This function coordinates the creation of RSA and AES keys,
    and may update the UI to reflect progress.
```

3.1.1.4 generate_rsa_keys()

```
key_generation_main.generate_rsa_keys ()
```

```
    @brief Generates a new RSA key pair (private and public keys).

    @return A tuple containing the private key and public key.
```

3.1.1.5 save_private()

```
key_generation_main.save_private ()
```

@brief Saves the encrypted private key to a file.

Asks the user to choose a location and writes the key securely.

3.1.1.6 save_public()

```
key_generation_main.save_public ()
```

@brief Saves the generated public key to a file.

Asks the user to choose a location and writes the key in PEM format.

3.1.1.7 update_task_progress()

```
key_generation_main.update_task_progress (
              value,
              text)
```

@brief Updates the GUI task progress bar and status message.

@param value The progress percentage (0-100).
@param text The status message to display.

3.1.2 Variable Documentation

3.1.2.1 aes_mode

```
key_generation_main.aes_mode = AES.MODE_GCM
```

3.1.2.2 background

```
key_generation_main.background
```

3.1.2.3 column

```
key_generation_main.column
```

3.1.2.4 columnspan

```
key_generation_main.columnspan
```

3.1.2.5 font

```
key_generation_main.font
```

3.1.2.6 gen_button

```
key_generation_main.gen_button = ttk.Button(main_window, text="Generate Keys", command=generate_keys,
style="Custom.TButton")
```

3.1.2.7 main_window

```
key_generation_main.main_window = tk.Tk()
```

3.1.2.8 padding

```
key_generation_main.padding
```

3.1.2.9 padx

```
key_generation_main.padx
```

3.1.2.10 pady

```
key_generation_main.pady
```

3.1.2.11 passphrase_entry

```
key_generation_main.passphrase_entry = ttk.Entry(main_window, show="*")
```

3.1.2.12 progress_bar

```
key_generation_main.progress_bar = ttk.Progressbar(main_window, length=290)
```

3.1.2.13 row

```
key_generation_main.row
```

3.1.2.14 rsa_bits

```
int key_generation_main.rsa_bits = 4096
```

3.1.2.15 save_private_button

```
key_generation_main.save_private_button = ttk.Button(main_window, text="Save Private Key",
command=save_private, state=tk.DISABLED, style="Custom.TButton")
```

3.1.2.16 save_public_button

```
key_generation_main.save_public_button = ttk.Button(main_window, text="Save Public Key", command=save_public,
state=tk.DISABLED, style="Custom.TButton")
```

3.1.2.17 status_label

```
key_generation_main.status_label = ttk.Label(main_window, text="", font=("Arial", 10))
```

3.1.2.18 sticky

```
key_generation_main.sticky
```

3.1.2.19 style

```
key_generation_main.style = ttk.Style()
```

3.1.2.20 text

```
key_generation_main.text
```

3.1.2.21 thickness

```
key_generation_main.thickness
```

3.1.2.22 weight

```
key_generation_main.weight
```

3.2 signing_app_main Namespace Reference

Functions

- decrypt_private_key (pk_path, passphrase)
- bytes adjust_metadata (str pdf_path, Optional[List[str]] remove_fields_metadata=None, Optional[Dict[str, Any]] add_fields_metadata=None)
- sign_pdf (private_key_pem, pdf_path)
- verify_signature (signed_pdf_path, public_key_path)
- detect_pendrive ()
- select_pdf_to_sign ()
- check_signature ()
- select_private_key ()

Variables

- aes_mode = AES.MODE_GCM
- PRIVATE_KEY = None
- bool MANUAL_KEY_SELECTION = False
- main_window = tk.Tk()
- weight
- bg
- style = ttk.Style()
- padding
- font
- background
- frame = ttk.Frame(main_window, padding=20)
- relx
- rely
- anchor
- row
- column
- sticky
- sign_pdf_button = ttk.Button(frame, text="Sign PDF", state=tk.DISABLED, command=select_pdf_to_sign, style="Custom.TButton")
- padx
- pady
- usb_status_label = ttk.Label(frame, text="There are no keys on USB detected", font=("Arial", 10))
- check_signature_button = ttk.Button(frame, text="Check Signature", command=check_signature, style="Custom.↩ TButton")
- select_key_button = ttk.Button(frame, text="Select Private Key", command=select_private_key, style="Custom.↩ TButton")
- usb_thread = threading.Thread(target=detect_pendrive, daemon=True)

3.2.1 Function Documentation

3.2.1.1 adjust_metadata()

```
bytes signing_app_main.adjust_metadata (
            str pdf_path,
            Optional[List[str]]  remove_fields_metadata = None,
            Optional[Dict[str, Any]]  add_fields_metadata = None)
```

```
   @brief Adjusts metadata of a PDF file.

   This function modifies the metadata of a PDF file by removing/adding specific fields
   as specified by the parameters. It allows for updating the PDF's metadata without altering
   the content of the document itself.

   @param pdf_path The path to the PDF file whose metadata is to be adjusted.
   @param remove_fields_metadata A list of metadata field names to remove from the PDF.
                                 (Optional, default is None).
   @param add_fields_metadata A dictionary of metadata field names and values to add to the PDF.
                              (Optional, default is None).

   @return The PDF file as a byte stream with the updated metadata.
```

3.2.1.2 check_signature()

```
signing_app_main.check_signature ()
```

@brief Checks the digital signature of the currently selected PDF.

@return None. Displays the result of the signature check.

3.2.1.3 decrypt_private_key()

```
signing_app_main.decrypt_private_key (
              pk_path,
              passphrase)
```

@brief Decrypts a private key stored in a file.

This function loads an encrypted private key from the specified file and decrypts it using the provided passphrase. The decrypted private key is returned for use in cryptographic operations such as signing documents.

@param pk_path The path to the file containing the encrypted private key.
@param passphrase The passphrase used to decrypt the private key.

@return The decrypted private key as a string.

3.2.1.4 detect_pendrive()

```
signing_app_main.detect_pendrive ()
```

@brief Detects if a USB pendrive is connected to the system.

@return The path to the detected pendrive or None if not found.

3.2.1.5 select_pdf_to_sign()

```
signing_app_main.select_pdf_to_sign ()
```

@brief Opens a file dialog to select a PDF file to sign.

@return None. Stores the selected PDF path for signing.

3.2.1.6 select_private_key()

```
signing_app_main.select_private_key ()
```

@brief Opens a file dialog to select a private key file.

@return None. Stores the selected private key for signing purposes.

3.2.1.7 sign_pdf()

```
signing_app_main.sign_pdf (
              private_key_pem,
              pdf_path)
```

```
    @brief Signs a PDF file using a private key.

    This function signs a PDF file with the provided private key, generating a cryptographic
    signature that ensures the authenticity of the document. The private key is used to sign
    the PDF, ensuring that the content has not been changed since the signature was applied.

    @param private_key_pem The private key in PEM format used for signing the PDF.
    @param pdf_path The path to the PDF file that needs to be signed.

    @return The signed PDF as a byte stream.
```

3.2.1.8 verify_signature()

```
signing_app_main.verify_signature (
              signed_pdf_path,
              public_key_path)
```

```
    @brief Verifies the digital signature of a PDF document.

    @param signed_pdf_path Path to the signed PDF file.
    @param public_key_path Path to the public key used for verification.

    @return None. Displays result of verification.
```

3.2.2 Variable Documentation

3.2.2.1 aes_mode

```
signing_app_main.aes_mode = AES.MODE_GCM
```

3.2.2.2 anchor

```
signing_app_main.anchor
```

3.2.2.3 background

```
signing_app_main.background
```

3.2.2.4 bg

```
signing_app_main.bg
```

3.2.2.5 check_signature_button

```
signing_app_main.check_signature_button = ttk.Button(frame, text="Check Signature", command=check_signature,
style="Custom.TButton")
```

3.2.2.6 column

```
signing_app_main.column
```

3.2.2.7 font

```
signing_app_main.font
```

3.2.2.8 frame

```
signing_app_main.frame = ttk.Frame(main_window, padding=20)
```

3.2.2.9 main_window

```
signing_app_main.main_window = tk.Tk()
```

3.2.2.10 MANUAL_KEY_SELECTION

```
bool signing_app_main.MANUAL_KEY_SELECTION = False
```

3.2.2.11 padding

```
signing_app_main.padding
```

3.2.2.12 padx

```
signing_app_main.padx
```

3.2.2.13 pady

```
signing_app_main.pady
```

3.2.2.14 PRIVATE_KEY

```
signing_app_main.PRIVATE_KEY = None
```

3.2.2.15 relx

```
signing_app_main.relx
```

3.2.2.16 rely

```
signing_app_main.rely
```

3.2.2.17 row

```
signing_app_main.row
```

3.2.2.18 select_key_button

```
signing_app_main.select_key_button = ttk.Button(frame, text="Select Private Key", command=select_private_key,
style="Custom.TButton")
```

3.2.2.19 sign_pdf_button

```
signing_app_main.sign_pdf_button = ttk.Button(frame, text="Sign PDF", state=tk.DISABLED, command=select_pdf_to
style="Custom.TButton")
```

3.2.2.20 sticky

```
signing_app_main.sticky
```

3.2.2.21 style

```
signing_app_main.style = ttk.Style()
```

3.2.2.22 usb_status_label

```
signing_app_main.usb_status_label = ttk.Label(frame, text="There are no keys on USB detected",
font=("Arial", 10))
```

3.2.2.23 usb_thread

```
signing_app_main.usb_thread = threading.Thread(target=detect_pendrive, daemon=True)
```

3.2.2.24 weight

```
signing_app_main.weight
```

Chapter 4

File Documentation

4.1 key_generation_app/key_generation_main.py File Reference

GUI application for secure key generation using RSA and AES encryption.

Namespaces

- namespace key_generation_main

Functions

- key_generation_main.encrypt_private_key (pk, pin)
- key_generation_main.generate_rsa_keys ()
- key_generation_main.generate_aes_key (pin, salt)
- key_generation_main.update_task_progress (value, text)
- key_generation_main.generate_keys ()
- key_generation_main.save_public ()
- key_generation_main.save_private ()

Variables

- int key_generation_main.rsa_bits = 4096
- key_generation_main.aes_mode = AES.MODE_GCM
- key_generation_main.main_window = tk.Tk()
- key_generation_main.background
- key_generation_main.style = ttk.Style()
- key_generation_main.padding
- key_generation_main.font
- key_generation_main.thickness
- key_generation_main.weight
- key_generation_main.text
- key_generation_main.row
- key_generation_main.column
- key_generation_main.padx
- key_generation_main.pady

- key_generation_main.sticky
- key_generation_main.passphrase_entry = ttk.Entry(main_window, show="∗")
- key_generation_main.gen_button = ttk.Button(main_window, text="Generate Keys", command=generate_keys, style="Custom.TButton")
- key_generation_main.columnspan
- key_generation_main.progress_bar = ttk.Progressbar(main_window, length=290)
- key_generation_main.status_label = ttk.Label(main_window, text="", font=("Arial", 10))
- key_generation_main.save_public_button = ttk.Button(main_window, text="Save Public Key", command=save_public, state=tk.DISABLED, style="Custom.TButton")
- key_generation_main.save_private_button = ttk.Button(main_window, text="Save Private Key", command=save_private, state=tk.DISABLED, style="Custom.TButton")

4.1.1 Detailed Description

GUI application for secure key generation using RSA and AES encryption.

This app provides a GUI for users to generate private and public keys. The application allows for:

- Generating RSA key pairs (public/private)
- Deriving AES keys using a PIN and salt
- Saving keys as files

GUI is built using Tkinter and includes feedback mechanisms to guide the user through the key generation process.

Date

2025-04-23

4.2 signing_app/signing_app_main.py File Reference

Namespaces

- namespace signing_app_main

Functions

- signing_app_main.decrypt_private_key (pk_path, passphrase)
- bytes signing_app_main.adjust_metadata (str pdf_path, Optional[List[str]] remove_fields_metadata=None, Optional[Dict[str, Any]] add_fields_metadata=None)
- signing_app_main.sign_pdf (private_key_pem, pdf_path)
- signing_app_main.verify_signature (signed_pdf_path, public_key_path)
- signing_app_main.detect_pendrive ()
- signing_app_main.select_pdf_to_sign ()
- signing_app_main.check_signature ()
- signing_app_main.select_private_key ()

Variables

- signing_app_main.aes_mode = AES.MODE_GCM
- signing_app_main.PRIVATE_KEY = None
- bool signing_app_main.MANUAL_KEY_SELECTION = False
- signing_app_main.main_window = tk.Tk()
- signing_app_main.weight
- signing_app_main.bg
- signing_app_main.style = ttk.Style()
- signing_app_main.padding
- signing_app_main.font
- signing_app_main.background
- signing_app_main.frame = ttk.Frame(main_window, padding=20)
- signing_app_main.relx
- signing_app_main.rely
- signing_app_main.anchor
- signing_app_main.row
- signing_app_main.column
- signing_app_main.sticky
- signing_app_main.sign_pdf_button = ttk.Button(frame, text="Sign PDF", state=tk.DISABLED, command=select_pdf_to_sign, style="Custom.TButton")
- signing_app_main.padx
- signing_app_main.pady
- signing_app_main.usb_status_label = ttk.Label(frame, text="There are no keys on USB detected", font=("Arial", 10))
- signing_app_main.check_signature_button = ttk.Button(frame, text="Check Signature", command=check_signature, style="Custom.TButton")
- signing_app_main.select_key_button = ttk.Button(frame, text="Select Private Key", command=select_private_key, style="Custom.TButton")
- signing_app_main.usb_thread = threading.Thread(target=detect_pendrive, daemon=True)

Index