Security of Computer Systems

Project Report

Authors:
Julian Wasylka 193223
Igor Józefowicz 193257

Version: 1.2

Versions

Version	Date	Description of changes
1.0	13.04.2025	Creation of the document
1.1	02.06.2025	Completion of the document
1.2	09.06.2025	Added Doxygen documentation

1. Project – control term

1.1 Description

This phase focused on implementing an RSA-4096 key generator. The application securely stores private keys using AES-256-EAX encryption, with the key derived from a PIN provided by a user through SHA-256. The GUI allows users to input a PIN, select an output directory (ideally a pendrive), and generate cryptographic key pairs. Private keys are never stored in plaintext.

1.2 Results

The application generates:

- rsa_private.bin: the AES-encrypted RSA private key,
- rsa_public.pem: the corresponding public key.

Key generation runs in a separate thread to prevent GUI freezing, with progress displayed via a progress bar.

1.3 Summary

The key generator successfully meets all functional and security requirements for cryptographic key generation. Keys are strongly encrypted, and the application effectively handles errors and validates user input.

2. Project - final term

2.1 Description

The final project is a complete PAdES-compliant digital signature tool. It allows users to sign and verify PDF documents using RSA-4096 and SHA-256. Private keys are loaded from USB pendrives, decrypted using the user PIN, and used to sign PDF hashes. The application uses *PyQt5* for its GUI and supports both signing and signature verification workflows.

The complete source code of the application is available on GitHub:
https://github.com/julianwasylka/Emulating-the-PAdES-Qualified-Electronic-Signature

2.2 Code Description

Listing 1 – Private key decryption and loading from pendrive

```python
def load_private_key_from_pendrive(pendrive_path, pin):
    """Load and decrypt private key from pendrive"""
    try:
        key_path = os.path.join(pendrive_path, "rsa_private.bin")
        if not os.path.exists(key_path):
            raise FileNotFoundError("Private key file not found on pendrive")

        with open(key_path, "rb") as f:
            data = f.read()

        # Extract nonce, tag, and encrypted data
        nonce = data[:16]
        tag = data[16:32]
        encrypted_private_key = data[32:]

        # Create AES key from PIN
        aes_key = SHA256.new(pin.encode()).digest()

        # Decrypt private key
        cipher = AES.new(aes_key, AES.MODE_EAX, nonce=nonce)
        private_key_data = cipher.decrypt_and_verify(encrypted_private_key, tag)

        # Import RSA key
        rsa_key = RSA.import_key(private_key_data)
        return rsa_key
```

This function loads the encrypted RSA private key from the pendrive, decrypts it using a key derived from the user's PIN via SHA-256, and imports it using the RSA module. It ensures that the private key never exists unencrypted on disk.

Listing 2 – Signature verification logic

```python
def verify_signature(hash_digest, signature, public_key):
    """Verify signature with public key"""
    try:
        verifier = PKCS1_v1_5.new(public_key)
        hash_obj = SHA256.new()
        hash_obj.update(hash_digest)
        return verifier.verify(hash_obj, signature)
    except Exception as e:
        return False
```

This function verifies a digital signature by recalculating the PDF's hash and checking it with the provided signature. It ensures the integrity and authenticity of the signed document.

Listing 3 – PDF signing process

```python
def create_signed_pdf(original_pdf_path, signature, hash_hex, output_path, signer_info="Anonymous"):
    """Create a new PDF with signature information embedded"""
    try:
        # Read original PDF
        with open(original_pdf_path, "rb") as f:
            pdf_reader = PyPDF2.PdfReader(f)

            # Create signature info
            signature_info = {
                'signature': signature.hex(),
                'hash': hash_hex,
                'timestamp': datetime.datetime.now().isoformat(),
                'signer': signer_info,
                'algorithm': 'RSA-4096 + SHA-256'
            }

            # Add metadata to PDF
            pdf_writer = PyPDF2.PdfWriter()

            # Copy all pages
            for page in pdf_reader.pages:
                pdf_writer.add_page(page)

            # Add signature metadata
            pdf_writer.add_metadata({
                '/PAdES_Signature': str(signature_info),
                '/SignatureHash': hash_hex,
                '/SignatureTimestamp': signature_info['timestamp'],
                '/SignatureAlgorithm': signature_info['algorithm']
            })

            # Write signed PDF
            with open(output_path, "wb") as output_file:
                pdf_writer.write(output_file)

            return True
```

This function signs a PDF document by generating a digital signature and embedding it into the PDF metadata. All original pages are preserved, and signature-related information is added in a PAdES-compliant manner.

Listing 4 – Pendrive detection logic

```python
def get_removable_drives():
    drives = []
    partitions = psutil.disk_partitions(all=False)

    for partition in partitions:
        if platform.system() == 'Windows':
            if 'removable' in partition.opts.lower():
                drives.append({
                    'device': partition.device,
                    'mountpoint': partition.mountpoint,
                    'fstype': partition.fstype
                })
        else:
            if partition.mountpoint.startswith("/media") or partition.mountpoint.startswith("/run/media"):
                drives.append({
                    'device': partition.device,
                    'mountpoint': partition.mountpoint,
                    'fstype': partition.fstype
                })
    return drives
```

This function detects connected removable drives by scanning system disk partitions.

2.3 Tests

The application has been tested with the following scenarios:

- RSA key generation with various PIN lengths
- Document signing with different PDF sizes and formats
- Signature verification with valid signatures
- Detecting changes in signed documents
- Pendrive detection across different USB devices
- Error handling for invalid PINs and missing keys

2.4 Results

- Successful integration of GUI, file handling, encryption, and signature generation.
- User feedback by progress bars and status indicators.
- Correct cryptographic processing using *PyCryptodome* and secure storage of keys.
- Fully working PDF signing and verification.

2.5 Summary

The final project meets all functional and security requirements. It implements a secure, user-friendly PAdES-compliant signing tool with support for key management, digital signatures, GUI interface, and error handling. The modular architecture ensures maintainability and clarity.

3. Literature

(1) Online Doxygen documentation, https://www.doxygen.nl/manual/lists.html

(2) AES Encryption & Decryption In Python, https://onboardbase.com/blog/aes-encryption-decryption

(3) PyCryptodome documentation, https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html

(4) Encrypt and decrypt using PyCrypto AES-256, https://stackoverflow.com/questions/12524994/encrypt-and-decrypt-using-pycrypto-aes-256

(5) Python progress bar, https://stackoverflow.com/questions/3160699/python-progress-bar

(6) Platform module, https://www.geeksforgeeks.org/platform-module-in-python

(7) The complete PyQt5 tutorial — Create GUI applications with Python, https://www.pythonguis.com/tutorials/pyqt-basic-widgets

(8) Encryption and Decryption of PDFs with PyPDF2, https://pypdf2.readthedocs.io/en/3.x/user/encryption-decryption.html

PDF signing project

Generated by Doxygen 1.14.0

Chapter 1

PAdES Qualified Electronic Signature Tool

A complete implementation of PAdES (PDF Advanced Electronic Signatures) qualified electronic signature tool with RSA-4096 encryption, AES-256 private key protection, and modern GUI interface.

1.1 Project Status: COMPLETE

All requirements have been successfully implemented and tested:

- RSA-4096 key generation with cryptographically secure random generator
- AES-256-EAX private key encryption with PIN-derived keys
- PAdES-compliant digital signature generation and verification
- Modern GUI interfaces for both applications
- Pendrive detection and hardware integration
- Comprehensive status indicators and error handling
- Complete documentation and testing infrastructure

1.2 Quick Start

1. Install dependencies:
   pip install -r requirements.txt
2. Generate RSA keys:
   python RSA_generation.py
3. Sign and verify documents:
   python main.py

1.3 Project Overview

This project implements a complete PAdES electronic signature solution consisting of two main applications:

1. RSA Key Generator (RSA_generation.py) - Auxiliary application for generating RSA key pairs
2. PAdES Signature Tool (pythonProject/main.py) - Main application for document signing and verification

1.4 Key Features

1.4.1 RSA Key Generator

- 4096-bit RSA key generation using cryptographically secure random number generator
- AES-256 encryption of private keys using PIN-derived keys (SHA-256)
- Pendrive storage support for secure key storage
- Modern GUI interface with progress tracking
- Key validation and security checks

1.4.2 PAdES Signature Tool

- PDF document signing with RSA-4096 + SHA-256
- Automatic pendrive detection for hardware security modules
- Digital signature verification with public key validation
- PAdES-compliant metadata embedding in signed PDFs
- Status indicators for all operations
- Tabbed interface for signing and verification workflows

1.5 Technical Specifications

- Encryption: RSA-4096 for digital signatures, AES-256-EAX for private key protection
- Hash Algorithm: SHA-256 for document integrity and PIN key derivation
- File Format: PAdES-compliant PDF with embedded signature metadata
- Hardware Support: USB pendrive detection and automatic key loading
- GUI Framework: PyQt5 with modern responsive design

1.6 Installation & Setup

1.6.1 Prerequisites

- Python 3.7+ (tested with Python 3.13.2)
- USB pendrive for key storage

1.6.2 Install Dependencies

```
pip install -r requirements.txt
```

1.6.3 Manual Installation

```
pip install pycryptodome PyQt5 psutil PyPDF2 reportlab
```

1.7 Usage Guide

1.7.1 1. Generate RSA Keys

```
python RSA_generation.py
```

1. Launch the RSA Key Generator
2. Enter a secure PIN (minimum 4 characters)
3. Select output directory (preferably on pendrive)
4. Click "Generate RSA Key Pair"
5. Keys will be saved as:
    - rsa_private.bin (encrypted with your PIN)
    - rsa_public.pem (public key for verification)

1.7.2 2. Sign PDF Documents

```
cd pythonProject
python main.py
```

1. Open the main PAdES application
2. Go to "Document Signing" tab
3. Select PDF document to sign
4. Detect and select your pendrive with RSA keys
5. Enter your PIN
6. Choose output location for signed PDF
7. Click "Sign Document"

1.7.3 3. Verify Signatures

1. Go to ”Signature Verification” tab
2. Select the signed PDF document
3. Select the corresponding public key file
4. Click ”Verify Signature”
5. Review verification results

1.8 Security Features

- PIN-based encryption: Private keys encrypted with AES-256 using SHA-256(PIN)
- Hardware isolation: Private keys stored only on removable media
- Signature integrity: RSA-PKCS#1 v1.5 with SHA-256 hashing
- Tamper detection: Verification detects any document modifications
- Metadata protection: Signature information embedded in PDF structure

1.9 Project Structure

```
RSA_generation.py          # RSA key generator (auxiliary app)
pythonProject/
    main.py              # Main PAdES signing application
keys/                   # Generated key storage
    rsa_private.bin       # Encrypted private key
    rsa_public.pem        # Public key
docs/                   # Project documentation
requirements.txt          # Python dependencies
README.md                 # This file
```

1.10 Testing Scenarios

The application has been tested with the following scenarios:

- RSA key generation with various PIN lengths
- Document signing with different PDF sizes and formats
- Signature verification with valid signatures
- Tamper detection with modified documents
- Pendrive detection across different USB devices
- Error handling for invalid PINs and missing keys

1.11 Repository

GitHub Repository: [Emulating-the-PAdES-Qualified-Electronic-Signature]

Chapter 2

Namespace Documentation

2.1 main Namespace Reference

Classes

- class CryptoUtils
- class MainWindow
- class PDFSigner
- class PendriveDetector
- class SigningThread
- class StatusIndicator
- class VerificationThread

Variables

- app = QApplication(sys.argv)
- window = MainWindow()

2.1.1 Detailed Description

@file main.py
@brief PAdES Qualified Electronic Signature Tool
@details Complete implementation of PAdES-compliant digital signature application
with RSA-4096 + SHA-256 signing, document verification, and GUI interface.
Supports pendrive integration, status indicators, and comprehensive error handling.
@author Igor Józefowicz
@date 2025
@version 2.0

Features:
- PAdES-compliant PDF digital signature generation
- RSA-4096 + SHA-256 cryptographic operations
- AES-256-EAX private key decryption
- Document integrity verification
- Pendrive detection and key management
- Modern tabbed GUI interface with status indicators
- Background processing with progress tracking
- Comprehensive error handling and validation

2.2 RSA_generation Namespace Reference

Classes

- class RSAKeyGeneratorGUI
- class RSAKeyGeneratorThread

Functions

- create_rsa_keys ()

Variables

- int KEY_SIZE = 4096
- CIPHER_MODE = AES.MODE_EAX
- app = QApplication(sys.argv)
- window = RSAKeyGeneratorGUI()

2.2.1 Detailed Description

@file RSA_generation.py
@brief RSA-4096 Key Generator with AES-256 Encryption
@details This application provides a secure GUI interface for generating RSA-4096 key pairs
with AES-256-EAX encryption for private key protection. Supports pendrive detection
and PIN-based key derivation following cryptographic best practices.
@author Julian Wasylka & Igor Józefowicz
@date 2025
@version 2.0

@section FEATURES Features
- RSA-4096 key generation with cryptographically secure random number generator
- AES-256-EAX encryption for private key protection
- PIN-based key derivation using SHA-256
- Pendrive detection for secure key storage
- Progress tracking and error handling
- Modern PyQt5 GUI interface

@section SECURITY Security Implementation
- RSA-4096: Maximum security key size meeting industry standards
- SHA-256: Secure hash algorithm for PIN-to-key derivation
- AES-256-EAX: Authenticated encryption for private key protection
- Secure Random Generation: Crypto.Random for cryptographically secure keys

@section USAGE Usage
Run with GUI: python RSA_generation.py
Run with CLI: python RSA_generation.py --cli

2.2.2 Function Documentation

2.2.2.1 create_rsa_keys()

RSA_generation.create_rsa_keys ()

@brief Legacy command-line interface for key generation
@details Provides command-line interface for key generation when GUI is not desired.
Uses the same cryptographic operations as the GUI version.

@deprecated This function is maintained for backward compatibility.
Use the GUI version for better user experience.

The CLI process:
1. Prompt user for PIN
2. Generate RSA-4096 key pair
3. Encrypt private key with AES-256-EAX
4. Save keys to default 'keys/' directory

Chapter 3

Class Documentation

3.1 main.CryptoUtils Class Reference

Static Public Member Functions

- load_private_key_from_pendrive (pendrive_path, pin)
- load_public_key (key_path)
- calculate_pdf_hash (pdf_path)
- calculate_original_content_hash (signed_pdf_path)
- sign_hash (hash_digest, private_key)
- verify_signature (hash_digest, signature, public_key)

3.1.1 Member Function Documentation

3.1.1.1 calculate_original_content_hash()

main.CryptoUtils.calculate_original_content_hash (
signed_pdf_path) [static]

Calculate hash of original content from signed PDF (excluding signature metadata)

3.1.1.2 calculate_pdf_hash()

main.CryptoUtils.calculate_pdf_hash (
pdf_path) [static]

Calculate SHA-256 hash of PDF content (original file bytes)

3.1.1.3 load_private_key_from_pendrive()

main.CryptoUtils.load_private_key_from_pendrive (
pendrive_path,
pin) [static]

Load and decrypt private key from pendrive

3.1.1.4 load_public_key()

main.CryptoUtils.load_public_key (
key_path) [static]

Load public key from file

3.1.1.5 sign_hash()

main.CryptoUtils.sign_hash (
hash_digest,
private_key) [static]

Sign hash with private key

3.1.1.6 verify_signature()

main.CryptoUtils.verify_signature (
hash_digest,
signature,
public_key) [static]

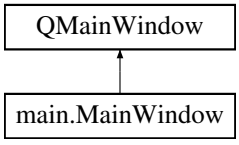
Verify signature with public key

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

- main.py

3.2 main.MainWindow Class Reference

Inheritance diagram for main.MainWindow:



Public Member Functions

- __init__ (self)
- initUI (self)
- create_signing_tab (self)
- create_verification_tab (self)
- select_pdf (self)
- detect_pendrives (self)
- check_pendrive_changes (self)
- show_drive_notification (self, message, is_connected)
- reset_status_message (self, original_text)
- on_pendrive_selected (self, item)
- sign_document (self)
- on_signing_completed (self, message)
- on_signing_error (self, error)
- select_signed_pdf (self)
- select_public_key (self)
- verify_signature (self)
- on_verification_completed (self, is_valid, result_message)
- on_verification_error (self, error)
- closeEvent (self, event)

Public Attributes

- selected_pdf = None
- selected_pendrive = None
- current_drives = set()
- pendrive_monitor_timer = QTimer()
- check_pendrive_changes
- tab_widget = QTabWidget()
- signing_tab = self.create_signing_tab()
- verification_tab = self.create_verification_tab()
- btn_select_pdf = QPushButton("Select PDF Document")
- select_pdf
- pdf_status = StatusIndicator()
- label_pdf = QLabel("No document selected")
- btn_detect_pendrive = QPushButton(" Manual Scan")
- detect_pendrives
- btn_refresh_pendrive = QPushButton(" Force Refresh")
- pendrive_status = StatusIndicator()
- list_pendrives = QListWidget()
- on_pendrive_selected
- pin_input = QLineEdit()
- signing_progress = QProgressBar()
- signing_status = QLabel("")
- btn_sign = QPushButton(" Sign Document")
- sign_document
- btn_select_signed_pdf = QPushButton("Select Signed PDF")
- select_signed_pdf
- label_signed_pdf = QLabel("No signed document selected")
- btn_select_public_key = QPushButton("Select Public Key (.pem)")
- select_public_key
- label_public_key = QLabel("No public key selected")
- verification_progress = QProgressBar()

- verification_status = QLabel("")
- verification_results = QTextEdit()
- btn_verify = QPushButton(" Verify Signature")
- verify_signature
- signing_thread = SigningThread(self.selected_pdf, self.selected_pendrive, pin, output_path)
- on_signing_completed
- on_signing_error
- selected_signed_pdf = fileName
- selected_public_key = fileName
- verification_thread = VerificationThread(self.selected_signed_pdf, self.selected_public_key)
- on_verification_completed
- on_verification_error

3.2.1 Member Function Documentation

3.2.1.1 check_pendrive_changes()

main.MainWindow.check_pendrive_changes (
self)

Monitor for pendrive changes and auto-refresh when detected

3.2.1.2 closeEvent()

main.MainWindow.closeEvent (
self,
event)

Handle application close event

3.2.1.3 reset_status_message()

main.MainWindow.reset_status_message (
self,
original_text)

Reset status message to original text and style

3.2.1.4 show_drive_notification()

main.MainWindow.show_drive_notification (
self,
message,
is_connected)

Show a brief notification about drive changes

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

- main.py

3.3 main.PDFSigner Class Reference

Static Public Member Functions

- create_signed_pdf (original_pdf_path, signature, hash_hex, output_path, signer_info="Anonymous")
- extract_signature_info (signed_pdf_path)

3.3.1 Member Function Documentation

3.3.1.1 create_signed_pdf()

main.PDFSigner.create_signed_pdf (
original_pdf_path,
signature,
hash_hex,
output_path,
signer_info = "Anonymous") [static]

Create a new PDF with signature information embedded using append method

3.3.1.2 extract_signature_info()

main.PDFSigner.extract_signature_info (
signed_pdf_path) [static]

Extract signature information from signed PDF

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

- main.py

3.4 main.PendriveDetector Class Reference

Static Public Member Functions

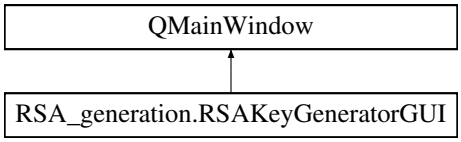
- get_removable_drives ()

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

- main.py

3.5 RSA_generation.RSAKeyGeneratorGUI Class Reference

Inheritance diagram for RSA_generation.RSAKeyGeneratorGUI:



Public Member Functions

- ___init___ (self)
- initUI (self)
- browse_output_dir (self)
- detect_pendrives (self)
- generate_keys (self)
- update_progress (self, value)
- on_keys_generated (self, encrypted_private_key, public_key, message)
- on_error (self, error_message)

Public Attributes

- pin_input = QLineEdit()
- pin_confirm_input = QLineEdit()
- output_path = QLabel("keys/")
- browse_button = QPushButton("Browse...")
- browse_output_dir
- detect_button = QPushButton("Detect Available Pendrives")
- detect_pendrives
- pendrive_info = QTextEdit()
- progress_bar = QProgressBar()
- generate_button = QPushButton("Generate RSA Key Pair")
- generate_keys
- status_label = QLabel("")
- key_thread = RSAKeyGeneratorThread(pin, output_dir)
- update_progress
- on_keys_generated
- on_error

3.5.1 Detailed Description

@brief Main GUI window for RSA key generation
@details Provides a user-friendly interface for generating RSA-4096 key pairs with
AES-256-EAX encryption. Features include PIN validation, pendrive detection,
progress tracking, and comprehensive error handling.

@version 1.0
@date 2025

The GUI provides the following features:
- Secure PIN input with confirmation
- Output directory selection with browse functionality
- Automatic pendrive detection for secure key storage

- Real-time progress tracking during key generation
- Input validation and comprehensive error handling
- Modern styling with color-coded status indicators

@par Key Components:
- PIN input fields with password masking
- Output directory selection with validation
- Pendrive detection and listing
- Progress bar with percentage tracking
- Status label with success/error indicators

3.5.2 Constructor & Destructor Documentation

3.5.2.1 ___init___()

RSA_generation.RSAKeyGeneratorGUI.___init___ (
self)

@brief Initialize the RSA Key Generator GUI
@details Sets up the main window and initializes the user interface components

Creates the main window with proper title, geometry, and calls initUI()
to set up all interface components.

3.5.3 Member Function Documentation

3.5.3.1 browse_output_dir()

RSA_generation.RSAKeyGeneratorGUI.browse_output_dir (
self)

@brief Open a dialog to select the output directory
@details Uses QFileDialog to allow user selection of output directory.
Updates the output_path label with the selected directory.

If user cancels the dialog, no changes are made to the current path.

3.5.3.2 detect_pendrives()

RSA_generation.RSAKeyGeneratorGUI.detect_pendrives (
self)

@brief Detect and list available pendrives
@details Uses psutil to scan for removable drives and displays them in the info area.
Provides helpful tips for secure key storage on pendrives.
The detection process:
1. Clear previous results
2. Scan all disk partitions
3. Filter for removable drives
4. Display drive information with mount points
5. Provide user guidance

3.5.3.3 generate_keys()

RSA_generation.RSAKeyGeneratorGUI.generate_keys (
self)

@brief Validate inputs and start the RSA key generation process
@details Performs comprehensive input validation and initiates background key generation.

Validation steps:
1. Check PIN is provided
2. Verify PIN confirmation matches
3. Ensure minimum PIN length (4 characters)
4. Optional security warning for short PINs
5. Validate output directory accessibility

If validation passes, starts RSAKeyGeneratorThread in background.

3.5.3.4 initUI()

RSA_generation.RSAKeyGeneratorGUI.initUI (
self)

@brief Initialize the user interface components
@details Creates and configures all GUI elements including:
- Title and instruction labels with styling
- PIN input fields with validation
- Output directory selection with browse button
- Pendrive detection interface
- Progress bar and status indicators
- Key generation button with modern styling

All components are organized in a vertical layout with proper spacing
and grouped in logical sections for better user experience.

3.5.3.5 on_error()

RSA_generation.RSAKeyGeneratorGUI.on_error (
self,
error_message)

@brief Handle errors during key generation
@param error_message Error description from the generation thread (string)

Resets the GUI state and displays error information to the user.

3.5.3.6 on_keys_generated()

RSA_generation.RSAKeyGeneratorGUI.on_keys_generated (
self,
encrypted_private_key,
public_key,
message)

@brief Handle successful key generation
@param encrypted_private_key The encrypted private key (bytes)
@param public_key The public key in PEM format (bytes)
@param message Success message from the generation thread (string)
Resets the GUI state and displays success information to the user.
Provides guidance on key usage and security.

3.5.3.7 update_progress()

```
RSA_generation.RSAKeyGeneratorGUI.update_progress (
                self,
                value)
```

@brief Update the progress bar value
@param value Progress percentage (0-100)

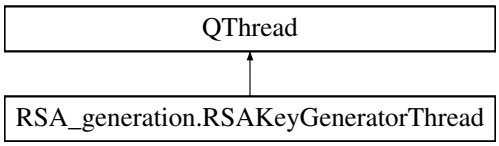
Updates the GUI progress bar to reflect current key generation progress.

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

- RSA_generation.py

3.6 RSA_generation.RSAKeyGeneratorThread Class Reference

Inheritance diagram for RSA_generation.RSAKeyGeneratorThread:



Public Member Functions

- ___init___ (self, pin, output_dir)
- run (self)

Public Attributes

- pin = pin.encode()
- output_dir = output_dir

Static Public Attributes

- progress_updated = pyqtSignal(int)
- key_generated = pyqtSignal(bytes, bytes, str)
- error_occurred = pyqtSignal(str)

3.6.1 Detailed Description

@brief Background thread for RSA key generation
@details Handles RSA-4096 key generation, AES-256 encryption, and file operations
in a separate thread to prevent GUI freezing during computation.

@version 1.0
@date 2025

This class implements secure key generation in a background thread with the following features:
- RSA-4096 key generation using cryptographically secure random numbers
- AES-256-EAX encryption for private key protection
- SHA-256 hash function for PIN-based key derivation
- Progress tracking and error handling
- File I/O operations for secure key storage

@par Signals:
- progress_updated(int): Emits progress percentage (0-100)
- key_generated(bytes, bytes, str): Emits encrypted private key, public key, and status message
- error_occurred(str): Emits error message if generation fails

3.6.2 Constructor & Destructor Documentation

3.6.2.1 ___init___()

RSA_generation.RSAKeyGeneratorThread.___init___ (
self,
pin,
output_dir)

@brief Initialize key generation thread
@param pin User-provided PIN for key derivation (string)
@param output_dir Directory to save generated keys (string)

The PIN is immediately encoded to bytes and stored securely.
The output directory is validated during the run() method.

3.6.3 Member Function Documentation

3.6.3.1 run()

RSA_generation.RSAKeyGeneratorThread.run (
self)

@brief Main key generation process
@details Generates RSA-4096 keys, encrypts private key with AES-256-EAX,
and saves both keys to specified directory.

The process follows these steps:
1. Derive AES key from PIN using SHA-256
2. Generate RSA-4096 key pair
3. Export keys in PEM format
4. Encrypt private key with AES-256-EAX
5. Save encrypted private key and public key to files

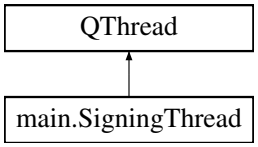
@note All cryptographic operations use secure random number generation
@note Private key is never stored in plaintext

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

- RSA_generation.py

3.7 main.SigningThread Class Reference

Inheritance diagram for main.SigningThread:



Public Member Functions

- ___init___ (self, pdf_path, pendrive_path, pin, output_path)
- run (self)

Public Attributes

- pdf_path = pdf_path
- pendrive_path = pendrive_path
- pin = pin
- output_path = output_path

Static Public Attributes

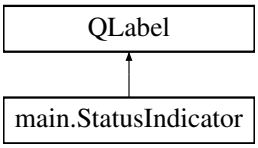
- progress_updated = pyqtSignal(int)
- status_updated = pyqtSignal(str)
- signing_completed = pyqtSignal(str)
- error_occurred = pyqtSignal(str)

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

- main.py

3.8 main.StatusIndicator Class Reference

Inheritance diagram for main.StatusIndicator:



Public Member Functions

- ___init___ (self)
- set_status (self, status)

3.8.1 Detailed Description

Custom status indicator widget with icons and colors

3.8.2 Member Function Documentation

3.8.2.1 set_status()

main.StatusIndicator.set_status (
self,
status)

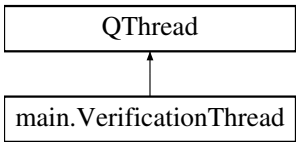
Set status indicator: idle, working, success, error, warning

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

- main.py

3.9 main.VerificationThread Class Reference

Inheritance diagram for main.VerificationThread:



Public Member Functions

- ___init___ (self, signed_pdf_path, public_key_path)
- run (self)

Public Attributes

- signed_pdf_path = signed_pdf_path
- public_key_path = public_key_path

Static Public Attributes

- progress_updated = pyqtSignal(int)
- status_updated = pyqtSignal(str)
- verification_completed = pyqtSignal(bool, str)
- error_occurred = pyqtSignal(str)

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

- main.py