Security of Computer Systems

Project Report

Authors:
Franciszek Gwarek, 193192
Maciej Górlaczyk, 193302

Version: 2.0

Versions

Version	Date	Description of changes
1.0	14.04.2025	Creation of the document and the control term section
2.0	09.06.2025	Final term section

1. Project – control term

1.1 Description

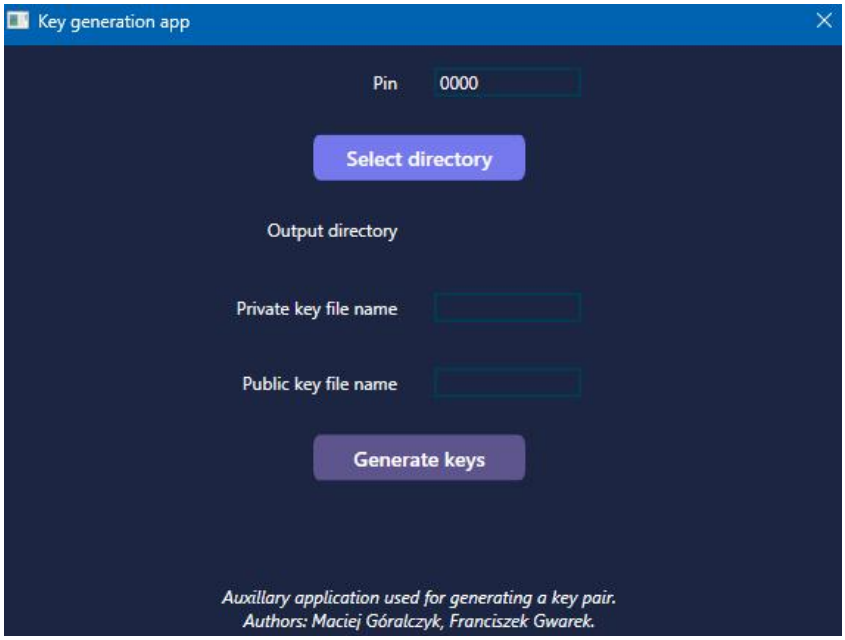
The primary goal of the first part of the project is to design and develop a supporting application that generates an RSA key pair and secures the private key using the AES algorithm, with the encryption key derived from the user's PIN. Additionally, this part includes the initial design of an application intended for creating qualified electronic signatures based on the PAdES standard concept.

1.2 Results

GitHub repository: https://github.com/Manie-K/PDF-signer

Technologies used: Both applications were written in C# using WPF and .NET. The standard *System.Security.Cryptography* library was used for generating keys and encrypting the private key.

Application for generating keys: The user provides the file save location, file names, and a PIN.
The application then generates an RSA key pair (public and private keys) using a 4096-bit key and a cryptographically secure pseudorandom number generator.
Next, the private key is encrypted using the AES algorithm with a 256-bit key derived from the hash of the user's PIN.
Both the public and private keys are saved in .key format at the location specified by the user.



Application for making qualified electronic signature: Only a basic framework is implemented.

1.3 Summary

The additional application for key generation is fully implemented and functions as intended. An initial concept and a basic framework of the main application have also been developed.

2. Project – Final term

2.1 Description

The objective of the second part of the project is to design and implement an application that emulates a qualified electronic signature in PDF documents, in accordance with the PAdES (PDF Advanced Electronic Signature) standard.

Additionally, this stage includes the development of a supporting tool for generating RSA key pairs, as well as the creation of documentation using Doxygen.

2.2 Code Description

Both applications were written in C# using WPF and .NET. The standard *System.Security.Cryptography* library was used for generating keys, hash and signature, encrypting the private key and verifying the signature.

```
using (RSA rsa = RSA.Create(4096))
{
    byte[] privateKeyBytes = rsa.ExportPkcs8PrivateKey();
    byte[] publicKeyBytes = rsa.ExportSubjectPublicKeyInfo();

    byte[] encryptedPrivateKeyBytes = EncryptPrivateKey(privateKeyBytes);
    // ...
}
```

Key pair generation

```
using (Aes aes = Aes.Create())
{
    var pbkdf2 = new Rfc2898DeriveBytes(pin, salt, 100_000,
HashAlgorithmName.SHA256);
    var key = pbkdf2.GetBytes(aes.KeySize / 8);

    aes.Key = key;
    aes.IV = iv;

    using (var encryptor = aes.CreateEncryptor())
    {
      byte[] cipherBytes = encryptor.TransformFinalBlock(privateKeyBytes, 0,
privateKeyBytes.Length);

      // ...
    }
}
```

Private key encryption

```
  byte[] hash;
  using (SHA256 sha256 = SHA256.Create())
  {
      hash = sha256.ComputeHash(pdfBytes);
   }
```

Hash generation

```
RSA rsa = RSA.Create();
rsa.ImportPkcs8PrivateKey(decryptedPrivateKeyBytes, out _);
byte[]     signature     =     rsa.SignHash(hash,     HashAlgorithmName.SHA256,
RSASignaturePadding.Pkcs1);
```

Signature generation

```
using Aes aes = Aes.Create();

var pbkdf2 = new Rfc2898DeriveBytes(pin, salt, 100_000,
HashAlgorithmName.SHA256);
var key = pbkdf2.GetBytes(aes.KeySize / 8);
aes.Key = key;
aes.IV = iv;

using (var decryptor = aes.CreateDecryptor())
{
    decryptedPrivateKey = decryptor.TransformFinalBlock(encryptedPrivateKey, 0,
encryptedPrivateKey.Length);
}
```

Private key decryption

```
byte[] publicKeyBytes = File.ReadAllBytes(publicKeyPath);
RSA rsa = RSA.Create();
rsa.ImportSubjectPublicKeyInfo(publicKeyBytes, out _);

bool isValid = rsa.VerifyHash(hash, signatureBytes, HashAlgorithmName.SHA256,
RSASignaturePadding.Pkcs1);
```

Signature verification

2.3 Use cases

Using our application is very intuitive due to its simple graphical interface. Additionally, the user is informed about the application's status through clear messages.

Key generation:
To generate a key pair, enter a four-digit PIN, choose a folder where the keys will be saved, and provide filenames for the private and public key files.

PDF file signing:
To sign a document, select a PDF file, enter a four-digit PIN, and connect the device (USB flash drive) containing the private key file.
A signed PDF file will be generated, with the suffix _signed.pdf in its name.

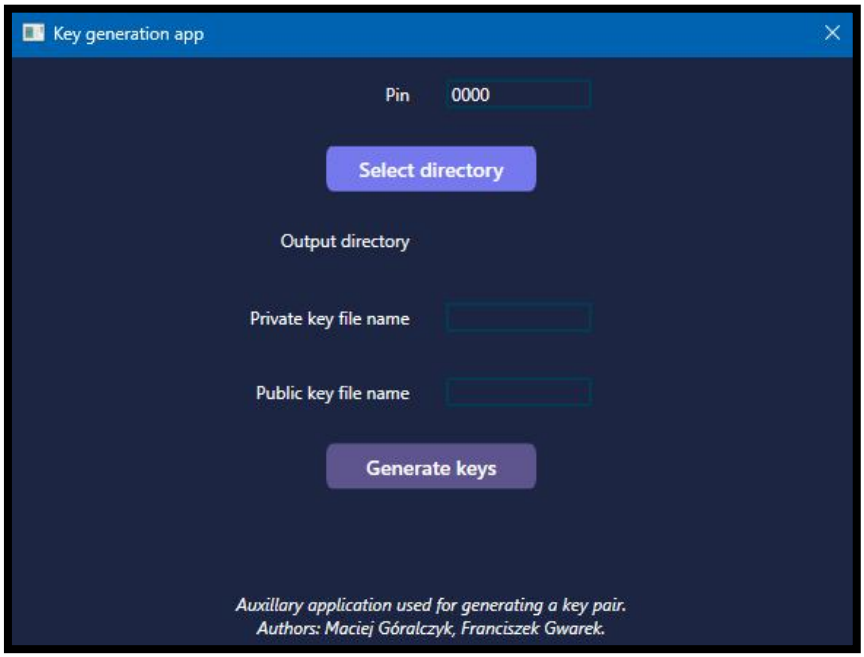
Signature verification:
First, select the previously signed PDF file and the file containing the public key. After pressing the button, information about the validity of the signature will be displayed.

2.4 Results

GitHub repository: https://github.com/Manie-K/PDF-signer

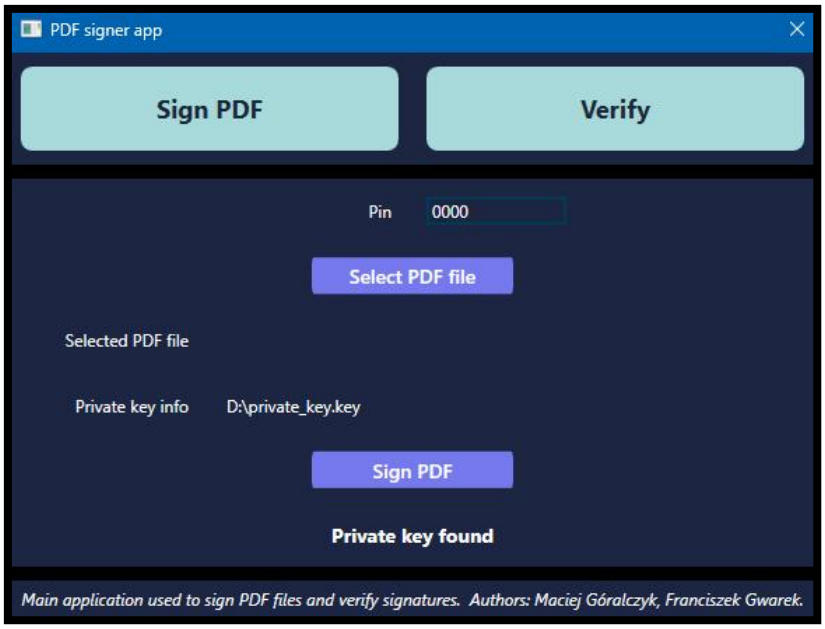
Tests:
Additionally, some unit test has been created.

Application for generating keys:
The user provides the file save location, file names, and a PIN.
The application then generates an RSA key pair (public and private keys) using a 4096-bit key and a cryptographically secure pseudorandom number generator.
Next, the private key is encrypted using the AES algorithm with a 256-bit key derived from the hash of the user's PIN.
Both the public and private keys are saved in .key format at the location specified by the user.



Key generation application

Application for generating and verifying qualified electronic signature:
Generating signature:
The user specifies the location of the PDF file and provides a PIN code.
The application automatically locates the private key stored on a connected USB drive.
Using the provided PIN, the private key is decrypted and a SHA-256 hash of the PDF file is generated.
This hash is then signed using the decrypted 4096-bit RSA private key.
The resulting digital signature is embedded in the PDF file, which is saved in the same location as the original document.



PDF signer application - Sign PDF tab

Verifying signature:

The user specifies the location of the signed PDF file and provides the corresponding public key.
The application extracts the digital signature from the PDF document.
It then generates a SHA-256 hash of the PDF content, excluding the signature.
Using the public key, the application decrypts the signature and compares it with the computed hash.
The result of the verification is displayed in the application window.

PDF signer application - Verify PDF tab

2.5 Summary

Both the primary and additional applications are fully implemented and functions as intended. The Doxygen documentation and some unit test has also been successfully created.

3. Literature

[1] https://enauczanie.pg.edu.pl/moodle/pluginfile.php/2465479/mod_resource/content/6/ENG_SCS_2025_project_v1.pdf

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

[3] https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography?view=net-9.0