LABORATORIUM BEZPIECZEŃSTWA SYSTEMÓW TELEINFORMATYCZNYCH	
Temat ćwiczenia: Podpis Cyfrowy	
Wykonał: Bartosz Gabruk 157546	**Data oddania:** 28.05.2025 r.

Wprowadzenie

Celem niniejszego projektu jest implementacja systemu podpisu cyfrowego w języku Python, opartego na asymetrycznym algorytmie RSA oraz funkcji skrótu SHA3-256. Projekt demonstruje proces generowania kluczy przy użyciu zewnętrznego źródła losowości (TRNG z pliku aes.bin), podpisywania plików tekstowych oraz weryfikacji ich integralności i autentyczności. Rozwiązanie sprawdza się w scenariuszach, gdzie wymagane jest zapewnienie nienaruszalności i potwierdzenie autorstwa dokumentów.

Kody źródłowe

Trng_rsa_get.py

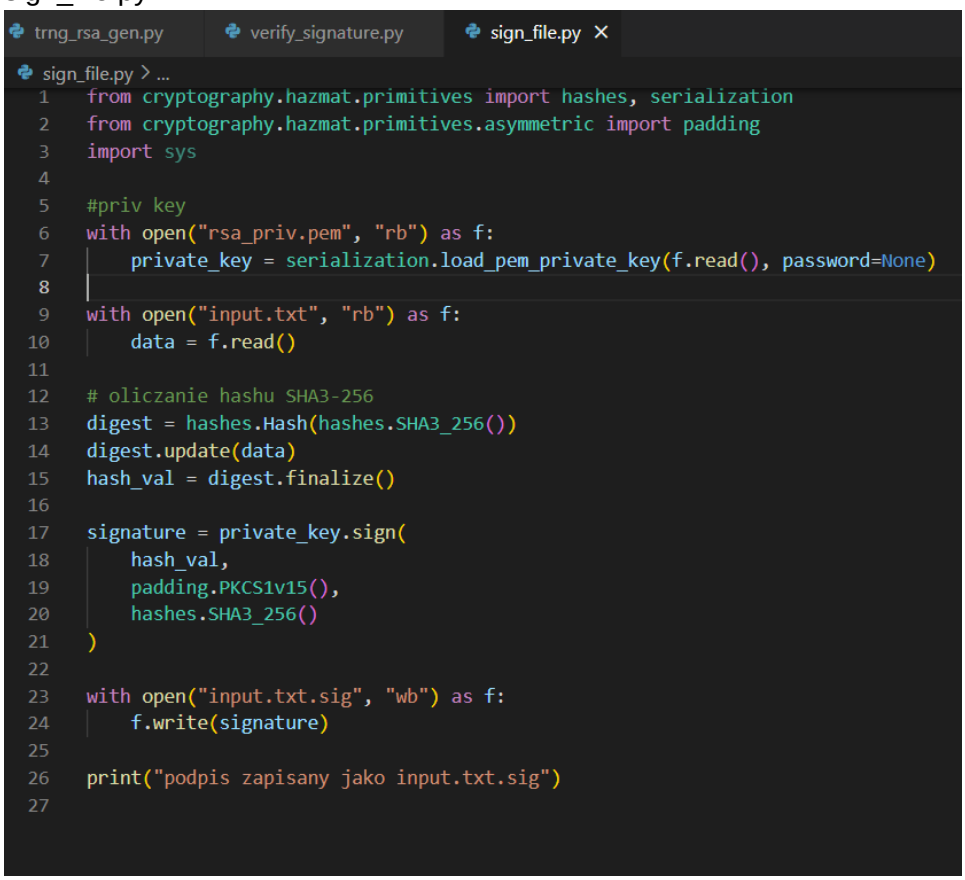
trng_rsa_gen.py | verify_signature.py | sign_file.py

trng_rsa_gen.py > ...

```python
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
import os
import random

with open("aes.bin", "rb") as f:
    trng_data = f.read()

random.seed(int.from_bytes(trng_data, "big"))

def custom_rng(n):
    return os.urandom(n)

#generowanie kluczy RSA
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)

public_key = private_key.public_key()

#zapis priv key
with open("rsa_priv.pem", "wb") as f:
    f.write(private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.TraditionalOpenSSL,
        encryption_algorithm=serialization.NoEncryption()
    ))

#zapis public key
with open("rsa_pub.pem", "wb") as f:
    f.write(public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    ))

print("Klucze RSA wygenerowane z TRNG poprawnie")

```

Verify_signature.py

sign_file.py



Kody źródłowe

Program rozpoczyna pracę od odczytania entropii z pliku aes.bin, co pozwala na deterministyczne zainicjowanie generatora losowości używanego przy tworzeniu pary kluczy RSA (2048 bitów). Wygenerowane klucze zapisywane są w formatach PEM jako rsa_priv.pem oraz rsa_pub.pem. Kolejnym krokiem jest wczytanie wybranego pliku tekstowego (input.txt), z którego obliczany jest skrót SHA3-256. Skrót ten jest następnie podpisywany kluczem prywatnym RSA, a wynikowy podpis zapisywany jest w pliku input.txt.sig. Podczas weryfikacji program ładuje klucz publiczny, ponownie oblicza hash z pliku, odszyfrowuje podpis i porównuje oba skróty, wyświetlając informację o poprawności podpisu lub wykryciu modyfikacji.