

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 x verify_signature.py sign_file.py
trng_rsa_gen.py > ...
1 from cryptography.hazmat.primitives.asymmetric import rsa
2 from cryptography.hazmat.primitives import serialization
3 import os
4 import random
5
6 with open("aes.bin", "rb") as f:
7     trng_data = f.read()
8
9 random.seed(int.from_bytes(trng_data, "big"))
10
11 def custom_rng(n):
12     return os.urandom(n)
13
14 #generowanie kluczy RSA
15 private_key = rsa.generate_private_key(
16     public_exponent=65537,
17     key_size=2048,
18 )
19
20 public_key = private_key.public_key()
21
22 #zapis priv key
23 with open("rsa_priv.pem", "wb") as f:
24     f.write(private_key.private_bytes(
25         encoding=serialization.Encoding.PEM,
26         format=serialization.PrivateFormat.TraditionalOpenSSL,
27         encryption_algorithm=serialization.NoEncryption()
28     ))
29
30 #zapis public key
31 with open("rsa_pub.pem", "wb") as f:
32     f.write(public_key.public_bytes(
33         encoding=serialization.Encoding.PEM,
34         format=serialization.PublicFormat.SubjectPublicKeyInfo
35     ))
36
37 print("Klucze RSA wygenerowane z TRNG poprawnie")
38
```

Verify_signature.py

```
trng_rsa_gen.py  verify_signature.py X  sign_file.py
verify_signature.py > ...
1  from cryptography.hazmat.primitives import hashes, serialization
2  from cryptography.hazmat.primitives.asymmetric import padding
3
4
5  with open("rsa_pub.pem", "rb") as f:
6      public_key = serialization.load_pem_public_key(f.read())
7
8
9  with open("input.txt", "rb") as f:
10     data = f.read()
11
12
13  with open("input.txt.sig", "rb") as f:
14     signature = f.read()
15
16  # obliczenie skrótu SHA3-256
17  digest = hashes.Hash(hashes.SHA3_256())
18  digest.update(data)
19  hash_val = digest.finalize()
20
21
22  try:
23      public_key.verify(
24          signature,
25          hash_val,
26          padding.PKCS1v15(),
27          hashes.SHA3_256()
28      )
29      print("Podpis poprawny")
30  except Exception:
31      print("Podpis niepoprawny!!!")
32
```

sign_file.py

```
trng_rsa_gen.py  verify_signature.py  sign_file.py X
sign_file.py > ...
1  from cryptography.hazmat.primitives import hashes, serialization
2  from cryptography.hazmat.primitives.asymmetric import padding
3  import sys
4
5  #priv key
6  with open("rsa_priv.pem", "rb") as f:
7      private_key = serialization.load_pem_private_key(f.read(), password=None)
8
9  with open("input.txt", "rb") as f:
10     data = f.read()
11
12  # obliczenie hashu SHA3-256
13  digest = hashes.Hash(hashes.SHA3_256())
14  digest.update(data)
15  hash_val = digest.finalize()
16
17  signature = private_key.sign(
18      hash_val,
19      padding.PKCS1v15(),
20      hashes.SHA3_256()
21  )
22
23  with open("input.txt.sig", "wb") as f:
24      f.write(signature)
25
26  print("podpis zapisany jako input.txt.sig")
27
```

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.