

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  import base64
4  import sys
5
6  if len(sys.argv) < 3:
7      print("Użycie: python verify_signature.py <plik_z_danymi> <plik_z_podpisem>")
8      sys.exit(1)
9
10 data_filename = sys.argv[1]
11 sig_filename = sys.argv[2]
12
13 with open("rsa_pub.pem", "rb") as f:
14     public_key = serialization.load_pem_public_key(f.read())
15
16 with open(data_filename, "rb") as f:
17     data = f.read()
18 print("Dane wejściowe:", data)
19
20 with open(sig_filename, "rb") as f:
21     signature = f.read()
22 print("Podpis (base64):", base64.b64encode(signature).decode())
23
24
25 # obliczanie skrótu SHA3-256
26 digest = hashes.Hash(hashes.SHA3_256())
27 digest.update(data)
28 hash_val = digest.finalize()
29 print("Hash SHA3-256:", hash_val.hex())
30
31
32 try:
33     public_key.verify(
34         signature,
35         hash_val,
36         padding.PKCS1v15(),
37         hashes.SHA3_256()
38     )
39     print("Podpis poprawny")
40 except Exception:
41     print("Podpis niepoprawny!!!")
42
```

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  import base64
5
6  if len(sys.argv) < 2:
7      print("Użycie: python sign_file.py <plik_do_podpisu>")
8      sys.exit(1)
9
10 input_filename = sys.argv[1]
11 sig_filename = input_filename + ".sig"
12
13 # priv key
14 with open("rsa_priv.pem", "rb") as f:
15     private_key = serialization.load_pem_private_key(f.read(), password=None)
16
17 with open(input_filename, "rb") as f:
18     data = f.read()
19 print("Dane wejściowe:", data)
20
21 # obliczanie hashu SHA3-256
22 digest = hashes.Hash(hashes.SHA3_256())
23 digest.update(data)
24 hash_val = digest.finalize()
25 print("Hash SHA3-256:", hash_val.hex())
26
27 signature = private_key.sign(
28     hash_val,
29     padding.PKCS1v15(),
30     hashes.SHA3_256()
31 )
32 print("Podpis (base64):", base64.b64encode(signature).decode())
33
34 with open(sig_filename, "wb") as f:
35     f.write(signature)
36
37 print(f"Podpis zapisany jako {sig_filename}")
38
```

Podsumowanie

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.