

## Laboratorium nr 1

# Wstęp do kryptografii w Python

Czas wykonania: **do laboratorium nr 4**

Liczba punktów: **3 + 2**

PRK: T-L-1

## 1. Opis laboratorium

Celem laboratorium jest zapoznanie się z podstawowymi operacjami kryptograficznymi w języku Python. W ramach laboratorium zapoznamy się z operacjami obliczania skrótu, szyfrowania symetrycznego i asymetrycznego oraz wykonywania podpisu cyfrowego. W ramach laboratorium są do wykonania 3 zadania. Na każde zadanie składa się przykładowy działający kod w Python, który trzeba zmodyfikować zgodnie z instrukcjami. Przykładowe fragmenty kodu zawierają większość potrzebnych instrukcji. Resztę można uzyskać z dokumentacji lub podpowiedzi środowiska programistycznego.

## 2. Zadania do wykonania

Poniższe zadania bazują na bibliotece/module Cryptodome. Instaluje się go z użyciem polecenia: `pip install pycryptodome`. Na komputerze w laboratorium moduł może być już zainstalowany.

### 1) Zadanie 1 (2 pkt na lab1) – Podstawowe operacje pomocnicze i funkcje skrótu

a. Wstaw swoje imię, nazwisko oraz numer grupy oraz uruchom poniższy skrypt:

```
from Crypto.Util.Padding import pad
import binascii

def encode_in_hex_with_padding(plaintext: str):
    # Zamieniamy tekst na bajty
    byte_data = plaintext.encode('utf-8')

    # Dodajemy padding PKCS7
    padded_data = pad(byte_data, 16)  # 16 bajtów to blok AES

    # Kodujemy na hex
    hex_data = binascii.hexlify(padded_data).decode('utf-8')

    # Formatowanie: każda linia 16 bajtów
    formatted_hex = [hex_data[i:i + 32] for i in range(0,
len(hex_data), 32)]

    # Wyświetlamy wynik
    for line in formatted_hex:
        print(line)

from Crypto.Hash import SHA256, SHA3_256, MD5
import json

def compute_hashes(plaintext: str):
    # Przekształcamy tekst na bajty
    byte_data = plaintext.encode('utf-8')
```

```

# Obliczamy skrót SHA-256
sha256_hash = SHA256.new(byte_data).hexdigest()

# Obliczamy skrót SHA3-256
sha3_256_hash = SHA3_256.new(byte_data).hexdigest()

# Obliczamy skrót MD5
md5_hash = MD5.new(byte_data).hexdigest()

# Zapisujemy skróty do jednego json
package = {
    "message": plaintext,
    "hash1": sha256_hash,
    "hash2": md5_hash,
    "hash3": sha3_256_hash
}
# Konwersja json do tablicy bajtów
package_json = json.dumps(package).encode("utf-8")

# Wyświetlamy wyniki
print("\nSkróty:")
print(f"SHA-256: {sha256_hash}")
print(f"SHA3-256: {sha3_256_hash}")
print(f"MD5: {md5_hash}")

def zad1():
    encode_in_hex_with_padding("Kryptologia 2025 - Tomasz Hyla")
    compute_hashes("Kryptologia 2025 - Tomasz Hyla")

```

- b. Zmodyfikuj skrypt wg następujących wytycznych:
- za pomocą funkcji RIPEMD160 oblicz skrót z json i go wyświetl,
  - zmierz czas obliczania 1 miliona skrótów (bez ich wyświetlania), wykorzystaj do tego następujący fragment kodu:

```

import time
start = time.perf_counter()
end = time.perf_counter()
print(f"Czas: {end - start:.6f} s")

```

## 2) Zadanie 2 (1 pkt) – Szyfrowanie symetryczne z użyciem AES

- a. Wstaw swoje imię, nazwisko, numer grupy oraz uruchom poniższe skrypty:

```

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# 1. Bezpieczne generowanie klucza
def generate_secure_key():
    return get_random_bytes(16)

# 2. Funkcja szyfrująca
def encrypt_aes(plaintext: str, key: bytes) -> str:
    cipher = AES.new(key, AES.MODE_CBC)
    ciphertext = cipher.encrypt(pad(plaintext.encode('utf-8'),
        AES.block_size))
    return (cipher.iv + ciphertext).hex()

# 3. Funkcja deszyfrująca
def decrypt_aes(hex_ciphertext: str, key: bytes) -> str:

```

```

    data = bytes.fromhex(hex_ciphertext)
    iv, ciphertext = data[:16], data[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return plaintext.decode('utf-8')

def zad2():
    key = generate_secure_key()
    plaintext = "Kryptologia laboratorium 2025 - <imię i nazwisko>"

    # Szyfrowanie
    ciphertext = encrypt_aes(plaintext, key)
    print(f"Zaszyfrowane: {ciphertext}")

    # Deszyfrowanie
    decrypted = decrypt_aes(ciphertext, key)
    print(f"Odszyfrowane: {decrypted}")

```

b. Zmodyfikuj skrypt wg następujących wytycznych:

- i. zmień rozmiar klucza na 32 bajty;
- ii. wyświetl tekst jawny po dodaniu paddingu i zakodowaniu do hex;
- iii. w skrypcie wektor inicjalizacyjny (IV) jest przechowywany w sposób jawny wraz z szyfrogramem, nie jest to dobra praktyka; zmodyfikuj skrypt tak aby IV i klucz były obliczane z skrótu z hasła podanego przez użytkownika z użyciem funkcji SHA3.

### 3) Zadanie 3 (2 pkt) – Szyfrowanie asymetryczne, podpis cyfrowy

a. Wstaw swoje imię, nazwisko oraz numer grupy oraz uruchom poniższe skrypty:

```

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Random import get_random_bytes

# Demo szyfrowania asymetrycznego
def zad3 szyfr_asym():
    # Generowanie klucza RSA
    key = RSA.generate(2048)

    # Eksport klucza publicznego i prywatnego
    private_key = key.export_key()
    public_key = key.publickey().export_key()

    print("Klucz publiczny:")
    print(public_key.decode())

    print("\nKlucz prywatny:")
    print(private_key.decode())

    # Tworzenie obiektu szyfrującego z klucza publicznego
    public_key_obj = RSA.import_key(public_key)
    cipher_rsa_encrypt = PKCS1_OAEP.new(public_key_obj)

    # Wiadomość do zaszyfrowania
    message = "To jest tajna wiadomość!".encode("utf-8")

    # Szyfrowanie
    encrypted_message = cipher_rsa_encrypt.encrypt(message)

    print("\nZaszyfrowana wiadomość (hex):")
    print(encrypted_message.hex())

```

```
# Tworzenie obiektu deszyfrującego z klucza prywatnego
private_key_obj = RSA.import_key(private_key)
cipher_rsa_decrypt = PKCS1_OAEP.new(private_key_obj)

# Deszyfrowanie
decrypted_message = cipher_rsa_decrypt.decrypt(encrypted_message)

print("\nOdszyfrowana wiadomość:")
print(decrypted_message.decode())

from Crypto.PublicKey import RSA
from Crypto.Signature import pss
from Crypto.Hash import SHA256

def zad3_podpis():
    # 1. Generowanie klucza RSA
    key = RSA.generate(2048)

    # 2. Eksport kluczy (opcjonalnie - można zapisać do pliku)
    private_key = key.export_key()
    public_key = key.publickey().export_key()

    print("Klucz publiczny:")
    print(public_key.decode())

    # 3. Wiadomość do podpisania
    message = "Tomasz Hyla 2025".encode("utf-8")

    # 4. Tworzenie skrótu wiadomości
    h = SHA256.new(message)

    # 5. Podpisywanie wiadomości kluczem prywatnym przy użyciu RSA PSS
    signer = pss.new(key)
    signature = signer.sign(h)

    print("\nPodpis (hex):")
    print(signature.hex())

    # 6. Weryfikacja podpisu kluczem publicznym
    public_key_obj = RSA.import_key(public_key)

    verifier = pss.new(public_key_obj)
    h = SHA256.new(message)

    try:
        verifier.verify(h, signature)
        print("\nPodpis jest poprawny!")
    except (ValueError, TypeError):
        print("\nPodpis nie jest poprawny!")
```

- b. Czy za pomocą przedstawionego szyfrowania asymetrycznego można zaszyfrować wiadomość o dowolnej długości? Poeksperymentuj z ciągami tekstowymi o różnej długości.
- c. Wykorzystaj oba poprzednie skrypty oraz skrypty z zadań 1 i 2 do demonstracji działania tzw. koperty cyfrowej tj.:
  - i. Alicia i Bob generują swoje pary kluczy asymetrycznych,
  - ii. Alicja podpisuje wiadomość własnym kluczem prywatnym,

- iii. Alicja szyfruje podpisaną wiadomość (wiadomość, podpis) losowym kluczem symetrycznym,
- iv. Alicja szyfruje klucz symetryczny kluczem publicznym Boba,
- v. Bob deszyfruje swoim kluczem prywatnym zaszyfrowany klucz symetryczny,
- vi. Bob deszyfruje kluczem symetrycznym wiadomość/podpis,
- vii. Bob sprawdza podpis.