Laporan Praktikum Kriptografi
Tugas 2: Algoritma Modern (RSA)

Universitas Teknologi Digital

Disusun Oleh :

1. Amelia Nazelina Kusmana – 20123028
2. Raisya Puspa Kencana – 20123007

PROGRAM STUDI S1 INFORMATIKA UNIVERSITAS TEKNOLOGI DIGITAL TAHUN AJARAN 2024/2025

1. Pendahuluan
   Pada praktikum ini dilakukan implementasi algoritma RSA (Rivest–Shamir–Adleman) sebagai bagian dari algoritma kriptografi modern.
   RSA merupakan algoritma kriptografi asimetris yang menggunakan dua kunci berbeda: public key untuk enkripsi dan private key untuk dekripsi.
   Tujuan dari tugas ini adalah memahami proses pembangkitan kunci RSA, enkripsi–dekripsi pesan, serta penerapan tanda tangan digital (digital signature) menggunakan library PyCryptodome.

   RSA bekerja berdasarkan prinsip faktorisasi bilangan prima besar. Proses utamanya:

   1. Key generation: menghasilkan pasangan kunci publik dan privat.
   2. Encryption: mengubah plaintext menjadi ciphertext dengan kunci publik.
   3. Decryption: mengembalikan ciphertext ke plaintext menggunakan kunci privat.

   Keterangan:

   - M: pesan asli
   - C: ciphertext
   - e: eksponen publik
   - d: eksponen privat
   - $n = p \times q$: hasil kali dua bilangan prima besar

```python
# -*- coding: utf-8 -*-
"""tugas2_modern.py

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1TddGSq6JkryXIxPKYA0jASvn1ZHwsaI9
"""

!pip install pycryptodome > /dev/null

from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Signature import pss
from Crypto.Hash import SHA256
import hashlib

def aes_encrypt(data):
    key = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_GCM)
    ciphertext, tag = cipher.encrypt_and_digest(data.encode())
    return key, cipher.nonce, ciphertext, tag

def aes_decrypt(key, nonce, ciphertext, tag):
    cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)
    return cipher.decrypt_and_verify(ciphertext, tag).decode()

# --- RSA ---
def gen_rsa_key():
    key = RSA.generate(2048)
    return key, key.publickey()

def rsa_encrypt(pub, data):
    cipher = PKCS1_OAEP.new(pub)
    return cipher.encrypt(data.encode())

def rsa_decrypt(priv, ciphertext):
    cipher = PKCS1_OAEP.new(priv)
    return cipher.decrypt(ciphertext).decode()

# --- SIGNATURE ---
def sign(priv, message):
    h = SHA256.new(message.encode())
    return pss.new(priv).sign(h)

def verify(pub, message, signature):
    h = SHA256.new(message.encode())
    try:
        pss.new(pub).verify(h, signature)
        return True
    except:
        return False

# --- HASH ---
def hash_all(msg):
    return {
        "MD5": hashlib.md5(msg.encode()).hexdigest(),
        "SHA1": hashlib.sha1(msg.encode()).hexdigest(),
        "SHA256": hashlib.sha256(msg.encode()).hexdigest()
    }

# ------ TESTING ------
msg = "Hello Cryptography"

# AES
key, nonce, ct, tag = aes_encrypt(msg)
print("AES decrypted:", aes_decrypt(key, nonce, ct, tag))

# RSA
priv, pub = gen_rsa_key()
cipher = rsa_encrypt(pub, msg)
print("RSA decrypted:", rsa_decrypt(priv, cipher))

# HASH
print("Hashes:", hash_all(msg))

# SIGNATURE
signature = sign(priv, msg)
print("Signature valid:", verify(pub, msg, signature))
```

2. Implementasi dan Langkah kerja

Implementasi RSA pada tugas ini dilakukan menggunakan Python library PyCryptodome dengan skema padding PKCS1_OAEP (untuk enkripsi) dan PSS (untuk digital signature).

A. Generate RSA Key Pair

Kode kamu membuat sepasang kunci RSA (privat & publik) dengan fungsi:

```
[3]
✓ 0s
# --- RSA ---
def gen_rsa_key():
    key = RSA.generate(2048)
    return key, key.publickey()
```

Penjelasan:

- RSA.generate(2048) menghasilkan kunci 2048-bit, yang merupakan ukuran aman secara modern.
- Fungsi mengembalikan dua nilai: priv (private key) dan pub (public key).

B. Enkripsi Pesan dengan RSA

Kode kamu mengenkripsi pesan string menggunakan *public key* dan padding OAEP:

```
def rsa_encrypt(pub, data):
    cipher = PKCS1_OAEP.new(pub)
    return cipher.encrypt(data.encode())
```

Penjelasan:

- PKCS1_OAEP.new(pub) membuat objek cipher RSA yang aman.
- data.encode() mengubah string menjadi byte sebelum dienkripsi.
- Hasilnya berupa ciphertext dalam bentuk byte.

C. Dekripsi Pesan dengan RSA

Dekripsi dilakukan menggunakan kunci privat (priv) agar bisa mengembalikan ciphertext ke bentuk aslinya:

```
def rsa_decrypt(priv, ciphertext):
    cipher = PKCS1_OAEP.new(priv)
    return cipher.decrypt(ciphertext).decode()
```

Penjelasan:

- cipher.decrypt(ciphertext) melakukan proses pembalikan enkripsi.
- .decode() mengubah hasil bytes kembali ke string.

Output console program

```
RSA decrypted: Hello Cryptography
```

D. Digital Signature (Tanda Tangan Digital)

Kamu menggunakan RSA dengan padding **PSS** (Probabilistic Signature Scheme) untuk proses tanda tangan digital.

```
[4]
✓ 0s
# --- SIGNATURE ---
def sign(priv, message):
    h = SHA256.new(message.encode())
    return pss.new(priv).sign(h)
```

Dan fungsi verifikasi

```
def verify(pub, message, signature):
    h = SHA256.new(message.encode())
    try:
        pss.new(pub).verify(h, signature)
        return True
    except:
        return False
```

Penjelasan:

- SHA256.new() membuat hash dari pesan.
- pss.new(priv).sign(h) menandatangani hash tersebut dengan private key.
- Fungsi verify() melakukan verifikasi tanda tangan memakai public key.
- Jika signature valid, fungsi mengembalikan True.

Output console program:

```
Signature valid: True
```

3. Hasil dan Pembahasan

Proses	Fungsi	Hasil
Key Generation	gen_rsa_key()	Kunci RSA 2048-bit berhasil dibuat
Enkripsi	rsa_encrypt(pub, msg)	Pesan terenkripsi (ciphertext biner)
Dekripsi	Rsa_decrypt(priv, msg)	Output: Hello Cryptography
Digital Signature	Sign(priv, msg)	Signature berhasil dibuat
Verifikasi	Verify(pub, msg, signature)	Signature valid “True”

Analisis hasil:

- RSA berhasil bekerja dengan sempurna untuk pesan teks.
- Karena RSA lambat untuk data besar, algoritma ini cocok untuk mengenkripsi kunci simetris (misalnya kunci AES).

- Penggunaan PKCS1_OAEP dan PSS menunjukkan implementasi modern yang lebih aman dari padding lama (v1.5).

4. Analisis Keamanan

   Kelebihan:
   - Keamanan tinggi karena berbasis faktorisasi bilangan prima besar.
   - Mendukung tanda tangan digital untuk autentikasi.
   - Library PyCryptodome sudah memakai skema padding modern (OAEP & PSS).

   Kelemahan:
   - Kecepatan enkripsi–dekripsi lebih lambat dibanding algoritma simetris.
   - Tidak efisien untuk file besar.
   - Jika private key bocor, seluruh sistem menjadi tidak aman.

   Rekomendasi:
   - Gunakan RSA hanya untuk enkripsi kunci simetris (*hybrid encryption*).
   - Panjang kunci minimal 2048 bit, direkomendasikan 3072 bit untuk keamanan jangka panjang.

Nama	Link Github
Raisya Puspa Kencana	https://github.com/RaisyaPuspa23
Amelia Nazelina Kusmana	https://github.com/Amelianazelina