Laporan Tugas 2 – Algoritma Modern

1. Pendahuluan

Tugas ini bertujuan untuk mengimplementasikan beberapa algoritma kriptografi modern, yaitu:

- Algoritma kunci simetris: AES
- Algoritma kunci asimetris: RSA
- Fungsi hash kriptografis: MD5, SHA-1, dan SHA-256
- Digital signature (tanda tangan digital): kombinasi RSA + fungsi hash

Implementasi dilakukan menggunakan bahasa pemrograman Python dengan library PyCryptodome sebagai core tools. Setiap algoritma diuji menggunakan pesan contoh yang sama sehingga mudah untuk membandingkan hasil dan memahami peran masing-masing komponen dalam sistem kriptografi modern.

2. Landasan Teori Singkat

2.1 Advanced Encryption Standard (AES)

AES adalah algoritma kriptografi kunci simetris berbasis blok dengan ukuran blok 128 bit dan panjang kunci yang umum digunakan 128, 192, atau 256 bit. AES bekerja dalam beberapa ronde transformasi (SubBytes, ShiftRows, MixColumns, AddRoundKey) yang membuat ciphertext sulit ditebak dari plaintext maupun kunci tanpa informasi rahasia.

Dalam tugas ini AES digunakan pada mode CBC (Cipher Block Chaining). Pada mode ini, setiap blok plaintext terlebih dahulu di-XOR dengan ciphertext blok sebelumnya (atau dengan IV untuk blok pertama), lalu dienkripsi. Mode ini menambah keamanan dan membuat ciphertext berbeda setiap kali walaupun plaintext dan kunci sama, asalkan IV berbeda.

2.2 RSA

RSA merupakan algoritma kunci publik yang bekerja berdasarkan kesulitan faktorisasi bilangan bulat besar. RSA menghasilkan sepasang kunci:

- Private key: digunakan untuk dekripsi atau penandatanganan dan harus dijaga rahasia.
- Public key: boleh disebar, digunakan untuk enkripsi atau verifikasi tanda tangan digital.

Dalam praktik modern, RSA biasanya digunakan untuk mengenkripsi data berukuran kecil (misalnya kunci simetris), dan untuk digital signature.

2.3 Fungsi Hash Kriptografis

Fungsi hash kriptografis memetakan data berukuran sembarang menjadi nilai hash dengan panjang tetap. Properti pentingnya adalah:

- One-way: sulit menemukan input dari hash.
- Collision resistant: sulit mencari dua input berbeda yang memiliki hash sama.
- Avalanche effect: perubahan kecil pada input mengubah hash secara drastis.

Dalam tugas ini digunakan tiga fungsi hash yang umum: MD5 (128 bit), SHA-1 (160 bit), dan SHA-256 (256 bit). MD5 dan SHA-1 sudah tidak direkomendasikan untuk aplikasi keamanan baru karena lemah terhadap collision, sedangkan SHA-256 masih banyak digunakan.

2.4 Digital Signature

Digital signature menggabungkan fungsi hash dan kriptografi kunci publik untuk menjamin:

- Keaslian (authentication) pengirim.
- Integritas (integrity) pesan.
- Non-repudiation: pengirim sulit menyangkal pernah mengirim pesan.

Skema umum digital signature adalah: pesan di-hash menjadi message digest, digest ditandatangani menggunakan private key (misalnya RSA), lalu penerima menghitung ulang hash pesan dan memverifikasi tanda tangan menggunakan public key.

3. Lingkungan dan Tools

Implementasi dilakukan dengan spesifikasi sebagai berikut:

- Bahasa pemrograman: Python.
- Library kriptografi: PyCryptodome (Crypto.Cipher, Crypto.PublicKey, Crypto.Signature, Crypto.Hash).
- Library tambahan: hashlib, base64.

4. Implementasi Program

4.1 Implementasi AES (Simetris)

Kode program AES berada pada berkas aes.py. Ringkasan implementasi:

1. Konstanta BLOCK_SIZE = 16 menandakan blok AES 128 bit. Fungsi pkcs7_pad dan pkcs7_unpad digunakan untuk menambah dan menghapus padding sesuai standar PKCS#7.
2. Enkripsi: program menghasilkan kunci acak 16 byte dan IV acak, lalu mengenkripsi plaintext dengan mode AES.MODE_CBC. Hasil ciphertext (IV + ciphertext) di-encode ke base64.

3. Dekripsi: program men-decode base64, memisahkan IV dan ciphertext, kemudian mendekripsi menggunakan kunci yang sama dan menghapus padding.

Pengujian dilakukan dengan pesan contoh "saya thia dan saya bangga menjadi fans emyu". Hasil dekripsi sama dengan plaintext awal sehingga implementasi dinyatakan benar.

4.2 Implementasi Fungsi Hash

Fungsi hash diimplementasikan dalam berkas hashFunction.py. Ringkasan:

4. Terdapat tiga fungsi: hash_md5, hash_sha1, dan hash_sha256 yang memanfaatkan modul hashlib dan mengembalikan nilai hash dalam bentuk heksadesimal.
5. Program utama menghitung dan menampilkan hash MD5, SHA-1, dan SHA-256 dari string yang sama dengan yang digunakan pada pengujian AES.

4.3 Implementasi RSA (Enkripsi & Dekripsi)

Implementasi RSA terdapat di berkas rsa.py. Program melakukan pembuatan kunci, enkripsi, dan dekripsi sebagai berikut:

6. Pembuatan kunci: menggunakan RSA.generate(2048) untuk membuat pasangan kunci RSA 2048 bit, kemudian diekspor ke format PEM.
7. Enkripsi: fungsi rsa_encrypt menggunakan skema PKCS1_OAEP dengan public key untuk mengenkripsi pesan biner.
8. Dekripsi: fungsi rsa_decrypt menggunakan private key untuk mengembalikan ciphertext menjadi plaintext semula.

Pengujian dilakukan dengan pesan "hello RSA". Hasil dekripsi terbukti sama dengan pesan awal.

4.4 Implementasi Digital Signature

Digital signature diimplementasikan dengan menggabungkan RSA dan SHA-256. Terdapat dua bentuk implementasi: fungsi sign/verify di rsa.py dan demo khusus di digitalSignature.py.

9. Pesan di-hash menggunakan SHA256.new(message) untuk menghasilkan message digest.
10. Digest ditandatangani menggunakan pkcs1_15.new(private_key).sign(hash_obj) untuk menghasilkan signature sepanjang 256 byte (sesuai kunci 2048 bit).
11. Verifikasi dilakukan dengan memanggil pkcs1_15.new(public_key).verify(hash_obj, signature). Jika tidak terjadi exception, tanda tangan dianggap valid.

Pengujian menunjukkan bahwa signature berhasil diverifikasi (Verified: True) ketika pesan tidak diubah dan public key yang benar digunakan.

5. Pengujian dan Analisis

5.1 Analisis AES

Hasil dekripsi AES selalu identik dengan plaintext awal selama kunci dan IV yang sama digunakan, sehingga fungsi enkripsi dan dekripsi dinyatakan benar. Penggunaan IV acak membuat ciphertext berbeda untuk setiap eksekusi sehingga lebih aman terhadap analisis pola.

Keterbatasan saat ini adalah manajemen kunci yang belum diatur dengan baik (kunci hanya disimpan di memori program) serta belum adanya antarmuka untuk mengenkripsi file.

5.2 Analisis Fungsi Hash

Tiga algoritma hash menghasilkan nilai yang berbeda untuk input yang sama. MD5 dan SHA-1 diketahui memiliki kelemahan collision sehingga tidak direkomendasikan untuk aplikasi yang membutuhkan keamanan tinggi, sedangkan SHA-256 masih aman dan cocok dipadukan dengan RSA dalam digital signature.

5.3 Analisis RSA

RSA 2048 bit menyediakan tingkat keamanan yang memadai untuk keperluan pembelajaran. Penggunaan skema OAEP menjadikan enkripsi tidak deterministik sehingga lebih kuat dibanding RSA tanpa padding. Dalam praktik nyata, RSA biasanya dipakai untuk mengenkripsi kunci simetris daripada data besar secara langsung.

5.4 Analisis Digital Signature

Kombinasi RSA dan SHA-256 terbukti mampu memberikan jaminan keaslian dan integritas pesan. Jika pesan diubah sedikit saja setelah ditandatangani, proses verifikasi akan gagal. Demikian pula jika verifikasi menggunakan public key yang berbeda, tanda tangan dinyatakan tidak valid.

6. Kesimpulan

Dari implementasi yang dilakukan dapat disimpulkan bahwa seluruh algoritma kriptografi modern yang diminta pada tugas ini (AES, RSA, fungsi hash, dan digital signature) berhasil diimplementasikan dan diuji dengan baik menggunakan Python dan PyCryptodome.

AES berhasil mengenkripsi dan mendekripsi pesan dengan benar pada mode CBC. Fungsi hash MD5, SHA-1, dan SHA-256 berhasil menghasilkan nilai hash yang berbeda untuk input yang sama. RSA 2048 bit bekerja dengan baik untuk enkripsi dan dekripsi, dan dapat digabung dengan SHA-256 untuk membentuk digital signature yang valid.

7. Saran Pengembangan

- Menambahkan antarmuka CLI atau GUI untuk memudahkan pengguna memasukkan pesan dan memilih algoritma.
- Membuat manajemen kunci yang lebih baik, misalnya menyimpan dan memuat kunci dari file terpisah.
- Menggabungkan AES dan RSA menjadi skema hybrid (RSA untuk mengenkripsi kunci AES).
- Menguji skema tanda tangan RSA-PSS yang lebih modern.