Implementasi Kriptografi Kunci Publik RSA untuk Enkripsi dan Dekripsi File Menggunakan Python

Laporan praktikum ini dibuat untuk memenuhi tugas mata kuliah Keamanan Data

Dosen
Sevi Nurafni, S.T., M.Si., M.Sc.

Kelompok 1 :

Betrand Athariq Delpiero	(2C2220001)
Ryan Fadhilah Faizal H	(2C2220007)
Teni Deinarosa Hermalia	(2C2220005)
Silvi Nurinsan	(2C2220006
Mulyati Eka Saputri	(2C2220009)
Intan Wahyuni Mangar	(2C2220013)
Herodia Mahwil	(2C2220015)

PRODI SAINS DATA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KOPERASI INDONESIA
2025

Tujuan Praktikum

Mengimplementasikan kriptografi kunci publik menggunakan algoritma RSA untuk melakukan enkripsi dan dekripsi pada file dengan extension (*.txt).

Pendahuluan

1.1. Algoritma Rivest Shamir Adleman (RSA)

Algoritma Rivest Shamir Adleman (RSA) adalah algoritma untuk enkripsi kunci publik (public-key encryption).Algoritma ini adalah algoritma pertama yang diketahui paling cocok untuk menandai (signing) dan untuk enkripsi (encryption) dan salah satu penemuan besar pertama dalam kriptografi kunci publik. RSA masih digunakan secara luas dalam protokol-protokol keamanan SSL/TLS, SET, SSH, S/MIME, PGP,DNSSEC (Frederico 2003).

Algoritma RSA dijabarkan pada tahun 1977 oleh Ron Rivest, Adi Shamir dan Len Adleman dari Massachusetts Institute of Technology (MIT). Nama RSA berasal dari inisial ketiganya (Rivest—Shamir— Adleman). Sebelumnya, pada tahun 1973, Clifford Cocks dari GCHQ Inggris telah menemukan sistem serupa, namun temuannya dirahasiakan hingga 1997 karena alasan keamanan nasional. RSA dipatenkan oleh MIT di AS pada tahun 1983 (U.S. Patent 4405829) dan berlaku hingga 21 September 2000. Karena publikasi dalam bentuk paten, sebagian besar negara lain tidak mengakuinya sebagai paten, sehingga temuan Cocks akhirnya dikenal secara luas.

1.2. Pembuatan Kunci Publik dan Kunci Privat

Algoritma kriptografi RSA didesain sesuai fungsinya sehingga menghasilkan kunci yang digunakan untuk enkripsi berbeda dari kunci yang digunakan untuk dekripsi pesan. Algoritma RSA disebut menggunakan kunci publik karena kunci enkripsi yang dibuat boleh diketahui semua orang, dan juga bisa melakukan enkripsi pesan tersebut. Sedangkan yang dimaksud kunci privat adalah kunci untuk melakukan dekripsi pesan tidak semua orang boleh mengetahuinya, hanya orang tertentu yang berhak saja untuk melakukan dekripsi pesan. Keamanan algoritma RSA didasarkan pada sulitnya memfaktorkan bilangan besar menjadi factor-faktor primanya (Lubis et al. 2013). Adapun langkah – langkah pembuatan kunci antara lain :

1. Pilih dua buah bilangan prima misal p dan q adapun $p \neq q$ dan secara acak dan terpisah untuk tiap-tiap p dan q. Hitung nilai N dimana nilai N adalah hasil perkalian antara bilangan p dan q, $N = p \cdot q$
2. Hitunglah nilai $(p - 1) \cdot (q - 1)$

3. Pilihlah bilangan bulat (integer) antara satu dan φ, (1 < e < φ) yang juga merupakan bilangan koprima dari φ.
4. Hitunglah nilai d hingga d e ≡ 1 (mod φ), adapun untuk mencari d bisa menggunakan rumus : $d = \frac{1+k\,N}{e}$

   Adapun nilai k adalah hasil percobaan nilai dari 1,2,3, … sehingga menghasilkan nilai d merupakan nilai bulat

   Adapun kunci publik antara lain :

1. Nilai bilangan N
2. Serta bilangan e (digunakan untuk proses enkripsi pesan)

   Adapun kunci privat antara lain :

1. Nilai bilangan N
2. Serta bilangan d (digunakan untuk proses dekripsi pesan)

1.3. Proses Enkripsi dan Dekripsi Pesan

Dalam melakukan proses enkripsi maupun dekripsi pesan algoritma RSA memiliki rumus yang berbeda dalam merubah pesan yang diterimanya. Adapun rumus yang digunakan untuk melakukan enkripsi pesan adalah :

$Ciphertext = Plaintext^{e}\ Mod\ N$

Keterangan :

Plaintext : Merupakan text asli yang akan dirubah

e : Merupakan kunci publik yang digunakan untuk enkripsi

N : Merupakan perkalian dua buah bilangan prima p dan q

Sedangkan untuk merubah ciphertext ke dalam bentuk semula memerlukan rumus yang berbeda dengan proses enkripsi, rumus dekripsi pesan adalah sebagai berikut:

$Plaintext = Ciphertext^{d}\ Mod\ N$

Keterangan :

Chipertext : Merupakan pesan yang akan diubah ke bentuk asli

d : Merupakan kunci privat yang digunakan untuk dekripsi

N : Merupakan perkalian dua buah bilangan prima p dan q

1.4. Contoh Proses Algoritma RSA

Diketahui telah dipilih dua buah bilangan prima yaitu p = 19 dan q = 41. Kemudian cari nilai N dengan mengkalikan kedua bilangan tersebut sehingga N = p.q = 19 . 41 = 779. Setelah itu carilah nilai φ dengan menggunakan rumus φ = (p - 1) . (q - 1) sehingga menghasilkan φ = (19 - 1) . (41 - 1) sehingga menghasilkan nilai φ = 720. Selanjutnya yang perlu dilakukan adalah mencari bilangan e dan d. Untuk mencari bilangan e dimana nilai e merupakan koprima dari nilai φ salah satu cara untuk mencari bilangan e yaitu menggunakan cara GCD. adapun hasil pencarian nilai e dapat dilihat pada tabel.

Contoh Pencarian Nilai e

Mulai dari	GCD (720,e)
e = 2	720 mod 2 = 0 GCD (2,720) = 2
e = 3	720 mod 3 = 0 GCD (3,720) = 3
e = 4	720 mod 4 = 0 GCD (4,720) = 4
e = 5	720 mod 5 = 0 GCD (5,720) = 5
e = 6	720 mod 6 = 0 GCD (6,720) = 6
e = 7	720 mod = 6 7 mod 6 = 1 6 mod 1 = 0 GCD(7,720) = 1

Untuk mencari nilai e adalah nilai yang menghasilkan GCD bernilai 1, untuk nilai e itu sendiri bisa dicoba dari nilai 2,3,4,… dst. Untuk mencari nilai d bisa menggunakan rumus $\frac{1+k\,N}{e}$. Adapun untuk contoh pencarian nilai d dapat dilihat pada tabel

Contoh Pencarian Nilai d

Nilai K	Persamaan $= d = \frac{1+k\,N}{e}$	Hasil
K = 1	$d = \frac{1+1\,.\,720}{7}$	103
K = 2	$d = \frac{1+2\,.\,720}{7}$	205.857142857142857
K = 3	$d = \frac{1+3\,.\,720}{7}$	308.71428571428571
…	…	…

Untuk nilai k pada pencarian nilai d bisa dilakukan percobaan dari 1,2,3, … dst hingga hasil dari persamaan tersebut menghasilkan bilangan bulat. Pada contoh ini nilai d didapat pada k = 1 dengan hasil 103. Kemudian misalkan pesan asli berupa “Tekno” akan diubah kedalam bentuk ASCII menjadi 87-101-107 110 yang kemudian akan dibagi menjadi P1 – P4 dan tiap Pi akan diubah menjadi chipertext menggunakan rumus sehingga dapat dilihat pada tabel

Contoh Proses Enkripsi

Pi	Enkripsi $Plaintext = Ciphertext^{103}\ Mod\ 779$
87	46
101	552
107	31
110	507

Sehingga pesan tersebut berubah menjadi 46-552-31-507. Untuk melakukan dekripsi pesan ciphertext yang didapatkan dipisah menjadi beberapa bagian dalam contoh ini dibagi menjadi 4 bagian C1 – C4 untuk proses dan hasil dekripsi pesan dapat dilihat pada tabel

Contoh Proses Dekripsi

Pi	Dekripsi $Ciphertext = Plaintext^{7}\ Mod\ 779$
46	87
552	101

31	107
507	110

Pada contoh tersebut didapatkan hasil 87-101-107-110 kemudian nilai tersebut diubah menjadi nilai ASCII sehingga menghasilkan teks berupa “Tekno”.

1.5. Kekuatan Kunci RSA

Keamanan algoritma RSA terletak pada kekuatan kunci yang dimiliki nya, hal ini disebabkan karena sulitnya memfaktorkan bilangan non prima menjadi bilangan prima yaitu r = p x q. Ketika r berhasil difaktorkan menjadi p dan q, maka $\Phi(r) = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya, karena kunci enkripsi PK diumumkan (tidak rahasia), maka kunci dekripsi SK dapat dihitung dari persamaan $PK \times SK \equiv 1 \pmod{\Phi(r)}$. Penemu algoritma RSA menyarankan nilai p dan q panjangnya lebih dari 100 digit. Dengan demikian hasil kali r = p x q akan berukuran lebih dari 200 digit. Menurut Rivest dan kawan-kawan, usaha untuk mencari faktor bilangan 200 digit membutuhkan waktu komputasi selama 4 milyar tahun (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik). Namun demikian berdasarkan paper yang berjudul Fault Based Attack of RSA Authentication yang ditulis oleh dan Andrea Pellegrini dengan rekan nya mereka mengklaim bahwa kunci private 1024 bit RSA telah mereka pecahkan dengan kurun waktu 104 jam. Hal ini telah membuat kunci privat 1024 bit tidak dirasa aman lagi. Namun demikian pihak RSA tetap menyatakan bahwa mereka tetap merekomendasikan 1024 bit untuk digunakan di berbagai perusahan dan juga menyarankan penggunaan kunci 2048 bit untuk penggunaan kunci khusus tambahan (RSA Laboratories n.d.).

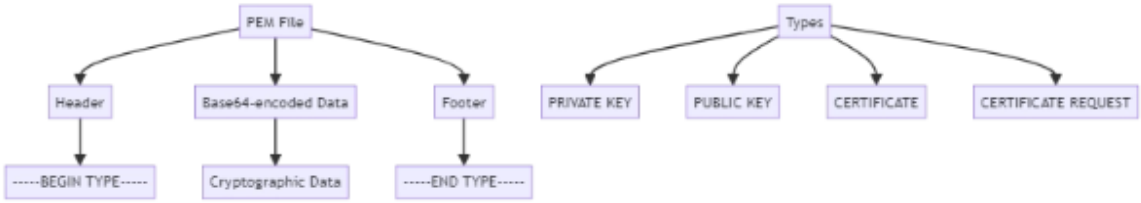
1.6. Format dan Penyimpanan Kunci RSA

a. Format Kunci

   Kunci RSA biasanya disimpan dalam dua format utama: PEM dan DER. Dalam program ini digunakan format **PEM (Privacy Enhanced Mail)**.

   File PEM adalah file berbasis teks yang mengikuti format tertentu untuk menyimpan data kriptografi Header dan footer menunjukkan jenis data yang disimpan, seperti kunci pribadi, kunci publik, sertifikat, atau permintaan sertifikat. Format PEM dirancang agar mudah dibaca manusia dan mudah ditransfer. Format ini menggunakan pengkodean base64 untuk merepresentasikan data biner dalam

format berbasis teks. Pengkodean base64 mengubah data biner menjadi sekumpulan karakter ASCII yang terbatas , sehingga cocok untuk transmisi melalui protokol berbasis teks seperti email.



(Struktur File PEM)

File PEM dapat menyimpan berbagai jenis data kriptografi, termasuk :

- Kunci pribadi (RSA, DSA, EC)
- Kunci publik
- Sertifikat (X.509)
- Permintaan penandatanganan sertifikat (CSR)

Setiap jenis data memiliki penanda header dan footer yang spesifik. Misalnya:

- Kunci pribadi: ——BEGIN PRIVATE KEY—— dan ——END PRIVATE KEY——
- Kunci publik: ——BEGIN PUBLIC KEY—— dan ——END PUBLIC KEY——
- Sertifikat: ——BEGIN SERTIFIKAT—— dan ——END SERTIFIKAT——

Dengan menggunakan penanda yang jelas ini, file PEM memudahkan identifikasi dan ekstraksi data yang relevan.

Contoh struktur kunci publik dalam format PEM:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAtUvA...
...ZQIDAQAB
-----END RSA PUBLIC KEY-----
```

Format PEM memudahkan distribusi dan kompatibilitas lintas sistem karena dapat disimpan dalam file teks biasa dan dibaca dengan editor teks umum.

b. Penyimpanan Kunci

Pada program yang digunakan, penyimpanan dilakukan sebagai berikut:

- **Kunci publik** disimpan dalam file bernama public_key.pub
- **Kunci privat** disimpan dalam file bernama private_key.pri

Kedua kunci disimpan menggunakan metode save_pkcs1('PEM') dari pustaka Python rsa. Metode ini memastikan bahwa kunci disimpan dalam format PEM sehingga mudah dimuat kembali menggunakan metode load_pkcs1() baik untuk kunci publik maupun privat.

c. Pembuatan Kunci RSA

Pseudocode untuk pembuatan kunci RSA adalah sebagai berikut:

Algorithm 1: RSA Key Generation

Data: The multiplication n of two large primes p and q, The Eulers' totient of n, a prime e
Result: The Public-Key and Private-Key, or INVALID
if $1 < e < \phi(n)$ *and* $gcd[e, \phi(n)] = 1$ **then**
  Calculate d such that e.d mod $\phi(n) \equiv 1$
  Public-Key $\leftarrow (e, n)$;
  Private-Key $\leftarrow (d, n)$;
  return Public-Key, Private-Key;
else
  return INVALID;
end

d. Enkripsi Pesan RSA

Pseudocode untuk enkripsi pesan RSA adalah sebagai berikut:

Algorithm 2: RSA Message Encryption

Data: Public-Key (e, n), Plaintext m
Result: The encrypted Ciphertext c
c $\leftarrow m^e$ mod n;
return Ciphertext c;

e. Dekripsi Pesan RSA

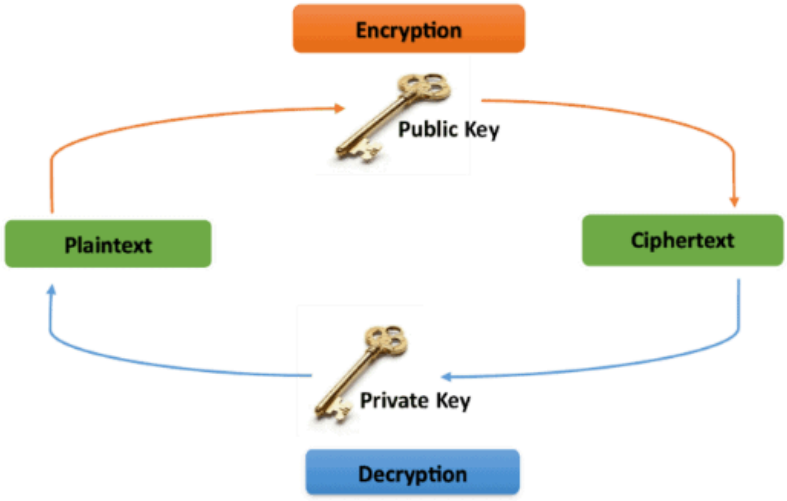
Pseudocode untuk dekripsi pesan RSA adalah sebagai berikut:

Algorithm 3: RSA Message Decryption

Data: Private-Key (d, n), Ciphertext c
Result: The decrypted Plaintext m
m $\leftarrow c^d$ mod n;
return Plaintext m;

f. Penggunaan Tombol

Penggunaan kunci publik dan kunci privat dalam algoritma RSA adalah sebagai berikut:



1.7. Enkripsi dan Dekripsi File

a. Enkripsi

- Enkripsi dilakukan dengan menggunakan **kunci publik** yang dimuat dari file .pub.
- File yang ingin dienkripsi dibaca dalam mode byte (rb) karena RSA bekerja pada data biner.
- Karena RSA memiliki batas ukuran data yang dapat dienkripsi (sekitar 245 byte untuk RSA 2048-bit), maka data dibagi menjadi potongan kecil (chunk) berukuran maksimal 200 byte.
- Setiap chunk kemudian dienkripsi satu per satu dan hasil akhirnya digabungkan menjadi file .enc.

b. Dekripsi

- Dekripsi dilakukan dengan menggunakan **kunci privat** dari file .pri.
- File terenkripsi (.enc) dibaca dan diproses dalam potongan 256 byte (ukuran hasil enkripsi RSA 2048-bit).

- Tiap chunk didekripsi secara bertahap dan digabungkan menjadi data asli, lalu disimpan dalam file .dec.

Metode ini memungkinkan penggunaan RSA secara langsung untuk mengenkripsi isi file meskipun secara umum RSA lebih cocok digunakan untuk mengenkripsi kunci simetris (seperti AES) dalam sistem hybrid encryption.

1.8. Ringkasan Format File Kunci

Secara ringkas, kunci RSA pada program ini disimpan dalam format PEM, yaitu format yang memuat data biner kunci yang telah dienkode dalam bentuk teks menggunakan skema Base64. Format ini disimpan dalam file teks yang diberi ekstensi tertentu agar dapat dikenali dan diproses dengan tepat. Kunci publik disimpan dalam file dengan ekstensi .pub, sedangkan kunci privat disimpan dalam file berekstensi .pri. Misalnya, jika dibuka dengan editor teks, isi file public_key.pub akan memperlihatkan teks seperti:

```
-----BEGIN RSA PUBLIC KEY-----

MIIBCgKCAQEAr...sQIDAQAB

-----END RSA PUBLIC KEY-----
```

Kedua file ini dapat digunakan kembali oleh program Python menggunakan pustaka rsa yang mendukung format PEM. Artinya, file kunci tersebut tidak hanya dapat disimpan dan dibaca ulang, tetapi juga dapat dipindahkan antar sistem dengan mudah dan tetap kompatibel.

1.9. Kriptografi Kunci Publik

Konsep kriptografi kunci publik pertama kali diperkenalkan oleh Whitfield Diffie dan Martin Hellman pada tahun 1976 dalam makalah mereka yang berjudul "New Directions in Cryptography". Penemuan ini merevolusi bidang kriptografi dengan memungkinkan komunikasi aman tanpa perlu berbagi kunci rahasia sebelumnya .

Kriptografi kunci publik, juga dikenal sebagai **kriptografi asimetris**, adalah sistem kriptografi yang menggunakan sepasang kunci:

- **Kunci publik**: Dapat disebarkan secara bebas dan digunakan untuk mengenkripsi pesan.

- **Kunci privat**: Disimpan secara rahasia oleh pemiliknya dan digunakan untuk mendekripsi pesan yang telah dienkripsi dengan kunci publik tersebut.

Keamanan sistem ini bergantung pada kesulitan matematis dalam memecahkan hubungan antara kunci publik dan kunci privat, seperti pada algoritma RSA yang didasarkan pada kesulitan faktorisasi bilangan besar .

Aspek	Kriptografi Kunci Simetris	Kriptografi Kunci Publik
Jumlah Kunci	Satu kunci yang sama untuk enkripsi dan dekripsi.	Sepasang kunci : publik dan privat.
Distribusi Kunci	Memerlukan saluran aman untuk berbagi kunci.	Kunci publik dapat disebarkan secara bebas.
Kecepatan Proses	Cepat dan efisien untuk data dalam jumlah besar.	Lebih lambat karena kompleksitas matematis.
keamanan	Bergantung pada kerahasiaan kunci bersama .	Lebih aman : kunci privat tidak perlu dibagikan.
Aplikasi umum	Enkripsi data besar, VPN, enkrisi file.	Tanda tangan digital, SSL/TLS, otentikasi.

Pada penerapanya, kedua metode kriptografi ini sering digunakan secara bersamaan dalam sistem yang disebut **kriptografi hibrida**. Dalam sistem ini, kriptografi kunci publik digunakan untuk mendistribusikan kunci simetris secara aman, yang kemudian digunakan untuk mengenkripsi dan mendekripsi data dalam jumlah besar. Contoh penerapan sistem hibrida ini dapat ditemukan pada protokol keamanan seperti SSL/TLS yang digunakan dalam komunikasi internet .

1.10. Notasi Hexadecimal dalam RSA

RSA (Rivest–Shamir–Adleman) adalah salah satu algoritma kriptografi kunci publik yang sangat bergantung pada operasi bilangan besar, terutama perkalian dan pemangkatan bilangan bulat. Dalam proses pembangkitan kunci RSA, digunakan bilangan-bilangan raksasa seperti modulus (n), eksponen publik (e), dan eksponen privat (d). Karena nilai-nilai ini seringkali memiliki panjang ratusan digit bila direpresentasikan dalam desimal, maka notasi hexadecimal—yang menggunakan basis 16—sering dipilih untuk merepresentasikannya secara lebih ringkas dan mudah dibaca. Dalam notasi hexadecimal, angka 0–9 dan huruf A sampai F digunakan untuk melambangkan nilai 0 hingga 15, sehingga setiap digit hexadecimal mewakili tepat 4 bit informasi.

Penggunaan notasi hexadecimal dalam konteks RSA memberikan beberapa keuntungan. Pertama, representasi angka menjadi jauh lebih kompak; sebuah bilangan besar yang memiliki puluhan atau ratusan digit desimal bisa direpresentasikan dengan jauh lebih sedikit digit ketika ditulis dalam format hexadecimal. Hal ini sangat berguna untuk keperluan debugging, dokumentasi, dan pertukaran data antar sistem, di mana keakuratan dan konsistensi data kriptografi sangat krusial. Selain itu, karena setiap digit hexadecimal berkorelasi langsung dengan 4 bit, konversi antara bentuk biner (yang digunakan oleh komputer) dan bentuk yang ditampilkan menjadi lebih mudah, sehingga membantu para pengembang untuk memeriksa struktur dan integritas dari angka-angka yang terlibat dalam algoritma RSA.

Sebagai contoh konkret, salah satu eksponen publik yang paling umum digunakan dalam RSA adalah angka 65537. Dalam notasi desimal, angka ini terlihat sederhana, namun ketika dikonversi ke notasi hexadecimal nilainya menjadi 0x10001. Penggunaan angka 0x10001 sebagai eksponen publik tidak hanya lebih ringkas dalam beberapa konteks penulisan, tetapi juga menegaskan bahwa angka tersebut dipilih karena struktural dan sifat matematisnya—misalnya, memiliki bilangan biner dengan sedikit bit “1”, yang membuat perhitungan perpangkatan lebih efisien.

Contoh lain yang lebih lengkap dalam implementasi RSA misalkan kita telah menghasilkan sepasang kunci dengan komponen-komponen berikut:
- Modulus (n): 0xC34B4F8D3FB9C8A0BB4431B
- Eksponen Publik (e): 0x10001
- Eksponen Privat (d): 0x2A1B3C4D5E6F7890ABCDEFF

Dalam contoh tersebut, nilai (n) yang sangat besar direpresentasikan secara ringkas menggunakan notasi hexadecimal. Ini tidak hanya memudahkan pemeriksaan visual dan verifikasi, tetapi juga memudahkan interoperabilitas antar perangkat lunak dan platform yang mengharuskan format input/output angka yang seragam. Dengan demikian, walaupun di balik layar komputer mengolah angka tersebut dalam bentuk biner, representasi hexadecimal memberikan jembatan antara dunia mesin dan manusia.

1.11. Keamanan dan Pertimbangan Algoritma RSA

RSA mengandalkan pada kesulitan faktorisasi bilangan besar sebagai dasar keamanannya. Proses enkripsi dan dekripsi menggunakan pasangan kunci publik dan privat yang dibentuk dari dua bilangan prima besar.

Keamanan RSA bergantung pada:

- **Kesulitan Faktorisasi**: Memecahkan bilangan besar menjadi faktor primanya merupakan masalah yang kompleks secara komputasional.
- **Pemilihan Bilangan Prima**: Bilangan prima yang digunakan harus dipilih secara acak dan cukup besar untuk mencegah serangan faktorisasi.

1.12. Ancaman dan Serangan terhadap RSA

Meskipun RSA dianggap aman, terdapat beberapa jenis serangan yang dapat mengancam keamanannya:

- **Serangan Faktorisasi**: Jika bilangan prima yang digunakan tidak cukup besar atau tidak acak, penyerang dapat memfaktorkan modulus RSA dan memperoleh kunci privat
- **Serangan Saluran Samping (Side-Channel Attacks)**: Penyerang dapat memanfaatkan informasi fisik seperti waktu eksekusi atau konsumsi daya untuk mengekstrak kunci privat.
- **Serangan Padding**: Implementasi padding yang tidak aman, seperti PKCS#1 v1.5, rentan terhadap serangan seperti Bleichenbacher attack.
- **Serangan ROCA**: Kerentanan dalam generasi kunci RSA pada perangkat keras tertentu memungkinkan penyerang mengekstrak kunci privat dari kunci publik.

1.13. Pertimbangan Implementasi RSA

Untuk memastikan keamanan RSA dalam praktik, beberapa pertimbangan penting meliputi:

- **Ukuran Kunci**: Penggunaan kunci dengan panjang minimal 2048 bit direkomendasikan untuk keamanan jangka panjang.
- **Sumber Bilangan Acak**: Menggunakan generator bilangan acak yang kuat dan terpercaya untuk pemilihan bilangan prima.
- **Penggunaan Padding yang Aman**: Menerapkan skema padding seperti OAEP untuk mencegah serangan padding.
- **Perlindungan terhadap Serangan Saluran Samping**: Mengimplementasikan countermeasures seperti blinding untuk mencegah serangan side-channel

1.14. Tantangan Masa Depan: Komputasi Kuantum

Kemajuan dalam komputasi kuantum menimbulkan ancaman terhadap RSA, karena algoritma seperti Shor dapat memfaktorkan bilangan besar secara efisien. Meskipun saat ini belum ada komputer kuantum yang mampu memecahkan RSA 2048-bit, penelitian dan pengembangan algoritma kriptografi pasca-kuantum sedang berlangsung untuk mengantisipasi ancaman ini.

Kode Program RSA

1. Import library yang diperlukan

```
import os
import time
```

2. Mendefinisikan fungsi dasar RSA

```
# --------------------------
# Fungsi dasar RSA
# --------------------------

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def modinv(a, m):
    """Menghitung invers modular a mod m menggunakan Extended
Euclidean Algorithm."""
    m0 = m
    x, y = 1, 0
    if m == 1:
        return 0
    while a > 1:
        q = a // m
        t = m
        m = a % m
        a = t
        t = y
        y = x - q * y
        x = t
    if x < 0:
        x += m0
    return x

def generate_keys():
    """
    Menghasilkan pasangan kunci RSA sederhana.
```

```
    Untuk tujuan pembelajaran, digunakan bilangan prima kecil.
    p = 61, q = 53 ---> n = 3233, phi(n) = 3120, pilih e = 17.
    d merupakan invers modular e mod phi.
    """
    p = 61
    q = 53
    n = p * q               # n = 3233
    phi = (p - 1) * (q - 1)  # phi = 3120
    e = 17                  # eksponen publik (dipilih karena
gcd(17,3120)==1)
    d = modinv(e, phi)      # eksponen privat
    return (e, n), (d, n)

def rsa_encrypt(plaintext, pubkey):
    """
    Mengenkripsi pesan dengan RSA.
    Untuk setiap karakter pada plaintext, enkripsi dengan:
        cipher = (ord(character) ** e) mod n
    """
    e, n = pubkey
    cipher_numbers = [pow(ord(ch), e, n) for ch in plaintext]
    return cipher_numbers

def rsa_decrypt(cipher_numbers, privkey):
    """
    Mendekripsi daftar bilangan ciphertext dengan RSA.
    Untuk setiap bilangan, lakukan dekripsi:
        plaintext = chr((cipher ** d) mod n)
    """
    d, n = privkey
    plaintext = ''.join(chr(pow(num, d, n)) for num in
cipher_numbers)
    return plaintext
```

3. Mendefinisikan fungsi untuk menyimpan dan memuat Kunci (private key dan public key)

```
# --------------------------
# Fungsi Penyimpanan dan Pemuatan Kunci
# --------------------------

def save_keys(pubkey, privkey, base_name):
    """
    Menyimpan kunci publik dan privat ke file dengan ekstensi .pub
dan .pri.
    Format file: "<eksponen> <modulus>"
    """
    pub_filename = base_name + ".pub"
    pri_filename = base_name + ".pri"
    try:
        with open(pub_filename, "w", encoding="utf-8") as f:
            f.write(f"{pubkey[0]} {pubkey[1]}")
        with open(pri_filename, "w", encoding="utf-8") as f:
            f.write(f"{privkey[0]} {privkey[1]}")
        print(f"\nKunci publik disimpan ke: {pub_filename}")
        print(f"Kunci privat disimpan ke: {pri_filename}")
    except Exception as err:
        print("Terjadi kesalahan saat menyimpan kunci:", err)

def load_public_key_from_file(filename):
    """Pemuatan kunci publik dari file dengan format: 'e n'."""
    try:
        with open(filename, "r", encoding="utf-8") as f:
            data = f.read().strip().split()
            e, n = int(data[0]), int(data[1])
        return (e, n)
    except Exception as err:
        print("Gagal membaca kunci publik:", err)
        return None

def load_private_key_from_file(filename):
```

```
    """Pemuatan kunci privat dari file dengan format: 'd n'."""
    try:
        with open(filename, "r", encoding="utf-8") as f:
            data = f.read().strip().split()
            d, n = int(data[0]), int(data[1])
        return (d, n)
    except Exception as err:
        print("Gagal membaca kunci privat:", err)
        return None
```

4. Mendefinisikan fungsi enkripsi file

```
# --------------------------
# Fungsi Enkripsi File
# --------------------------
def encrypt_file():
    # Meminta input nama file plaintext (beserta path-nya)
    input_file = input("Masukkan path file plaintext (.txt):
").strip()
    if not os.path.exists(input_file):
        print("File tidak ditemukan.")
        return

    try:
        with open(input_file, "r", encoding="utf-8") as f:
            plaintext = f.read()
    except Exception as err:
        print("Gagal membaca file:", err)
        return

    # Pilihan cara pengambilan kunci publik
    pilihan = input("Ambil kunci publik dari file? (y/n):
").strip().lower()
    if pilihan.startswith('y'):
        key_file = input("Masukkan path file kunci publik (*.pub):
").strip()
        if not os.path.exists(key_file):
            print("File kunci publik tidak ditemukan.")
```

```
            return
        pubkey = load_public_key_from_file(key_file)
        if pubkey is None:
            return
    else:
        try:
            e = int(input("Masukkan nilai e (eksponen publik): "))
            n = int(input("Masukkan nilai n (modulus): "))
            pubkey = (e, n)
        except Exception as err:
            print("Input kunci tidak valid:", err)
            return

    # Proses enkripsi dan ukur waktu proses
    start_time = time.perf_counter()
    cipher_numbers = rsa_encrypt(plaintext, pubkey)
    end_time = time.perf_counter()
    encryption_time = end_time - start_time

    # Konversi bilangan cipher ke notasi heksadesimal
    cipher_hex_list = [hex(num)[2:] for num in cipher_numbers]  #
menghilangkan "0x"
    cipher_hex = ' '.join(cipher_hex_list)

    # Tampilkan plaintext dan ciphertext ke layar
    print("\n--- Plaintext ---")
    print(plaintext)
    print("\n--- Ciphertext (Hex) ---")
    print(cipher_hex)

    # Menyimpan ciphertext ke file
    output_file = input("Masukkan path file untuk menyimpan
ciphertext: ").strip()
    try:
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(cipher_hex)
        file_size = os.path.getsize(output_file)
        print(f"\nCiphertext telah disimpan ke: {output_file}")
        print(f"Lama waktu enkripsi: {encryption_time:.6f} detik")
        print(f"Ukuran file ciphertext: {file_size} bytes")
```

```
    except Exception as err:
        print("Gagal menyimpan file ciphertext:", err)
```

5. Mendefinisikan fungsi dekripsi file

```
# --------------------------
# Fungsi Dekripsi File
# --------------------------
def decrypt_file():
    # Meminta input file ciphertext
    cipher_file = input("Masukkan path file ciphertext (.txt):
").strip()
    if not os.path.exists(cipher_file):
        print("File ciphertext tidak ditemukan.")
        return

    try:
        with open(cipher_file, "r", encoding="utf-8") as f:
            cipher_text = f.read().strip()
    except Exception as err:
        print("Gagal membaca file ciphertext:", err)
        return

    # Konversi notasi heksadesimal ke bilangan integer
    try:
        cipher_numbers = [int(token, 16) for token in
cipher_text.split()]
    except Exception as err:
        print("Format file ciphertext tidak valid:", err)
        return

    # Pilihan cara pengambilan kunci privat
    pilihan = input("Ambil kunci privat dari file? (y/n):
").strip().lower()
    if pilihan.startswith('y'):
        key_file = input("Masukkan path file kunci privat (*.pri):
").strip()
        if not os.path.exists(key_file):
```

```
            print("File kunci privat tidak ditemukan.")
            return
        privkey = load_private_key_from_file(key_file)
        if privkey is None:
            return
    else:
        try:
            d = int(input("Masukkan nilai d (eksponen privat): "))
            n = int(input("Masukkan nilai n (modulus): "))
            privkey = (d, n)
        except Exception as err:
            print("Input kunci tidak valid:", err)
            return

    # Proses dekripsi dan ukur waktu proses
    start_time = time.perf_counter()
    plaintext = rsa_decrypt(cipher_numbers, privkey)
    end_time = time.perf_counter()
    decryption_time = end_time - start_time

    # Tampilkan plaintext hasil dekripsi ke layar
    print("\n--- Plaintext ---")
    print(plaintext)

    # Simpan hasil dekripsi ke file
    output_file = input("Masukkan path file untuk menyimpan hasil
dekripsi: ").strip()
    try:
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(plaintext)
        file_size = os.path.getsize(output_file)
        print(f"\nPlaintext telah disimpan ke: {output_file}")
        print(f"Lama waktu dekripsi: {decryption_time:.6f} detik")
        print(f"Ukuran file plaintext: {file_size} bytes")
    except Exception as err:
        print("Gagal menyimpan file plaintext:", err)
```

6. Mendefinisikan fungsi generate dan menyimpan kunci RSA

```
# --------------------------
# Fungsi Generate dan Penyimpanan Kunci RSA
# --------------------------
def generate_and_save_keys():
    pubkey, privkey = generate_keys()
    print("\nKunci RSA telah dihasilkan:")
    print("Kunci publik (e, n):", pubkey)
    print("Kunci privat (d, n):", privkey)
    base_name = input("Masukkan nama dasar file kunci (tanpa
ekstensi): ").strip()
    if base_name == "":
        print("Nama dasar kosong. Kunci tidak disimpan.")
        return
    save_keys(pubkey, privkey, base_name)
```

7. Mendefinisikan fungsi main sebagai fungsi menu utama program

```
# --------------------------
# Menu Utama Program
# --------------------------
def main():
    while True:
        print("\n=== Program RSA Sederhana ===")
        print("1. Pembuatan Kunci RSA")
        print("2. Enkripsi File")
        print("3. Dekripsi File")
        print("4. Keluar")
        pilihan = input("Pilih menu (1/2/3/4): ").strip()

        if pilihan == "1":
            generate_and_save_keys()
        elif pilihan == "2":
```

```
            encrypt_file()
        elif pilihan == "3":
            decrypt_file()
        elif pilihan == "4":
            print("Keluar program.")
            break
        else:
            print("Pilihan tidak valid, silakan pilih kembali.")

if __name__ == "__main__":
    main()
```

8. Hasil (Running Program)
    a. Pembuatan Kunci RSA

```
=== Program RSA Sederhana ===
1. Pembuatan Kunci RSA
2. Enkripsi File
3. Dekripsi File
4. Keluar
Pilih menu (1/2/3/4): 1

Kunci RSA telah dihasilkan:
Kunci publik (e, n): (17, 3233)
Kunci privat (d, n): (2753, 3233)
Masukkan nama dasar file kunci (tanpa ekstensi): key

Kunci publik disimpan ke: key.pub
Kunci privat disimpan ke: key.pri
```

b. Enkripsi File

```
=== Program RSA Sederhana ===
1. Pembuatan Kunci RSA
2. Enkripsi File
3. Dekripsi File
4. Keluar
Pilih menu (1/2/3/4): 2

Masukkan path file plaintext (.txt): D:\Data_Encryption\RSA\Test.txt

Ambil kunci publik dari file? (y/n): y

Masukkan path file kunci publik (*.pub): D:\Data_Encryption\key.pub

--- Plaintext ---
Ini adalah file yang akan dienkripsi

--- Ciphertext (Hex) ---
5ce 8bb c6b 7c8 660 6ed 660 2e9 660 87a 7c8 559 c6b 2e9 521 7c8 1e7 660 8bb b6b
7c8 660 2b2 660 8bb 7c8 6ed c6b 521 8bb 2b2 96c c6b 264 4ce c6b

Masukkan path file untuk menyimpan ciphertext: D:\Data_Encryption\enkription.txt

Ciphertext telah disimpan ke: D:\Data_Encryption\enkription.txt

Lama waktu enkripsi: 0.000021 detik
Ukuran file ciphertext: 143 bytes
```

c. Dekripsi File

```
=== Program RSA Sederhana ===
1. Pembuatan Kunci RSA
2. Enkripsi File
3. Dekripsi File
4. Keluar
Pilih menu (1/2/3/4): 3

Masukkan path file ciphertext (.txt): D:\Data_Encryption\enkription.txt

Ambil kunci privat dari file? (y/n): y

Masukkan path file kunci privat (*.pri): D:\Data_Encryption\key.pri

--- Plaintext ---
Ini adalah file yang akan dienkripsi

Masukkan path file untuk menyimpan hasil dekripsi:
D:\Data_Encryption\RSA\dekripsi.txt

Plaintext telah disimpan ke: D:\Data_Encryption\dekripsi.txt

Lama waktu dekripsi: 0.000030 detik
Ukuran file plaintext: 36 bytes
```

9. Struktur Direktori yang terbentuk

```
> RSA
  → RSA_Simple_TXT.py
  → key.pri
  → key.pub
  → Test.txt
  → enkription.txt
  → dekripsi.txt
```