

Kelompok 7

Ararya Pramadani Alief Rahman 215150207111030

Azarya Santoso 215150200111026

Made Diksa Pitra 215150201111025

Judul: AES DHKE File Send & Encryption

Topik: Pengamanan File

1. Mengapa topik Anda perlu dilakukan? Apakah urgensinya?

Dalam era digital yang terus berkembang, keamanan informasi menjadi krusial. Penggunaan teknologi enkripsi file dan pertukaran kunci, seperti melalui algoritma Diffie-Hellman, menjadi sangat penting. Enkripsi file memastikan bahwa informasi yang dikirim atau disimpan tidak dapat dibaca oleh pihak yang tidak berwenang, sementara protokol pertukaran kunci yang aman seperti Diffie-Hellman memungkinkan pihak yang berkomunikasi untuk menciptakan kunci bersama tanpa mengirimkan kunci itu sendiri [1]. Hal ini tidak hanya melindungi informasi dari akses yang tidak sah, tetapi juga meminimalkan risiko kunci yang disadap selama proses pertukaran. Dengan menggunakan teknologi ini, keamanan data dalam komunikasi digital menjadi lebih terjamin, memberikan perlindungan yang diperlukan terhadap serangan dan intersepsi informasi yang tidak diinginkan [2].

Urgensi topik ini terletak pada kebutuhan untuk melindungi kerahasiaan informasi di dunia digital yang terus berkembang. Enkripsi file memastikan bahwa data yang ditransmisikan atau disimpan tidak dapat diakses oleh pihak yang tidak berwenang, sedangkan protokol pertukaran kunci seperti Diffie-Hellman memungkinkan entitas yang berkomunikasi untuk berbagi kunci enkripsi tanpa mengirimkan kunci itu sendiri. Keamanan yang kuat dalam enkripsi dan pertukaran kunci menjadi penting karena mencegah serangan pengintai yang berusaha mencuri atau memodifikasi informasi saat berpindah dari satu titik ke titik lainnya [3].

2. Apakah terdapat penelitian lain yang mirip dengan topik Anda? Jelaskan!

Ya, terdapat penelitian lain yang mirip dengan topik kelompok kami diantaranya beberapa penelitian menggunakan AES sebagai algoritma enkripsi data, DHKE sebagai pertukaran kunci serta beberapa penelitian menggunakan algoritma lain yang kemudian diterapkan pada sistem maupun protokol yang berbeda-beda mulai dari perangkat *cloud platform*, *Internet of Things*, TFTP, *mobile application* dan *Wireless Sensor Networks*.

3. Mengapa Anda memilih algoritma yang Anda terapkan? Jelaskan!

Algoritma Advanced Encryption Standard (AES) seringkali menjadi pilihan utama untuk enkripsi file karena kombinasi keamanan tinggi dan kinerja yang cepat. Dikenal sebagai salah satu algoritma yang paling aman dan tahan terhadap serangan kriptanalisis, AES menawarkan tingkat keamanan yang tinggi dengan panjang kunci yang bervariasi (128-bit, 192-bit, atau 256-bit). Kemampuannya untuk melakukan enkripsi dan dekripsi

dengan cepat menjadikannya cocok untuk berbagai perangkat, dari sistem komputasi tingkat tinggi hingga perangkat mobile [2]. AES juga merupakan standar yang diakui secara internasional dan banyak digunakan dalam berbagai aplikasi, termasuk komunikasi online, pengamanan data pemerintah, dan perlindungan informasi di sektor swasta. Meskipun RSA dan DES memiliki kegunaan uniknya sendiri, AES sering dipilih karena kombinasi keamanan yang tinggi dan efisiensi kinerjanya dalam melakukan enkripsi file dengan volume besar [5].

4. Jelaskan secara detail terkait cara kerja algoritma yang Anda gunakan?

- Diffie Hellman Key Exchange (DHKE)

Algoritma pertukaran kunci Diffie-Hellman (DHKE) adalah metode kriptografi yang digunakan untuk menghasilkan kunci rahasia bersama antara dua pihak tanpa perlu bertukar kunci secara langsung [4].

- Menentukan sebuah bilangan prima
- Menentukan bilangan bulat positif yang disebut generator. Generator g harus relatif prima terhadap p .
- A memilih kunci privat acak (a) dari ruang kunci yang mungkin
- A menghitung kunci publik $A = g^a \bmod p$
- A mengirim kunci publik A ke pihak B
- B memilih kunci privat acak (b) dari ruang kunci yang mungkin
- B menghitung kunci publik $B = g^b \bmod p$
- B mengirim kunci publik B ke pihak A
- Pihak A menerima kunci publik B dari pihak B dan pihak B menerima kunci publik A dari pihak A
- Shared Key dihitung oleh kedua belah pihak dengan rumus $B^a \bmod p$ atau $A^b \bmod p$
- Kunci bersama yang dihasilkan oleh kedua belah pihak akan sama, sehingga keduanya dapat menggunakan kunci ini sebagai kunci rahasia bersama untuk berkomunikasi.

- Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) adalah salah satu algoritma enkripsi kunci simetris yang paling umum digunakan untuk melindungi data yang disimpan atau dikirimkan melalui jaringan. AES memiliki beberapa tingkat keamanan, yang diukur dalam panjang kunci, yaitu 128-bit, 192-bit, dan 256-bit [1].

1. Inisialisasi Kunci (Key Expansion):

- AES 128-bit menerima kunci input dengan panjang 128 bit. Kunci ini kemudian dijadikan rahasia antara pengirim dan penerima.
- Proses ini melibatkan perluasan kunci (key expansion), di mana kunci asli dibagi menjadi beberapa kunci putaran (round keys). Jumlah kunci putaran tergantung pada panjang kunci.

2. Tahap Inisialisasi (Initial Round):

- Data blok yang akan dienkripsi (plaintext) dipecah menjadi blok-blok 128-bit.
- Blok plaintext akan di-XOR-kan dengan kunci putaran pertama.

3. Putaran Enkripsi (Rounds):

AES memiliki 10 putaran enkripsi untuk kunci 128-bit. Setiap putaran terdiri dari empat operasi utama: SubBytes, ShiftRows, MixColumns, dan AddRoundKey.

a. SubBytes:

Setiap byte dalam blok diubah menggunakan tabel substitusi (S-box). SubBytes memberikan non-linearitas pada algoritma dengan menggantikan setiap byte dengan nilai byte yang sesuai dalam S-box.

b. ShiftRows:

Setiap baris blok dipindahkan ke kiri. Operasi ini mencampur data di seluruh blok dan memastikan bahwa byte-byte yang sama tidak berada dalam satu baris.

c. MixColumns:

Kolom blok diubah dengan matriks operasi. Ini memberikan pengacakan dan difusi pada data, sehingga mempersulit analisis kriptografis.

d. AddRoundKey:

Blok di-XOR-kan dengan kunci putaran yang sesuai. Kunci putaran telah dihasilkan selama inisialisasi kunci.

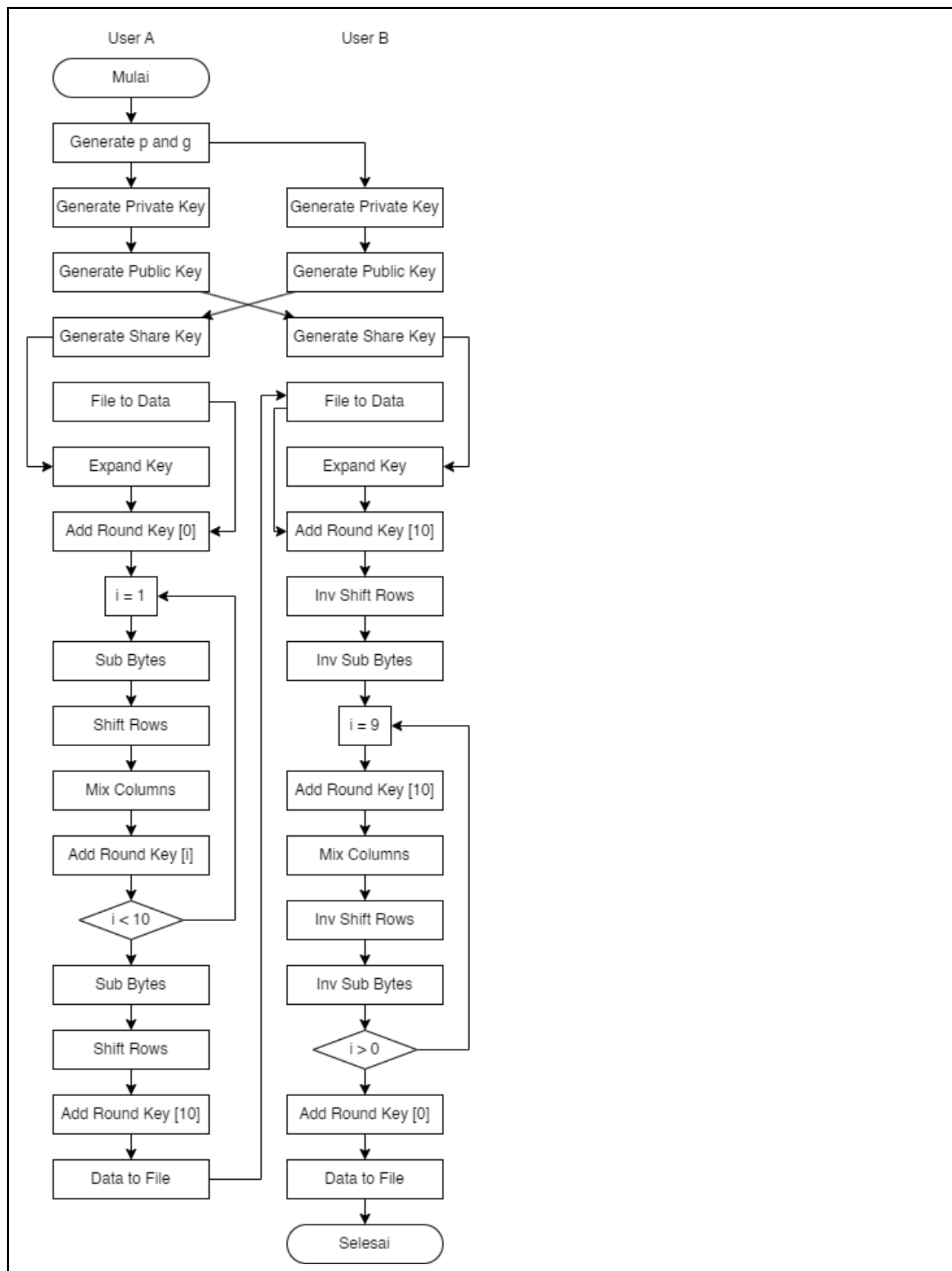
4. Putaran Terakhir (Final Round):

Putaran terakhir tidak melibatkan operasi MixColumns dan SubBytes, ShiftRows serta AddRoundKey masih diterapkan seperti pada putaran sebelumnya.

5. Hasil Akhir (Ciphertext):

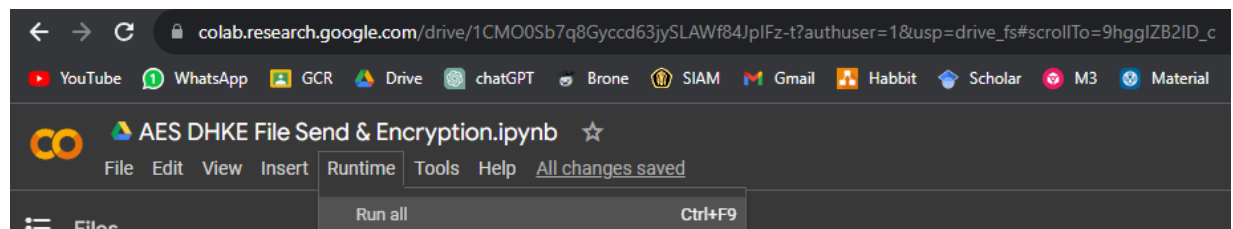
Setelah putaran terakhir, blok ciphertext dihasilkan dan disimpan.

5. Jelaskan dengan gambar serta flowchart cara kerja implementasi Anda secara garis besar!



6. Jelaskan secara detail bagaimana cara Anda mengimplementasikannya? (tutorial step-by-step from scratch disertai bukti tangkapan layar)

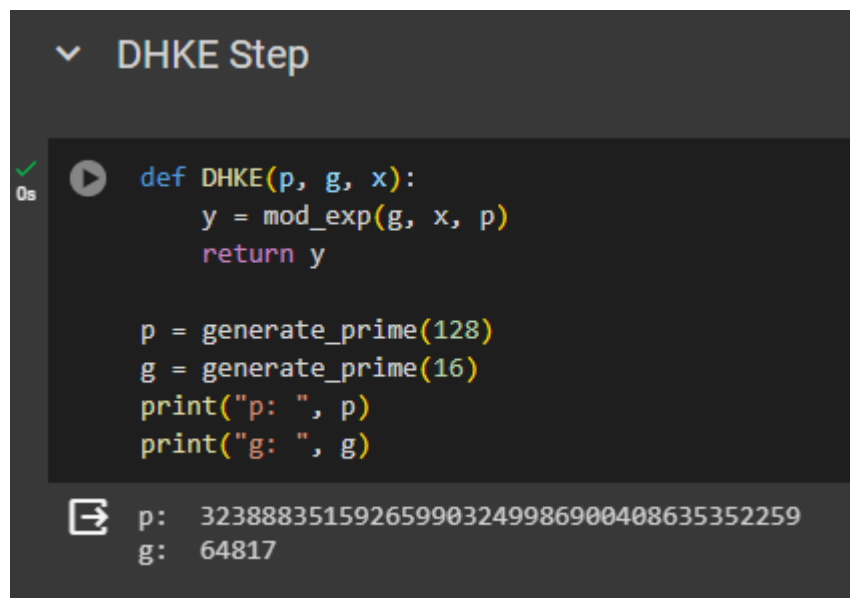
Membuka link & Run All



Pertama membuka

https://drive.google.com/open?id=1CEpmYMCOGUuxrixrX5wXG5ZuLKuRrM6t&usp=drive_fs. Caranya adalah membuka link drive dan membuka file colab research. Dari colab akan ke tab Runtime dan menjalankan run all untuk inisiasi semua kode.

Generate p & g



Langkah pertama untuk bertukar file yaitu melakukan key exchange. Hal ini dilakukan dengan teknik DHKE. Teknik ini diawali dengan menggenerate bilangan prima dengan menekan tombol play pada cell. Untuk p akan generate prima dengan 128 bit dan g dengan 16 bit. Hal ini dilakukan untuk menjamin keamanan key dengan panjang key 128 bit.

Generate private key

✓
0s

```
[17] private_key_a = generate_random(128)
      private_key_b = generate_random(128)
      print("Private Key A: ", private_key_a)
      print("Private Key B: ", private_key_b)
```

```
Private Key A: 94278460830755196014428891046127893408
Private Key B: 5572754603181439064586801575873016509
```

Selanjutnya setiap user akan generate private key dengan tombol play pada cell. Private key sendiri yaitu angka 128 bit untuk menjamin keamanan file.

Generate Public Key

✓
0s

```
[13] public_key_a = DHKE(p, g, private_key_a)
      public_key_b = DHKE(p, g, private_key_b)
      print("Public Key A: ", public_key_a)
      print("Public Key B: ", public_key_b)
```



```
Public Key A: 111097418719572514816040318374281181237
Public Key B: 100013482142407835768478549773347593777
```

Setiap user akan menggenerate public key dengan menekan tombol play pada cell. Public key digenerate dengan variabel dari private key masing2 user. Public key nantinya akan di share ke publik.

Generate Shared Key

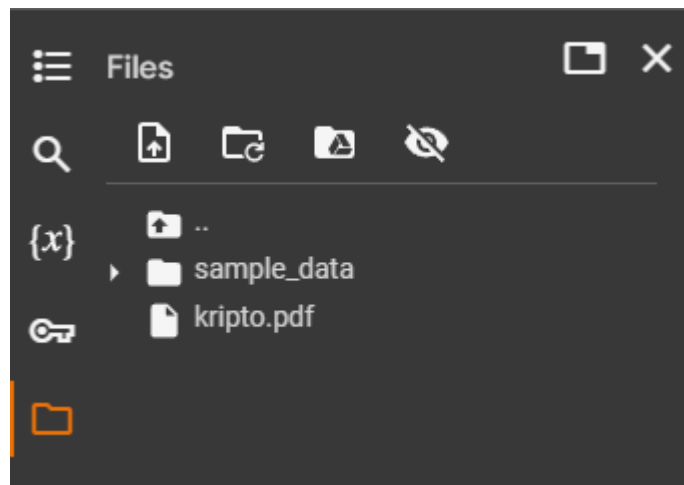
✓
0s

```
[19] shared_key_a = DHKE(p, public_key_b, private_key_a)
      shared_key_b = DHKE(p, public_key_a, private_key_b)
      print("Shared Key A: ", shared_key_a)
      print("Shared Key B: ", shared_key_b)
```

```
Shared Key A: 152029194775216395034792115876980280511
Shared Key B: 152029194775216395034792115876980280511
```

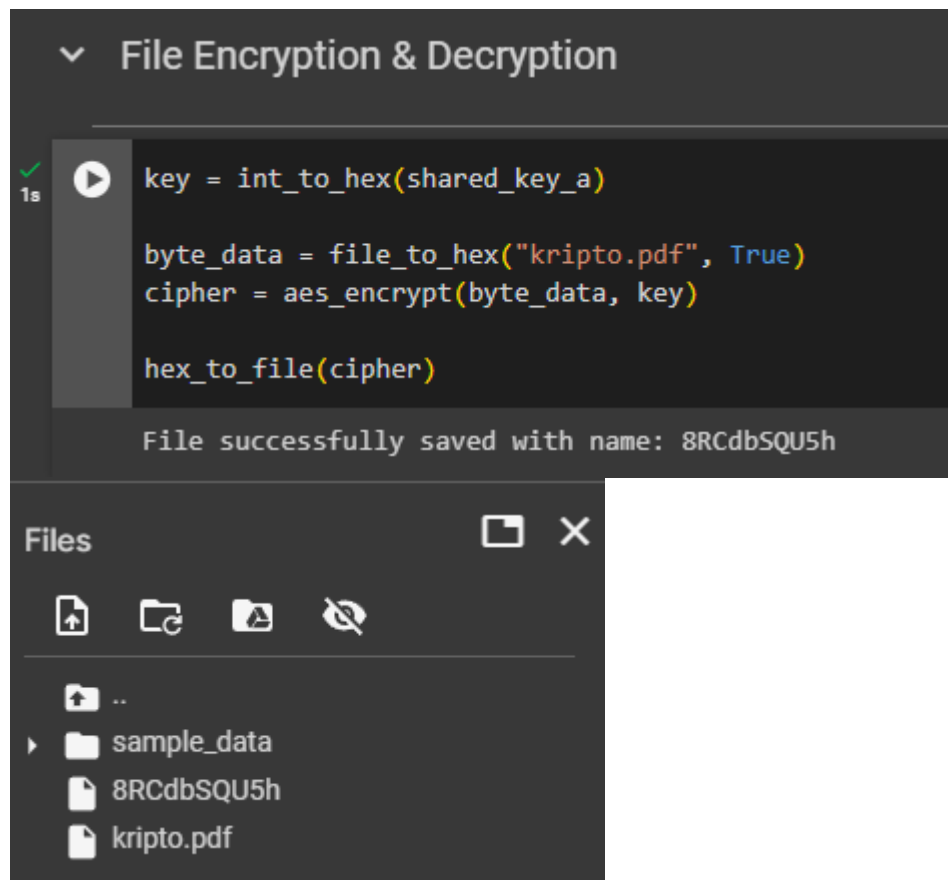
Setiap user akan menggenerate shared key dengan menekan tombol play pada cell. Shared key digenerate dengan variabel dari public key masing2 user. Shared key nantinya akan di gunakan untuk proses enkripsi pada shared key A dan dekripsi pada shared key B.

Memasukkan File



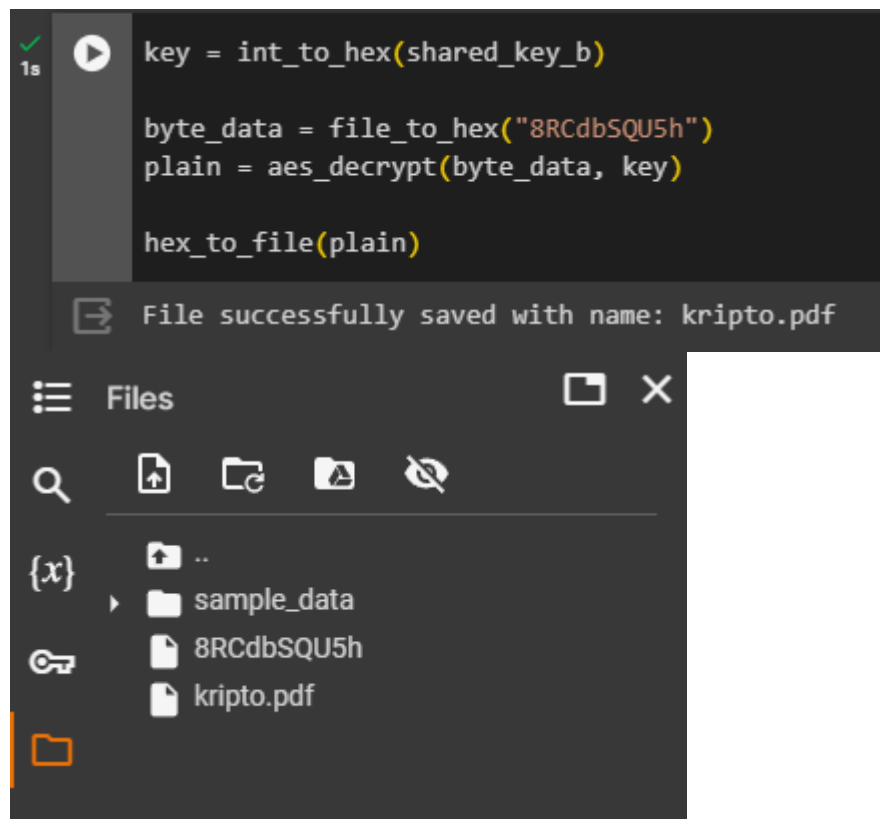
Langkah selanjutnya adalah mengenkripsi file yang akan dikirim. Caranya adalah membuka tab file di samping dan mengupload file ke tab tersebut. Bisa lewat dari tombol upload maupun menarik secara langsung file.

Enkripsi File



Langkah selanjutnya yaitu mengganti string pada `byte_data = file_to_hex()`. User mengganti dengan nama file original. Disini file yang dipakai bernama `kripto.pdf`. User menekan tombol play pada cell. Hal ini selanjutnya dimasukkan ke algoritma enkripsi AES yang nantinya akan menghasilkan output nama file baru yang telah terenkripsi. File baru dapat dilihat pada tab file.

Dekripsi File



```
key = int_to_hex(shared_key_b)

byte_data = file_to_hex("8RCdbSQU5h")
plain = aes_decrypt(byte_data, key)

hex_to_file(plain)
```

File successfully saved with name: kripto.pdf

Files

- ..
- sample_data
- 8RCdbSQU5h
- kripto.pdf

Untuk menjalankan cell tersebut, dapat menekan tombol play di bagian kiri cell tersebut. Cell tersebut, merupakan kode untuk melakukan dekripsi pada file yang telah terenkripsi sebelumnya. Untuk melakukan dekripsi file, pertama-tama mengubah `shared_key_b` yang sebelumnya bertipe `int` akan diubah menjadi format `hex` untuk kemudian disimpan dalam variabel `key`. Selanjutnya memasukkan nama file yang telah terenkripsi ke perintah `byte_data = file_to_hex`. Perintah tersebut digunakan untuk mengubah isi dari file tersebut menjadi `hex` dan hasilnya akan disimpan pada variabel `byte_data`. Selanjutnya melakukan proses dekripsi file menggunakan perintah `plain = aes_decrypt(byte_data, key)`. Perintah tersebut berfungsi untuk melakukan dekripsi berdasarkan `byte_data` dan `key` yang telah dibuat sebelumnya, dan hasil dari perintah tersebut disimpan ke dalam variabel `plain`. Terakhir, menggunakan perintah `hex_to_file(plain)` untuk mengubah isi file yang sebelumnya `hex` menjadi bentuk aslinya. Ketika cell tersebut berhasil dijalankan, maka akan menghasilkan output "File successfully saved with name: <nama file>" dan akan menghasilkan file baru yang telah didekripsi. File baru tersebut dapat dilihat pada tab file

7. Jelaskan secara detail kode program Anda? (tutorial step-by-step from scratch)

Tabel SBOX & Inverse

```
def sbox():
    return [
        [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,
0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76],
        [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4,
0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0],
        [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5,
0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15],
        [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12,
0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75],
        [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b,
0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84],
        [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf],
        [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9,
0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8],
        [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6,
0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2],
        [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73],
        [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee,
0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb],
        [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3,
0xac, 0x62, 0x91, 0x95, 0xe4, 0x79],
        [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56,
0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08],
        [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd,
0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a],
        [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35,
0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e],
        [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,
0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf],
        [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99,
0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]
    ]
```

```
def inv_sbox():
    return [
        [0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40,
0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb],
        [0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e,
0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb],
        [0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c,
0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e],
        [0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b,
0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25],
        [0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4,
0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92],
        [0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15,
```

```

0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84],
    [0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4,
0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06],
    [0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf,
0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b],
    [0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2,
0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73],
    [0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9,
0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e],
    [0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7,
0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b],
    [0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb,
0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4],
    [0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12,
0x10, 0x59, 0x27, 0x80, 0xec, 0x5f],
    [0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5,
0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef],
    [0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb,
0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61],
    [0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69,
0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d]
]

```

Fungsi "sbox()" mengembalikan matriks substitusi (S-Box). Matriks ini terdiri dari elemen-elemen heksadesimal yang menentukan penggantian byte-byte masukan pada proses enkripsi. Setiap elemen matriks S-Box adalah hasil substitusi dari 8-bit binary input ke 8-bit binary output. Sebaliknya, fungsi "inv_sbox()" mengembalikan matriks substitusi invers (Inverse S-Box) yang digunakan dalam tahap dekripsi AES. Matriks ini digunakan untuk mengembalikan byte-byte ke nilai semula selama proses dekripsi.

Tabel Multiplication 1,2,3,4,9,11,13,14

```

def t1():
    return [

0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d
,0x0e,0x0f,

0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d
,0x1e,0x1f,

0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d
,0x2e,0x2f,

0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d
,0x3e,0x3f,

```

```
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d  
,0x4e,0x4f,
```

```
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d  
,0x5e,0x5f,
```

```
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d  
,0x6e,0x6f,
```

```
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d  
,0x7e,0x7f,
```

```
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d  
,0x8e,0x8f,
```

```
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d  
,0x9e,0x9f,
```

```
0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xaa,0xab,0xac,0xad  
,0xae,0xaf,
```

```
0xb0,0xb1,0xb2,0xb3,0xb4,0xb5,0xb6,0xb7,0xb8,0xb9,0xba,0xbb,0xbc,0xbd  
,0xbe,0xbf,
```

```
0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd  
,0xce,0xcf,
```

```
0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd  
,0xde,0xdf,
```

```
0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0xea,0xeb,0xec,0xed  
,0xee,0xef,
```

```
0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd  
,0xfe,0xff
```

```
]
```

```
def t2():  
    return [
```

```
0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a  
,0x1c,0x1e,
```

```
0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a  
,0x3c,0x3e,
```

```
0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a  
,0x5c,0x5e,
```

```
0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a  
,0x7c,0x7e,
```

0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a
,0x9c,0x9e,

0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba
,0xbc,0xbe,

0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda
,0xdc,0xde,

0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa
,0xfc,0xfe,

0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01
,0x07,0x05,

0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21
,0x27,0x25,

0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41
,0x47,0x45,

0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61
,0x67,0x65,

0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81
,0x87,0x85,

0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1
,0xa7,0xa5,

0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1
,0xc7,0xc5,

0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1
,0xe7,0xe5

]

```
def t3():  
    return [
```

0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17
,0x12,0x11,

0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27
,0x22,0x21,

0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77
,0x72,0x71,

0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47

```

,0x42,0x41,

0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7
,0xd2,0xd1,

0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7
,0xe2,0xe1,

0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7
,0xb2,0xb1,

0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87
,0x82,0x81,

0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c
,0x89,0x8a,

0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc
,0xb9,0xba,

0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec
,0xe9,0xea,

0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc
,0xd9,0xda,

0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c
,0x49,0x4a,

0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c
,0x79,0x7a,

0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c
,0x29,0x2a,

0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c
,0x19,0x1a
    ]

def t9():
    return [

0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65
,0x7e,0x77,

0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5
,0xee,0xe7,

0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e
,0x45,0x4c,

```

0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce,
0xd5,0xdc,

0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13,
0x08,0x01,

0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83,
0x98,0x91,

0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28,
0x33,0x3a,

0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8,
0xa3,0xaa,

0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89,
0x92,0x9b,

0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19,
0x02,0x0b,

0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2,
0xa9,0xa0,

0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22,
0x39,0x30,

0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff,
0xe4,0xed,

0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f,
0x74,0x7d,

0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4,
0xdf,0xd6,

0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54,
0x4f,0x46

]

```
def t11():  
    return [
```

0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f,
0x62,0x69,

0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf,
0xd2,0xd9,

0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f,0x04,
0x19,0x12,

0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4
,0xa9,0xa2,

0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89
,0x94,0x9f,

0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39
,0x24,0x2f,

0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2
,0xef,0xe4,

0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42
,0x5f,0x54,

0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88
,0x95,0x9e,

0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38
,0x25,0x2e,

0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3
,0xee,0xe5,

0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43
,0x5e,0x55,

0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e
,0x63,0x68,

0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce
,0xd3,0xd8,

0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05
,0x18,0x13,

0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5
,0xa8,0xa3

]

```
def t13():  
    return [
```

0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51
,0x46,0x4b,

0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81
,0x96,0x9b,

0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea

```

,0xfd,0xf0,

0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a
,0x2d,0x20,

0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c
,0x2b,0x26,

0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec
,0xfb,0xf6,

0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87
,0x90,0x9d,

0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57
,0x40,0x4d,

0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b
,0x9c,0x91,

0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b
,0x4c,0x41,

0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30
,0x27,0x2a,

0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0
,0xf7,0xfa,

0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6
,0xf1,0xfc,

0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36
,0x21,0x2c,

0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d
,0x4a,0x47,

0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d
,0x9a,0x97
    ]

def t14():
    return [

0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46
,0x54,0x5a,

0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6
,0xb4,0xba,

```



```

0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d
,0x8f,0x81,

0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d
,0x6f,0x61,

0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb
,0xf9,0xf7,

0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b
,0x19,0x17,

0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30
,0x22,0x2c,

0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0
,0xc2,0xcc,

0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07
,0x15,0x1b,

0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7
,0xf5,0xfb,

0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc
,0xce,0xc0,

0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c
,0x2e,0x20,

0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa
,0xb8,0xb6,

0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a
,0x58,0x56,

0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71
,0x63,0x6d,

0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f,0x91
,0x83,0x8d
]

```

Fungsi "t1", "t2", dan "t3" adalah konstanta yang digunakan untuk melakukan enkripsi dan dekripsi dalam algoritma AES. Konstanta "t2" dan "t3" digunakan dalam proses enkripsi "mix columns", di mana "t2" digunakan untuk mengalikan kolom oleh 2 dan "t3" untuk mengalikan kolom oleh 3. Sedangkan konstanta "t9", "t11", "t13", dan "t14" dipakai dalam proses dekripsi "inverse mix columns". "t9" digunakan untuk mengalikan kolom oleh 9, "t11" untuk 11, "t13" untuk 13, dan "t14" untuk 14. Hal ini merupakan proses memanfaatkan operasi XOR, shift, dan pengalihan dalam langkah-langkah enkripsi dan dekripsi data.

Tabel rcon

```
def rcon():  
    return [  
        [0x01, 0x00, 0x00, 0x00],  
        [0x02, 0x00, 0x00, 0x00],  
        [0x04, 0x00, 0x00, 0x00],  
        [0x08, 0x00, 0x00, 0x00],  
        [0x10, 0x00, 0x00, 0x00],  
        [0x20, 0x00, 0x00, 0x00],  
        [0x40, 0x00, 0x00, 0x00],  
        [0x80, 0x00, 0x00, 0x00],  
        [0x1b, 0x00, 0x00, 0x00],  
        [0x36, 0x00, 0x00, 0x00]  
    ]
```

Fungsi "rcon" mengembalikan sebuah matriks 2D yang merepresentasikan nilai-nilai rcon (round constants) yang digunakan dalam proses ekspansi kunci. Rcon digunakan untuk menambahkan kompleksitas dan non-linearitas pada proses ekspansi kunci. Matriks yang dihasilkan memiliki 10 baris dan 4 kolom, dengan setiap elemen mewakili nilai rcon yang akan digunakan pada setiap putaran kunci (round key) dari ronde 1 hingga 10. Setiap nilai rcon disusun dalam format little-endian, dengan byte pertama memiliki nilai yang berubah pada setiap putaran, sementara byte kedua hingga keempat diatur menjadi nol.

Fungsi Padding Data

```
def pad_data(input_data, block_size=16):  
    padded_data = input_data.copy()  
    padding = block_size - (len(input_data) % block_size)  
    if padding != block_size:  
        padded_data.extend([padding] * padding)  
    return padded_data
```

Fungsi tersebut adalah fungsi untuk melakukan padding pada data. Padding adalah proses penambahan nilai tambahan (biasanya nol) ke data sehingga panjang data menjadi kelipatan dari suatu ukuran blok tertentu. Fungsi akan mengambil data input dan menambahkan padding ke dalamnya sehingga panjang data setelah padding menjadi kelipatan dari block_size. Selanjutnya, fungsi memeriksa apakah padding diperlukan. Jika nilai padding tidak sama dengan ukuran blok (padding != block_size), berarti padding perlu ditambahkan

Fungsi Unpadding Data

```
def unpad_data(data):  
    padding = data[-1]  
    if padding > 0 and all(x == padding for x in data[-padding:]):
```

```
    return data[:-padding]
else:
    return data
```

Fungsi ini digunakan untuk menghilangkan padding yang ada pada data yang diberikan. Pertama, fungsi mengambil nilai padding terakhir dari data yang diberikan. Kemudian, dengan menggunakan kondisi, fungsi memeriksa apakah nilai padding tersebut lebih dari 0 dan apakah semua nilai di bagian terakhir data sesuai dengan nilai padding yang diidentifikasi. Jika kedua kondisi terpenuhi, fungsi akan mengembalikan data tanpa bagian padding, yaitu data sebelum bagian terakhir yang merupakan padding. Namun, jika salah satu dari kondisi tersebut tidak terpenuhi, fungsi akan mengembalikan data tanpa mengubahnya. Hal ini bertujuan untuk memastikan bahwa hanya data dengan padding yang valid yang akan dihilangkan padding-nya, sementara data tanpa padding atau dengan padding yang tidak valid akan tetap utuh.

Fungsi Split Data

```
def split_data(input_data, block_size=16):
    result = [input_data[i:i + block_size] for i in range(0,
len(input_data), block_size)]
    return result
```

Fungsi `split_data` digunakan untuk membagi data input menjadi potongan-potongan dengan ukuran yang telah ditentukan, di mana nilai default untuk ukuran blok adalah 16. Fungsi ini menerima dua parameter, yaitu `input_data`, yang merupakan data yang akan dibagi, dan `block_size`, yang menyatakan ukuran blok yang digunakan dalam pembagian data. Dalam implementasinya, fungsi menggunakan list comprehension untuk menghasilkan sebuah list yang berisi potongan-potongan data dengan ukuran sesuai yang ditentukan oleh `block_size`. Proses pembagian data dilakukan dengan menggunakan loop yang memotong data input mulai dari indeks 0 hingga panjang data input dengan selang sepanjang `block_size`. Hasilnya, fungsi mengembalikan list yang berisi potongan-potongan data tersebut sebagai output.

Fungsi Generate Random String

```
def generate_random_string(length):
    import random
    import string
    return ''.join(random.choices(string.ascii_letters +
string.digits, k=length))
```

Fungsi tersebut digunakan untuk menciptakan string acak yang kemudian akan terpanggil jika file yang ingin di enkripsi atau dekripsi tidak memiliki nama header file.

Fungsi File to Hex

```
def file_to_hex(path, header=False):
    header_list = []
```

```

if header == True:
    import os
    file_name = os.path.basename(path)
    header = f'header<<{file_name}>>'
    header_list = [ord(char) for char in header]

byte_data = ""

with open(path, 'rb') as file:
    byte_data = file.read()

return header_list + list(bytearray(byte_data))

```

Fungsi ini bertujuan untuk mengonversi data dari sebuah file ke dalam format heksadesimal. Saat dipanggil, fungsi menerima argumen path yang merupakan path dari file yang ingin dikonversi. Terdapat parameter opsional 'header' dengan default yaitu False. Ketika 'header' diatur sebagai True, maka fungsi akan membuat sebuah header dengan nama file yang diberikan dalam format tertentu dan mengonversinya menjadi nilai ASCII menggunakan 'ord()'. Fungsi kemudian membaca data biner dari file yang diberikan menggunakan 'open()' dengan mode 'rb' (read binary), dan mengubahnya ke dalam bentuk list byte menggunakan 'bytearray()'. Fungsi ini akan mengembalikan gabungan antara header_list (jika header=True) dan list byte dari data file yang telah dikonversi ke dalam heksadesimal.

Fungsi Hex to File

```

def hex_to_file(hex):
    path = ""
    start_list = [104, 101, 97, 100, 101, 114, 60, 60]
    if hex[:len(start_list)] == start_list:
        end_index = None
        for i in range(len(hex) - 1):
            if hex[i] == 62 and hex[i + 1] == 62:
                end_index = i
                break
        if end_index is not None:
            hex_path = hex[len(start_list):end_index]
            path = ''.join(chr(char) for char in hex_path)
            hex = hex[end_index+2:]
    else:
        path = generate_random_string(10)

    with open(path, 'wb') as file:
        file.write(bytes(hex))
        print(f'File successfully saved with name: {path}')

```

Fungsi hex_to_file melakukan konversi data heksadesimal ke file biner dan menyimpannya dengan nama file yang berasal dari sebagian data heksadesimal tersebut. Pertama, fungsi mengecek apakah awalan data heksadesimal sesuai dengan nilai yang ada di start_list. Jika cocok, fungsi mencari indeks di mana urutan karakter >> muncul, menandakan akhir dari path

file dalam bentuk heksadesimal. Setelah menemukan indeks tersebut, fungsi mengambil sebagian data heksadesimal yang mencakup path file, mengonversinya ke bentuk string, dan menyimpannya di dalam variabel path. Jika kondisi awal tidak terpenuhi, fungsi menggunakan fungsi `generate_random_string(10)` untuk menghasilkan nama file acak dengan panjang 10 karakter. Selanjutnya, fungsi membuka file dengan nama yang telah ditentukan (baik itu hasil konversi path heksadesimal atau nama file acak) dengan mode write binary ('wb'). Data heksadesimal yang tersisa setelah path file diambil dan diubah menjadi string, diubah ke dalam bentuk byte dengan menggunakan fungsi `bytes(hex)`, dan ditulis ke dalam file tersebut. Terakhir, fungsi mencetak pesan yang memberitahu bahwa file berhasil disimpan, bersama dengan nama file yang digunakan.

Fungsi State

```
def state(plain):  
    state = [  
        [plain[0], plain[4], plain[8], plain[12]],  
        [plain[1], plain[5], plain[9], plain[13]],  
        [plain[2], plain[6], plain[10], plain[14]],  
        [plain[3], plain[7], plain[11], plain[15]]  
    ]  
    return state
```

Fungsi ini menerima inputan berupa array satu dimensi dan fungsi akan mengubahnya menjadi matriks 4x4 secara berurutan.

Fungsi Unstate Matrix

```
def unstate(matrix):  
    unstate_list = [matrix[row][col] for col in range(len(matrix[0]))  
for row in range(len(matrix))]  
    return unstate_list
```

Fungsi ini menerima matriks sebagai argumen input dan melakukan proses unstate atau menata ulang matriks tersebut ke dalam bentuk list satu dimensi. Dalam fungsi ini, setiap elemen dari matriks dimasukkan ke dalam list secara berurutan, diambil per baris, dari kiri ke kanan. Fungsi ini akan mengeluarkan sebuah list satu dimensi yang berisi nilai-nilai dari matriks tersebut dalam urutan yang sesuai dengan penataan baris dan kolomnya.

Fungsi Add Round Key

```
def add_round_key(state, key):  
    added_state = [[state[i][j] ^ key[i][j] for j in range(4)] for i  
in range(4)]  
    return added_state
```

Fungsi ini menerima dua parameter, yaitu state yang merepresentasikan keadaan blok saat ini dalam proses enkripsi, dan key yang merupakan kunci putaran yang akan digunakan. Operasi

ini dilakukan dengan melakukan operasi XOR (bitwise exclusive OR) antara setiap elemen blok state dan elemen yang sesuai dalam kunci putaran (key). Hasilnya disimpan dalam sebuah matriks baru yang disebut `added_state`. Matriks `added_state` ini kemudian dikembalikan sebagai hasil dari fungsi.

Fungsi Sub Bytes

```
def sub_bytes(state, sbox):  
    subs_state = [[sbox[val >> 4][val & 0xF] for val in row] for row  
in state]  
    return subs_state
```

Fungsi ini merupakan SubBytes dalam algoritma enkripsi Advanced Encryption Standard (AES). Langkah SubBytes dilakukan pada blok data 4x4 dalam proses enkripsi AES. Fungsi ini menerima dua parameter: `state`, yang merupakan blok data 4x4 yang akan dienkripsi, dan `sbox`, yang merupakan tabel substitusi (substitution box) yang digunakan untuk menggantikan setiap elemen dalam blok data dengan nilai yang sesuai dari tabel tersebut.

Pada setiap iterasi, fungsi akan melakukan substitusi pada setiap elemen dalam blok data `state` dengan menggunakan nilai yang terdapat dalam `sbox`. Elemen blok data diakses satu per satu, dan nilai masing-masing elemen dibagi menjadi dua bagian: empat bit pertama dan empat bit terakhir. Bagian pertama digunakan sebagai indeks baris, sedangkan bagian kedua digunakan sebagai indeks kolom dalam tabel substitusi `sbox`. Nilai yang diambil dari `sbox` tersebut kemudian digunakan untuk menggantikan nilai asli elemen tersebut dalam blok data.

Nama fungsi

```
def shift_rows(state, direction):  
    shifted_state = [[0] * 4 for _ in range(4)]  
  
    for row in range(4):  
        # untuk dekripsi  
        if direction == 1:  
            shifted_state[row] = state[row][-row:] + state[row][:  
row]  
        # untuk enkripsi  
        elif direction == -1:  
            shifted_state[row] = state[row][row:] + state[row][:row]  
  
    return shifted_state
```

Fungsi ini bertujuan untuk melakukan pergeseran baris pada matriks `'state'` sesuai arah yang ditentukan (`'direction'`). Pertama, fungsi membuat matriks kosong `'shifted_state'` yang memiliki ukuran 4x4. Selanjutnya, dengan menggunakan loop `'for'`, fungsi melakukan pergeseran baris berdasarkan nilai `'direction'`. Jika `'direction'` adalah 1 (untuk dekripsi), setiap baris dari `'state'` akan digeser ke kanan sebanyak `'row'` langkah, sehingga bagian belakang akan dipindahkan ke bagian depan. Sedangkan jika `'direction'` adalah -1 (untuk enkripsi), setiap baris akan digeser ke kiri sebanyak `'row'` langkah, sehingga bagian depan akan dipindahkan ke bagian belakang.

Hasil dari pergeseran ini akan disimpan dalam 'shifted_state' yang kemudian dikembalikan sebagai output dari fungsi ini.

Fungsi Mix Columns

```
def mix_columns(state, t1, t2, t3, t4):
    mixed_state = [[0] * 4 for _ in range(4)]

    # enkripsi 02 03 01 01
    # dekripsi 14 11 13 09
    for c in range(4):
        mixed_state[0][c] = t1[state[0][c]] ^ t2[state[1][c]] ^
        t3[state[2][c]] ^ t4[state[3][c]]
        mixed_state[1][c] = t4[state[0][c]] ^ t1[state[1][c]] ^
        t2[state[2][c]] ^ t3[state[3][c]]
        mixed_state[2][c] = t3[state[0][c]] ^ t4[state[1][c]] ^
        t1[state[2][c]] ^ t2[state[3][c]]
        mixed_state[3][c] = t2[state[0][c]] ^ t3[state[1][c]] ^
        t4[state[2][c]] ^ t1[state[3][c]]

    return mixed_state
```

Fungsi ini menerima lima parameter, yaitu "state" yang merepresentasikan keadaan blok saat ini dalam proses enkripsi, dan empat parameter lainnya ("t1", "t2", "t3", "t4") yang digunakan sebagai tabel konstanta untuk operasi MixColumns. Operasi MixColumns melibatkan transformasi matriks "state" dengan matriks konstanta tertentu. Dalam implementasi ini, operasi dilakukan pada setiap kolom matriks "state". Setiap elemen baru dalam matriks hasil ("mixed_state") dihitung dengan menggabungkan operasi XOR dan penggantian byte (substitution) pada tiap baris menggunakan tabel konstanta yang sesuai. Proses ini dilakukan untuk setiap elemen matriks "state" dalam setiap kolom. Hasil akhirnya adalah matriks "mixed_state" yang mewakili keadaan blok setelah operasi MixColumns diterapkan.

Fungsi G

```
def g_function(list, rcon_index, sbox=sbox(), rcon=rcon()):
    shift = list[1:] + [list[0]]
    subs = [sbox[val >> 4][val & 0xF] for val in shift]
    added = [rcon[rcon_index][i] ^ subs[i] for i in range(4)]
    return added
```

Fungsi ini menerima tiga parameter, yaitu list, rcon_index, sbox, dan rcon kemudian fungsi ini melakukan operasi penggeseran (shift) pada elemen-elemen dalam list. Setiap elemen dipindahkan ke posisi berikutnya, dan elemen terakhir dipindahkan ke posisi pertama. Operasi ini menciptakan array yang diwakili oleh variabel shift. Selanjutnya, fungsi melakukan substitusi byte (byte substitution) pada array shift dengan menggunakan kotak substitusi (s-box) yang diberikan melalui parameter opsional sbox. Substitusi ini dilakukan untuk setiap nilai byte dalam array, dan hasilnya disimpan dalam array yang diwakili oleh variabel subs. Setelah itu, fungsi

melakukan operasi bitwise XOR (exclusive OR) antara setiap elemen dari array substitusi (subs) dan elemen pada rcon sesuai dengan indeks rcon_index. Rcon (round constant) adalah array konstan yang digunakan dalam proses kunci ekspansi pada setiap putaran kunci. Hasil operasi XOR ini disimpan dalam array yang diwakili oleh variabel added.

Fungsi Expand Key

```
def expand_key(key):
    st = state(key)
    added_st = [[0] * 4 for _ in range(4)]

    key_list = [st]

    for i in range(10):
        g_column = g_function([st[r][3] for r in range(4)], i)
        for r in range(4):
            added_st[r][3] = g_column[r]
        for c in range(4):
            for r in range(4):
                added_st[r][c] = st[r][c] ^ added_st[r][c-1]
        st = [row.copy() for row in added_st]
        key_list.append(st)

    return key_list
```

Fungsi ini bertujuan untuk melakukan proses *key expansion* pada kunci yang diberikan. Awalnya, fungsi mengonversi kunci ke dalam bentuk matriks dengan fungsi 'state()'. Kemudian, dibuat matriks tambahan 'added_st' yang diisi dengan nilai nol dan sebuah list 'key_list' yang akan menyimpan setiap iterasi dari kunci yang diekspansi. Melalui iterasi sebanyak 10 kali, fungsi melakukan operasi g-function dengan parameter kolom terakhir dari 'st' untuk menghasilkan nilai baru yang digunakan dalam proses ekspansi kunci. Setelah itu, dilakukan operasi XOR antara 'st' dan 'added_st' untuk menghasilkan nilai baru yang akan menjadi bagian dari kunci yang diekspansi. Hasil akhir dari setiap iterasi disimpan dalam 'key_list', yang berisi kunci yang diekspansi dalam setiap langkah iterasi, dan akan dikembalikan sebagai output dari fungsi ini.

Fungsi Enkripsi Block

```
def block_encrypt(plain, list_key):
    st = state(plain)

    rk = add_round_key(st, list_key[0])

    for i in range(1, 10):
        sb = sub_bytes(rk, sbox())
        sr = shift_rows(sb, -1)
        mc = mix_columns(sr, t2(), t3(), t1(), t1())
```



```

        rk = add_round_key(mc, list_key[i])

    sb = sub_bytes(rk, sbx())
    sr = shift_rows(sb, -1)
    rk = add_round_key(sr, list_key[10])

    cipher = unstate(rk)
    return cipher

```

Fungsi "block_encrypt" ini digunakan untuk melakukan enkripsi blok data menggunakan algoritma Advanced Encryption Standard (AES). Fungsi menerima dua parameter, yaitu "plain" yang merupakan blok data plaintext yang akan dienkripsi, dan "list_key" yang berisi kunci-kunci putaran yang digunakan selama proses enkripsi. Pertama, fungsi menginisialisasi keadaan awal "st" dengan menggunakan fungsi "state" pada blok plaintext yang diberikan. Selanjutnya, kunci putaran pertama ("rk") dihasilkan dengan menjalankan fungsi "add_round_key" menggunakan keadaan awal "st" dan kunci putaran pertama dari "list_key". Proses iteratif dilakukan sebanyak sembilan putaran enkripsi dengan menggunakan langkah-langkah substitusi byte ("sub_bytes"), pergeseran baris ("shift_rows"), mix columns ("mix_columns"), dan penambahan kunci putaran ("add_round_key"). Setelah sembilan putaran, langkah terakhir dilakukan tanpa operasi mix columns. Setelah kesepuluh putaran, keadaan akhir dihasilkan dari substitusi byte, pergeseran baris, dan penambahan kunci terakhir. Keadaan akhir ini kemudian diubah kembali menjadi bentuk blok data yang dienkripsi menggunakan fungsi "unstate", dan hasilnya dikembalikan sebagai blok data ciphertext. Dengan demikian, fungsi "block_encrypt" secara keseluruhan merupakan implementasi lengkap dari algoritma enkripsi AES pada satu blok data.

Fungsi Block Decrypt

```

def block_decrypt(cipher, list_key):
    st = state(cipher)
    rk = add_round_key(st, list_key[10])
    sr = shift_rows(rk, 1)
    sb = sub_bytes(sr, inv_sbx())

    for i in range(9, 0, -1):
        rk = add_round_key(sb, list_key[i])
        mc = mix_columns(rk, t14(), t11(), t13(), t9())
        sr = shift_rows(mc, 1)
        sb = sub_bytes(sr, inv_sbx())

    rk = add_round_key(sb, list_key[0])

    plain = unstate(rk)
    return plain

```

Fungsi menerima dua parameter, yaitu cipher yang merupakan blok cipher yang akan didekripsi, dan list_key yang berisi kunci-kunci ronde yang akan digunakan selama proses

dekripsi. Pertama, pembuat objek state `st` dari blok cipher menggunakan fungsi `state(cipher)`. Selanjutnya, kunci ronde terakhir (`list_key[10]`) ditambahkan ke state menggunakan operasi `add_round_key`. Kemudian, dilakukan operasi shift pada baris-state (`shift_rows`) dengan parameter 1, dan dilanjutkan dengan substitusi byte terbalik (`sub_bytes`) menggunakan invers dari S-box (`inv_sbox()`). Selanjutnya, dilakukan iterasi dari ronde ke-9 hingga ronde ke-1. Pada setiap iterasi, kunci ronde (`list_key[i]`) ditambahkan ke state, kemudian dilakukan operasi mix columns (`mix_columns`) dengan matriks transformasi (`t14()`, `t11()`, `t13()`, `t9()`). Setelah itu, dilakukan shift rows dan substitusi byte terbalik seperti pada ronde terakhir. Setelah iterasi selesai, kunci ronde pertama (`list_key[0]`) ditambahkan ke state hasil iterasi terakhir. Terakhir, state tersebut diubah kembali menjadi bentuk blok teks biasa menggunakan fungsi `unstate(rk)` dan hasilnya dikembalikan sebagai blok teks terdekripsi.

Fungsi AES Encrypt

```
def aes_encrypt(plain, key):
    keys = expand_key(key)
    chunks = split_data(pad_data(plain))

    cipher = []

    for chunk in chunks:
        encrypted_chunk = block_encrypt(chunk, keys)
        cipher.extend(encrypted_chunk)

    return cipher
```

Fungsi ini berfungsi untuk melakukan enkripsi AES pada teks 'plain' menggunakan kunci 'key' yang diberikan. Pertama, fungsi melakukan ekspansi kunci dengan menggunakan `'expand_key(key)'` untuk mendapatkan serangkaian kunci yang akan digunakan selama proses enkripsi. Selanjutnya, teks 'plain' dipad dan dibagi menjadi blok-blok data menggunakan `'pad_data()'` dan `'split_data()'`. Setelah itu, fungsi melakukan enkripsi pada setiap blok data menggunakan fungsi `'block_encrypt()'` dengan menggunakan serangkaian kunci yang telah dihasilkan. Blok-blok data yang telah dienkripsi kemudian digabungkan menjadi satu kesatuan 'cipher' yang akan dikembalikan sebagai hasil enkripsi dari teks 'plain' menggunakan AES dengan kunci yang diberikan.

Fungsi Dekripsi

```
def aes_decrypt(cipher, key):
    keys = expand_key(key)
    chunks = split_data(cipher)

    plain = []

    for chunk in chunks:
        decrypted_chunk = block_decrypt(chunk, keys)
        plain.extend(decrypted_chunk)
```

```
return unpad_data(plain)
```

Fungsi "aes_decrypt" digunakan untuk mendekripsi data ciphertext yang telah dienkripsi. Fungsi ini menerima dua parameter, yaitu "cipher" yang merupakan data ciphertext yang akan didekripsi, dan "key" yang merupakan kunci rahasia yang digunakan dalam proses dekripsi. Pertama, fungsi meng-expand kunci rahasia ("key") menjadi serangkaian kunci menggunakan fungsi "expand_key". Selanjutnya, data ciphertext ("cipher") dibagi menjadi potongan-potongan dengan menggunakan fungsi "split_data". Proses dekripsi dilakukan pada setiap potongan ciphertext dalam loop. Setiap potongan didekripsi menggunakan fungsi "block_decrypt", yang sebagian besar mengikuti langkah-langkah kebalikan dari langkah-langkah enkripsi AES. Hasil dekripsi dari setiap potongan ciphertext ditambahkan ke dalam daftar "plain". Setelah seluruh potongan ciphertext telah didekripsi, data plaintext hasil dekripsi diubah kembali menjadi bentuk semula dengan menggunakan fungsi "unpad_data", dan hasilnya dikembalikan oleh fungsi sebagai output.

Fungsi Nilai Acak

```
def generate_random(bit):  
    import random  
    return random.getrandbits(bit)
```

Fungsi tersebut digunakan untuk menghasilkan nilai acak berdasarkan jumlah bit yang ditentukan oleh argumen bit. Semakin besar nilai bit, semakin besar pula rentang nilai acak yang mungkin dihasilkan. Fungsi ini digunakan untuk menghasilkan nilai private key acak

Fungsi Check Prima

```
def is_prime(n, k=32):  
    import random  
  
    if n <= 1:  
        return False  
    if n <= 3:  
        return True  
    if n % 2 == 0:  
        return False  
    r, d = 0, n - 1  
    while d % 2 == 0:  
        r += 1  
        d //= 2  
    for _ in range(k):  
        a = random.randint(2, n - 2)  
        x = pow(a, d, n)  
        if x == 1 or x == n - 1:  
            continue  
        for _ in range(r - 1):  
            x = pow(x, 2, n)
```

```

        if x == n - 1:
            break
    else:
        return False
return True

```

Fungsi ini digunakan untuk melakukan pengecekan apakah bilangan 'n' merupakan bilangan prima atau bukan. Pertama, fungsi mengecek kondisi khusus seperti jika 'n' kurang dari atau sama dengan 1, maka langsung dikembalikan nilai False. Jika 'n' bernilai 2 atau 3, fungsi akan mengembalikan nilai True karena keduanya merupakan bilangan prima. Kemudian, dilakukan pengujian untuk mengecek apakah 'n' habis dibagi 2. Jika 'n' genap (habis dibagi 2), maka langsung dikembalikan nilai False karena bilangan genap selain 2 bukan bilangan prima. Selanjutnya, fungsi melakukan pengujian Miller-Rabin dengan menggunakan 'k' iterasi yang secara acak memilih bilangan 'a' dan melakukan perhitungan pada nilai 'x'. Jika nilai 'x' tidak memenuhi kriteria pada pengujian, maka 'n' dipastikan bukan bilangan prima dan dikembalikan nilai False. Jika selama proses pengujian tidak ditemukan bukti bahwa 'n' bukan prima, maka fungsi mengembalikan nilai True, menandakan bahwa 'n' merupakan bilangan prima berdasarkan pengujian Miller-Rabin.

Fungsi Generate Prima

```

def generate_prime(bit_length):
    import random

    while True:
        candidate = random.getrandbits(bit_length)
        candidate |= (1 << bit_length - 1) | 1
        if is_prime(candidate):
            return candidate

```

Fungsi "generate_prime" bertujuan untuk menghasilkan bilangan prima dengan panjang bit tertentu. Fungsi menggunakan library "random" untuk menghasilkan bilangan bulat acak dengan jumlah bit yang ditentukan oleh parameter "bit_length". Selama perulangan tak terbatas, fungsi membangun kandidat bilangan prima dengan mengatur bit paling signifikan (MSB) dan bit paling kurang signifikan (LSB) untuk memastikan bahwa bilangan tersebut ganjil dan memiliki panjang bit yang diinginkan. Setelah itu, fungsi memeriksa apakah kandidat tersebut merupakan bilangan prima dengan memanggil fungsi "is_prime". Jika benar, kandidat tersebut dikembalikan sebagai hasil dari fungsi. Fungsi ini menggabungkan pendekatan pembangkitan bilangan prima dengan pengujian primalitas untuk memastikan bahwa bilangan yang dihasilkan adalah prima. Pendekatan pembangkitan menggunakan bit-wise operations untuk mengatur bit-bit tertentu pada kandidat bilangan, sementara pengujian primalitas dilakukan dengan memanggil fungsi "is_prime".

Fungsi Modulus Eksponen

```

def mod_exp(base, exponent, modulus):
    result = 1

```

```

base = base % modulus
while exponent > 0:
    if exponent % 2 == 1:
        result = (result * base) % modulus
    base = (base * base) % modulus
    exponent = exponent // 2
return result

```

Fungsi menginisialisasi variabel result dengan nilai 1. Kemudian, variabel base diubah menjadi sisa hasil bagi base terhadap modulus, memastikan bahwa nilai base tetap dalam batas modulus. Selanjutnya, dilakukan iterasi menggunakan loop while dengan kondisi bahwa exponent lebih besar dari 0. Dalam setiap iterasi, dilakukan pengecekan apakah exponent ganjil atau genap. Jika exponent ganjil, maka nilai result akan dikalikan dengan base dan diambil sisa hasil bagi terhadap modulus. Selanjutnya, nilai base diperbarui menjadi kuadrat dari nilai sebelumnya, juga diambil sisa hasil bagi terhadap modulus. Terakhir, eksponen dibagi dua setiap iterasi dengan menggunakan operasi floor division (/). Proses ini berlanjut hingga eksponen mencapai nilai 0, dan hasil akhir dari perhitungan eksponensial modulo disimpan dalam variabel result, yang kemudian dikembalikan sebagai output dari fungsi.

Fungsi Int to Hex

```

def int_to_hex(decimal):
    hex_string = hex(decimal)[2:]

    if len(hex_string) > 32:
        hex_string = hex_string[:32]

    while len(hex_string) < 32:
        hex_string = '0' + hex_string

    byte_list = [hex_string[i:i+2] for i in range(0, len(hex_string),
2)]

    hex_list = [int(byte, 16) for byte in byte_list]

    return hex_list

```

Fungsi ini bertujuan untuk mengonversi bilangan desimal menjadi representasi heksadesimal dalam bentuk list bilangan bulat. Pertama, fungsi mengonversi bilangan desimal menjadi string heksadesimal menggunakan fungsi 'hex()', kemudian membuang awalan '0x' dengan mengambil substring dari indeks ke-2 ke depan. Fungsi kemudian memastikan bahwa panjang string heksadesimal tidak melebihi 32 karakter dengan memotongnya jika panjangnya lebih dari 32 karakter. Selanjutnya, dilakukan penambahan nol pada awal string heksadesimal sampai panjangnya mencapai 32 karakter. Setelah itu, string heksadesimal dibagi menjadi byte-byte dua karakter, dan setiap byte dikonversi kembali menjadi bilangan bulat dengan basis 16 menggunakan fungsi 'int()' dan argumen basis 16. Hasil akhir dari konversi ini adalah list yang berisi bilangan bulat yang merepresentasikan nilai heksadesimal dari bilangan desimal yang diberikan.

Fungsi DHKE

```
def DHKE(p, g, x):  
    y = mod_exp(g, x, p)  
    return y  
  
p = generate_prime(128)  
g = generate_prime(16)  
print("p: ", p)  
print("g: ", g)
```

Fungsi "DHKE" (Diffie-Hellman Key Exchange) menerapkan algoritma pertukaran kunci Diffie-Hellman. Fungsi ini menerima tiga parameter, yaitu bilangan prima "p", generator "g", dan eksponen pribadi "x". Dalam algoritma Diffie-Hellman, "y" dihitung dengan mengambil nilai "g" dipangkatkan pada eksponen "x" modulo "p". Nilai "y" ini kemudian dapat ditukar dengan pihak lain dalam suatu protokol pertukaran kunci. Pada potongan kode selanjutnya, bilangan prima ("p") dihasilkan menggunakan fungsi "generate_prime" dengan panjang bit 128, sementara generator ("g") dihasilkan dengan panjang bit 16.

Generate Private Key

```
private_key_a = generate_random(128)  
private_key_b = generate_random(128)  
print("Private Key A: ", private_key_a)  
print("Private Key B: ", private_key_b)
```

Kode tersebut akan memanggil fungsi generate_random untuk menghasilkan nilai acak berdasarkan sejumlah 128 bit untuk nilai private key a dan nilai private key b

Generate Public Key

```
public_key_a = DHKE(p, g, private_key_a)  
public_key_b = DHKE(p, g, private_key_b)  
print("Public Key A: ", public_key_a)  
print("Public Key B: ", public_key_b)
```

Kode ini melakukan pertukaran kunci Diffie-Hellman antara dua entitas dengan menggunakan parameter 'p' dan 'g' sebagai bagian dari proses pembuatan kunci. Pertama, kode menghasilkan kunci publik untuk entitas pertama dengan menggunakan nilai 'p', 'g', dan kunci privat 'private_key_a' yang dimilikinya. Kemudian, hal yang sama dilakukan untuk entitas kedua menggunakan kunci privat 'private_key_b'. Hasil dari kunci publik untuk kedua entitas kemudian ditampilkan melalui fungsi 'print()' untuk 'public_key_a' dan 'public_key_b'. Ini memungkinkan entitas tersebut untuk bertukar informasi kunci publik yang nantinya akan digunakan dalam pembentukan kunci bersama untuk komunikasi aman.

Generate Shared Key

```
shared_key_a = DHKE(p, public_key_b, private_key_a)
shared_key_b = DHKE(p, public_key_a, private_key_b)
print("Shared Key A: ", shared_key_a)
print("Shared Key B: ", shared_key_b)
```

Kode mengimplementasikan pertukaran kunci Diffie-Hellman antara dua entitas (A dan B) menggunakan parameter yang telah dihasilkan sebelumnya. Pada pertukaran kunci ini, setiap entitas memiliki pasangan kunci pribadi-publik. Nilai "shared_key_a" dihitung oleh entitas A dengan memanggil fungsi "DHKE" menggunakan bilangan prima "p", kunci publik entitas B ("public_key_b"), dan kunci pribadi A ("private_key_a"). Demikian juga, nilai "shared_key_b" dihitung oleh entitas B dengan memanggil fungsi "DHKE" menggunakan bilangan prima "p", kunci publik entitas A ("public_key_a"), dan kunci pribadi B ("private_key_b"). Hasilnya, kedua nilai shared key ("shared_key_a" dan "shared_key_b") akan dicetak untuk menunjukkan bahwa keduanya telah berhasil berbagi kunci dengan menggunakan protokol pertukaran kunci Diffie-Hellman. Shared key ini dapat digunakan sebagai dasar untuk mengamankan komunikasi selanjutnya antara kedua entitas tanpa perlu mengirimkan kunci pribadi melalui jaringan.

Enkripsi File

```
key = int_to_hex(shared_key_a)

byte_data = file_to_hex("tugaskripto.pdf", True)
cipher = aes_encrypt(byte_data, key)

hex_to_file(cipher)
```

Kode ini melakukan serangkaian operasi. Pertama, mengonversi kunci bersama 'shared_key_a' menjadi representasi heksadesimal menggunakan fungsi 'int_to_hex()'. Kemudian, kode membaca data dari file dengan nama "tugaskripto.pdf" dan mengkonversinya menjadi representasi heksadesimal menggunakan 'file_to_hex()'. Selanjutnya, melakukan enkripsi AES pada data heksadesimal 'byte_data' menggunakan kunci yang telah dikonversi sebelumnya 'key' dengan fungsi 'aes_encrypt()'. Terakhir, hasil enkripsi tersebut diubah kembali menjadi file cipher dalam bentuk heksadesimal menggunakan 'hex_to_file()'. Ini adalah serangkaian langkah untuk membaca, menenkripsi, dan memproses kembali data dalam bentuk heksadesimal menjadi file cipher.

Dekripsi File

```
key = int_to_hex(shared_key_b)

byte_data = file_to_hex("AMtONFVBgV")
plain = aes_decrypt(byte_data, key)

hex_to_file(plain)
```

Kode ini melakukan serangkaian operasi. Pertama, mengonversi kunci bersama 'shared_key_b' menjadi representasi heksadesimal menggunakan fungsi 'int_to_hex()'. Kemudian, kode membaca data dari file dengan nama "AMtONFVBgV" dan mengonversinya menjadi representasi heksadesimal menggunakan 'file_to_hex()'. Selanjutnya, melakukan dekripsi AES pada data heksadesimal 'byte_data' menggunakan kunci yang telah dikonversi sebelumnya 'key' dengan fungsi 'aes_decrypt()'. Terakhir, hasil dekripsi tersebut diubah kembali menjadi file aslinya dalam bentuk heksadesimal menggunakan 'hex_to_file()'. Ini adalah serangkaian langkah untuk membaca, mendekripsi, dan memproses kembali data dalam bentuk heksadesimal menjadi file yang dimaksud.

8. Jelaskan secara detail kode program Anda secara khusus terkait algoritma yang Anda gunakan? (tutorial step-by-step from scratch)

Tabel SBOX & Inverse

```
def sbbox():
    return [
        [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,
        0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76],
        [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4,
        0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0],
        [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5,
        0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15],
        [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12,
        0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75],
        [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b,
        0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84],
        [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
        0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf],
        [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9,
        0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8],
        [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6,
        0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2],
        [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
        0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73],
        [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee,
        0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb],
        [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3,
        0xac, 0x62, 0x91, 0x95, 0xe4, 0x79],
        [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56,
        0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08],
        [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd,
        0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a],
        [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35,
        0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e],
        [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,
        0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf],
        [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99,
        0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]
    ]
```



```

def inv_sbox():
    return [
        [0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40,
0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb],
        [0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e,
0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb],
        [0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c,
0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e],
        [0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b,
0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25],
        [0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4,
0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92],
        [0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15,
0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84],
        [0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4,
0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06],
        [0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf,
0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b],
        [0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2,
0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73],
        [0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9,
0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e],
        [0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7,
0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b],
        [0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb,
0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4],
        [0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12,
0x10, 0x59, 0x27, 0x80, 0xec, 0x5f],
        [0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5,
0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef],
        [0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb,
0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61],
        [0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69,
0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d]
    ]

```

Fungsi "sbox()" mengembalikan matriks substitusi (S-Box). Matriks ini terdiri dari elemen-elemen heksadesimal yang menentukan penggantian byte-byte masukan pada proses enkripsi. Setiap elemen matriks S-Box adalah hasil substitusi dari 8-bit binary input ke 8-bit binary output. Sebaliknya, fungsi "inv_sbox()" mengembalikan matriks substitusi invers (Inverse S-Box) yang digunakan dalam tahap dekripsi AES. Matriks ini digunakan untuk mengembalikan byte-byte ke nilai semula selama proses dekripsi.

Tabel Multiplication 1,2,3,4,9,11,13,14

```
def t1():
```

```
    return [  
0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d  
,0x0e,0x0f,  
  
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d  
,0x1e,0x1f,  
  
0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d  
,0x2e,0x2f,  
  
0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d  
,0x3e,0x3f,  
  
0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d  
,0x4e,0x4f,  
  
0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d  
,0x5e,0x5f,  
  
0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d  
,0x6e,0x6f,  
  
0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d  
,0x7e,0x7f,  
  
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d  
,0x8e,0x8f,  
  
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9a,0x9b,0x9c,0x9d  
,0x9e,0x9f,  
  
0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xaa,0xab,0xac,0xad  
,0xae,0xaf,  
  
0xb0,0xb1,0xb2,0xb3,0xb4,0xb5,0xb6,0xb7,0xb8,0xb9,0xba,0xbb,0xbc,0xbd  
,0xbe,0xbf,  
  
0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd  
,0xce,0xcf,  
  
0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd  
,0xde,0xdf,  
  
0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0xea,0xeb,0xec,0xed  
,0xee,0xef,  
  
0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd  
,0xfe,0xff  
    ]
```

```

def t2():
    return [

0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a
,0x1c,0x1e,

0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a
,0x3c,0x3e,

0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a
,0x5c,0x5e,

0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a
,0x7c,0x7e,

0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a
,0x9c,0x9e,

0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba
,0xbc,0xbe,

0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda
,0xdc,0xde,

0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa
,0xfc,0xfe,

0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01
,0x07,0x05,

0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21
,0x27,0x25,

0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41
,0x47,0x45,

0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61
,0x67,0x65,

0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81
,0x87,0x85,

0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1
,0xa7,0xa5,

0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1
,0xc7,0xc5,

0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1
,0xe7,0xe5

    ]

```

```
def t3():
    return [

0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17
,0x12,0x11,

0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27
,0x22,0x21,

0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77
,0x72,0x71,

0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47
,0x42,0x41,

0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7
,0xd2,0xd1,

0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7
,0xe2,0xe1,

0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7
,0xb2,0xb1,

0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87
,0x82,0x81,

0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c
,0x89,0x8a,

0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc
,0xb9,0xba,

0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec
,0xe9,0xea,

0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc
,0xd9,0xda,

0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c
,0x49,0x4a,

0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c
,0x79,0x7a,

0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c
,0x29,0x2a,

0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c
,0x19,0x1a
```

```
]
```

```
def t9():
```

```
    return [
```

```
0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65  
,0x7e,0x77,
```

```
0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5  
,0xee,0xe7,
```

```
0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e  
,0x45,0x4c,
```

```
0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce  
,0xd5,0xdc,
```

```
0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13  
,0x08,0x01,
```

```
0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83  
,0x98,0x91,
```

```
0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28  
,0x33,0x3a,
```

```
0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8  
,0xa3,0xaa,
```

```
0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89  
,0x92,0x9b,
```

```
0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19  
,0x02,0x0b,
```

```
0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2  
,0xa9,0xa0,
```

```
0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22  
,0x39,0x30,
```

```
0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff  
,0xe4,0xed,
```

```
0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f  
,0x74,0x7d,
```

```
0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4  
,0xdf,0xd6,
```

```
0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54
```

```
,0x4f,0x46
    ]

def t11():
    return [

0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f
,0x62,0x69,

0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf
,0xd2,0xd9,

0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f,0x04
,0x19,0x12,

0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4
,0xa9,0xa2,

0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89
,0x94,0x9f,

0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39
,0x24,0x2f,

0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2
,0xef,0xe4,

0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42
,0x5f,0x54,

0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88
,0x95,0x9e,

0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38
,0x25,0x2e,

0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3
,0xee,0xe5,

0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43
,0x5e,0x55,

0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e
,0x63,0x68,

0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce
,0xd3,0xd8,

0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05
,0x18,0x13,
```

```
0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5,0xa8,0xa3  
    ]
```

```
def t13():  
    return [
```

```
0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51,0x46,0x4b,
```

```
0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81,0x96,0x9b,
```

```
0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea,0xfd,0xf0,
```

```
0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a,0x2d,0x20,
```

```
0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c,0x2b,0x26,
```

```
0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec,0xfb,0xf6,
```

```
0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87,0x90,0x9d,
```

```
0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57,0x40,0x4d,
```

```
0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b,0x9c,0x91,
```

```
0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b,0x4c,0x41,
```

```
0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30,0x27,0x2a,
```

```
0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0,0xf7,0xfa,
```

```
0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6,0xf1,0xfc,
```

```
0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36,0x21,0x2c,
```

```
0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d,0x4a,0x47,
```

```
0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d  
,0x9a,0x97  
]
```

```
def t14():  
    return [
```

```
0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46  
,0x54,0x5a,
```

```
0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6  
,0xb4,0xba,
```

```
0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d  
,0x8f,0x81,
```

```
0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d  
,0x6f,0x61,
```

```
0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb  
,0xf9,0xf7,
```

```
0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b  
,0x19,0x17,
```

```
0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30  
,0x22,0x2c,
```

```
0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0  
,0xc2,0xcc,
```

```
0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07  
,0x15,0x1b,
```

```
0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7  
,0xf5,0xfb,
```

```
0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc  
,0xce,0xc0,
```

```
0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c  
,0x2e,0x20,
```

```
0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa  
,0xb8,0xb6,
```

```
0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a  
,0x58,0x56,
```

```
0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71
```



```
, 0x63, 0x6d,
```

```
0xd7, 0xd9, 0xcb, 0xc5, 0xef, 0xe1, 0xf3, 0xfd, 0xa7, 0xa9, 0xbb, 0xb5, 0x9f, 0x91  
, 0x83, 0x8d  
]
```

Fungsi "t1", "t2", dan "t3" adalah konstanta yang digunakan untuk melakukan enkripsi dan dekripsi dalam algoritma AES. Konstanta "t2" dan "t3" digunakan dalam proses enkripsi "mix columns", di mana "t2" digunakan untuk mengalikan kolom oleh 2 dan "t3" untuk mengalikan kolom oleh 3. Sedangkan konstanta "t9", "t11", "t13", dan "t14" dipakai dalam proses dekripsi "inverse mix columns". "t9" digunakan untuk mengalikan kolom oleh 9, "t11" untuk 11, "t13" untuk 13, dan "t14" untuk 14. Hal ini merupakan proses memanfaatkan operasi XOR, shift, dan pengalihan dalam langkah-langkah enkripsi dan dekripsi data.

Tabel rcon

```
def rcon():  
    return [  
        [0x01, 0x00, 0x00, 0x00],  
        [0x02, 0x00, 0x00, 0x00],  
        [0x04, 0x00, 0x00, 0x00],  
        [0x08, 0x00, 0x00, 0x00],  
        [0x10, 0x00, 0x00, 0x00],  
        [0x20, 0x00, 0x00, 0x00],  
        [0x40, 0x00, 0x00, 0x00],  
        [0x80, 0x00, 0x00, 0x00],  
        [0x1b, 0x00, 0x00, 0x00],  
        [0x36, 0x00, 0x00, 0x00]  
    ]
```

Fungsi "rcon" mengembalikan sebuah matriks 2D yang merepresentasikan nilai-nilai rcon (round constants) yang digunakan dalam proses ekspansi kunci. Rcon digunakan untuk menambahkan kompleksitas dan non-linearitas pada proses ekspansi kunci. Matriks yang dihasilkan memiliki 10 baris dan 4 kolom, dengan setiap elemen mewakili nilai rcon yang akan digunakan pada setiap putaran kunci (round key) dari ronde 1 hingga 10. Setiap nilai rcon disusun dalam format little-endian, dengan byte pertama memiliki nilai yang berubah pada setiap putaran, sementara byte kedua hingga keempat diatur menjadi nol.

Fungsi Padding Data

```
def pad_data(input_data, block_size=16):  
    padded_data = input_data.copy()  
    padding = block_size - (len(input_data) % block_size)  
    if padding != block_size:  
        padded_data.extend([padding] * padding)
```

```
return padded_data
```

Fungsi tersebut adalah fungsi untuk melakukan padding pada data. Padding adalah proses penambahan nilai tambahan (biasanya nol) ke data sehingga panjang data menjadi kelipatan dari suatu ukuran blok tertentu. Fungsi akan mengambil data input dan menambahkan padding ke dalamnya sehingga panjang data setelah padding menjadi kelipatan dari `block_size`. Selanjutnya, fungsi memeriksa apakah padding diperlukan. Jika nilai padding tidak sama dengan ukuran blok (`padding != block_size`), berarti padding perlu ditambahkan

Fungsi Unpadding Data

```
def unpad_data(data):  
    padding = data[-1]  
    if padding > 0 and all(x == padding for x in data[-padding:]):  
        return data[:-padding]  
    else:  
        return data
```

Fungsi ini digunakan untuk menghilangkan padding yang ada pada data yang diberikan. Pertama, fungsi mengambil nilai padding terakhir dari data yang diberikan. Kemudian, dengan menggunakan kondisi, fungsi memeriksa apakah nilai padding tersebut lebih dari 0 dan apakah semua nilai di bagian terakhir data sesuai dengan nilai padding yang diidentifikasi. Jika kedua kondisi terpenuhi, fungsi akan mengembalikan data tanpa bagian padding, yaitu data sebelum bagian terakhir yang merupakan padding. Namun, jika salah satu dari kondisi tersebut tidak terpenuhi, fungsi akan mengembalikan data tanpa mengubahnya. Hal ini bertujuan untuk memastikan bahwa hanya data dengan padding yang valid yang akan dihilangkan padding-nya, sementara data tanpa padding atau dengan padding yang tidak valid akan tetap utuh.

Fungsi Split Data

```
def split_data(input_data, block_size=16):  
    result = [input_data[i:i + block_size] for i in range(0,  
len(input_data), block_size)]  
    return result
```

Fungsi `split_data` digunakan untuk membagi data input menjadi potongan-potongan dengan ukuran yang telah ditentukan, di mana nilai default untuk ukuran blok adalah 16. Fungsi ini menerima dua parameter, yaitu `input_data`, yang merupakan data yang akan dibagi, dan `block_size`, yang menyatakan ukuran blok yang digunakan dalam pembagian data. Dalam implementasinya, fungsi menggunakan list comprehension untuk menghasilkan sebuah list yang berisi potongan-potongan data dengan ukuran sesuai yang ditentukan oleh `block_size`. Proses pembagian data dilakukan dengan menggunakan loop yang memotong data input mulai dari indeks 0 hingga panjang data input dengan selang sepanjang `block_size`. Hasilnya, fungsi mengembalikan list yang berisi potongan-potongan data tersebut sebagai output.

Fungsi Generate Random String

```
def generate_random_string(length):
    import random
    import string
    return ''.join(random.choices(string.ascii_letters +
string.digits, k=length))
```

Fungsi tersebut digunakan untuk menciptakan string acak yang kemudian akan terpanggil jika file yang ingin di enkripsi atau dekripsi tidak memiliki nama header file.

Fungsi File to Hex

```
def file_to_hex(path, header=False):
    header_list = []

    if header == True:
        import os
        file_name = os.path.basename(path)
        header = f'header<<{file_name}>>'
        header_list = [ord(char) for char in header]

    byte_data = ""

    with open(path, 'rb') as file:
        byte_data = file.read()

    return header_list + list(bytearray(byte_data))
```

Fungsi ini bertujuan untuk mengonversi data dari sebuah file ke dalam format heksadesimal. Saat dipanggil, fungsi menerima argumen path yang merupakan path dari file yang ingin dikonversi. Terdapat parameter opsional 'header' dengan default yaitu False. Ketika 'header' diatur sebagai True, maka fungsi akan membuat sebuah header dengan nama file yang diberikan dalam format tertentu dan mengonversinya menjadi nilai ASCII menggunakan 'ord()'. Fungsi kemudian membaca data biner dari file yang diberikan menggunakan 'open()' dengan mode 'rb' (read binary), dan mengubahnya ke dalam bentuk list byte menggunakan 'bytearray()'. Fungsi ini akan mengembalikan gabungan antara header_list (jika header=True) dan list byte dari data file yang telah dikonversi ke dalam heksadesimal.

Fungsi Hex to File

```
def hex_to_file(hex):
    path = ""
    start_list = [104, 101, 97, 100, 101, 114, 60, 60]
    if hex[:len(start_list)] == start_list:
        end_index = None
        for i in range(len(hex) - 1):
            if hex[i] == 62 and hex[i + 1] == 62:
                end_index = i
                break
        if end_index is not None:
```

```

        hex_path = hex[len(start_list):end_index]
        path = ''.join(chr(char) for char in hex_path)
        hex = hex[end_index+2:]
    else:
        path = generate_random_string(10)

    with open(path, 'wb') as file:
        file.write(bytes(hex))
    print(f'File successfully saved with name: {path}')

```

Fungsi `hex_to_file` melakukan konversi data heksadesimal ke file biner dan menyimpannya dengan nama file yang berasal dari sebagian data heksadesimal tersebut. Pertama, fungsi mengecek apakah awalan data heksadesimal sesuai dengan nilai yang ada di `start_list`. Jika cocok, fungsi mencari indeks di mana urutan karakter >> muncul, menandakan akhir dari path file dalam bentuk heksadesimal. Setelah menemukan indeks tersebut, fungsi mengambil sebagian data heksadesimal yang mencakup path file, mengonversinya ke bentuk string, dan menyimpannya di dalam variabel `path`. Jika kondisi awal tidak terpenuhi, fungsi menggunakan fungsi `generate_random_string(10)` untuk menghasilkan nama file acak dengan panjang 10 karakter. Selanjutnya, fungsi membuka file dengan nama yang telah ditentukan (baik itu hasil konversi path heksadesimal atau nama file acak) dengan mode write binary ('wb'). Data heksadesimal yang tersisa setelah path file diambil dan diubah menjadi string, diubah ke dalam bentuk byte dengan menggunakan fungsi `bytes(hex)`, dan ditulis ke dalam file tersebut. Terakhir, fungsi mencetak pesan yang memberitahu bahwa file berhasil disimpan, bersama dengan nama file yang digunakan.

Fungsi State

```

def state(plain):
    state = [
        [plain[0], plain[4], plain[8], plain[12]],
        [plain[1], plain[5], plain[9], plain[13]],
        [plain[2], plain[6], plain[10], plain[14]],
        [plain[3], plain[7], plain[11], plain[15]]
    ]
    return state

```

Fungsi ini menerima inputan berupa array satu dimensi dan fungsi akan mengubahnya menjadi matriks 4x4 secara berurutan.

Fungsi Unstate Matrix

```

def unstate(matrix):
    unstate_list = [matrix[row][col] for col in range(len(matrix[0]))]
    for row in range(len(matrix)):
        return unstate_list

```

Fungsi ini menerima matriks sebagai argumen input dan melakukan proses unstate atau menata ulang matriks tersebut ke dalam bentuk list satu dimensi. Dalam fungsi ini, setiap

elemen dari matriks dimasukkan ke dalam list secara berurutan, diambil per baris, dari kiri ke kanan. Fungsi ini akan mengeluarkan sebuah list satu dimensi yang berisi nilai-nilai dari matriks tersebut dalam urutan yang sesuai dengan penataan baris dan kolomnya.

Fungsi Add Round Key

```
def add_round_key(state, key):  
    added_state = [[state[i][j] ^ key[i][j] for j in range(4)] for i  
in range(4)]  
    return added_state
```

Fungsi ini menerima dua parameter, yaitu state yang merepresentasikan keadaan blok saat ini dalam proses enkripsi, dan key yang merupakan kunci putaran yang akan digunakan. Operasi ini dilakukan dengan melakukan operasi XOR (bitwise exclusive OR) antara setiap elemen blok state dan elemen yang sesuai dalam kunci putaran (key). Hasilnya disimpan dalam sebuah matriks baru yang disebut added_state. Matriks added_state ini kemudian dikembalikan sebagai hasil dari fungsi.

Fungsi Sub Bytes

```
def sub_bytes(state, sbox):  
    subs_state = [[sbox[val >> 4][val & 0xF] for val in row] for row  
in state]  
    return subs_state
```

Fungsi ini merupakan SubBytes dalam algoritma enkripsi Advanced Encryption Standard (AES). Langkah SubBytes dilakukan pada blok data 4x4 dalam proses enkripsi AES. Fungsi ini menerima dua parameter: state, yang merupakan blok data 4x4 yang akan dienkripsi, dan sbox, yang merupakan tabel substitusi (substitution box) yang digunakan untuk menggantikan setiap elemen dalam blok data dengan nilai yang sesuai dari tabel tersebut.

Pada setiap iterasi, fungsi akan melakukan substitusi pada setiap elemen dalam blok data state dengan menggunakan nilai yang terdapat dalam sbox. Elemen blok data diakses satu per satu, dan nilai masing-masing elemen dibagi menjadi dua bagian: empat bit pertama dan empat bit terakhir. Bagian pertama digunakan sebagai indeks baris, sedangkan bagian kedua digunakan sebagai indeks kolom dalam tabel substitusi sbox. Nilai yang diambil dari sbox tersebut kemudian digunakan untuk menggantikan nilai asli elemen tersebut dalam blok data.

Nama fungsi

```
def shift_rows(state, direction):  
    shifted_state = [[0] * 4 for _ in range(4)]  
  
    for row in range(4):  
        # untuk dekripsi  
        if direction == 1:
```

```

        shifted_state[row] = state[row][-row:] + state[row][: -
row]
        # untuk enkripsi
        elif direction == -1:
            shifted_state[row] = state[row][row:] + state[row][:row]

    return shifted_state

```

Fungsi ini bertujuan untuk melakukan pergeseran baris pada matriks 'state' sesuai arah yang ditentukan ('direction'). Pertama, fungsi membuat matriks kosong 'shifted_state' yang memiliki ukuran 4x4. Selanjutnya, dengan menggunakan loop 'for', fungsi melakukan pergeseran baris berdasarkan nilai 'direction'. Jika 'direction' adalah 1 (untuk dekripsi), setiap baris dari 'state' akan digeser ke kanan sebanyak 'row' langkah, sehingga bagian belakang akan dipindahkan ke bagian depan. Sedangkan jika 'direction' adalah -1 (untuk enkripsi), setiap baris akan digeser ke kiri sebanyak 'row' langkah, sehingga bagian depan akan dipindahkan ke bagian belakang. Hasil dari pergeseran ini akan disimpan dalam 'shifted_state' yang kemudian dikembalikan sebagai output dari fungsi ini.

Fungsi Mix Columns

```

def mix_columns(state, t1, t2, t3, t4):
    mixed_state = [[0] * 4 for _ in range(4)]

    # enkripsi 02 03 01 01
    # dekripsi 14 11 13 09
    for c in range(4):
        mixed_state[0][c] = t1[state[0][c]] ^ t2[state[1][c]] ^
t3[state[2][c]] ^ t4[state[3][c]]
        mixed_state[1][c] = t4[state[0][c]] ^ t1[state[1][c]] ^
t2[state[2][c]] ^ t3[state[3][c]]
        mixed_state[2][c] = t3[state[0][c]] ^ t4[state[1][c]] ^
t1[state[2][c]] ^ t2[state[3][c]]
        mixed_state[3][c] = t2[state[0][c]] ^ t3[state[1][c]] ^
t4[state[2][c]] ^ t1[state[3][c]]

    return mixed_state

```

Fungsi ini menerima lima parameter, yaitu "state" yang merepresentasikan keadaan blok saat ini dalam proses enkripsi, dan empat parameter lainnya ("t1", "t2", "t3", "t4") yang digunakan sebagai tabel konstanta untuk operasi MixColumns. Operasi MixColumns melibatkan transformasi matriks "state" dengan matriks konstanta tertentu. Dalam implementasi ini, operasi dilakukan pada setiap kolom matriks "state". Setiap elemen baru dalam matriks hasil ("mixed_state") dihitung dengan menggabungkan operasi XOR dan penggantian byte (substitution) pada tiap baris menggunakan tabel konstanta yang sesuai. Proses ini dilakukan untuk setiap elemen matriks "state" dalam setiap kolom. Hasil akhirnya adalah matriks "mixed_state" yang mewakili keadaan blok setelah operasi MixColumns diterapkan.

Fungsi G

```
def g_function(list, rcon_index, sbox=sbox(), rcon=rcon()):
    shift = list[1:] + [list[0]]
    subs = [sbox[val >> 4][val & 0xF] for val in shift]
    added = [rcon[rcon_index][i] ^ subs[i] for i in range(4)]
    return added
```

Fungsi ini menerima tiga parameter, yaitu `list`, `rcon_index`, `sbox`, dan `rcon` kemudian fungsi ini melakukan operasi penggeseran (`shift`) pada elemen-elemen dalam `list`. Setiap elemen dipindahkan ke posisi berikutnya, dan elemen terakhir dipindahkan ke posisi pertama. Operasi ini menciptakan array yang diwakili oleh variabel `shift`. Selanjutnya, fungsi melakukan substitusi byte (`byte substitution`) pada array `shift` dengan menggunakan kotak substitusi (`s-box`) yang diberikan melalui parameter opsional `sbox`. Substitusi ini dilakukan untuk setiap nilai byte dalam array, dan hasilnya disimpan dalam array yang diwakili oleh variabel `subs`. Setelah itu, fungsi melakukan operasi bitwise XOR (`exclusive OR`) antara setiap elemen dari array substitusi (`subs`) dan elemen pada `rcon` sesuai dengan indeks `rcon_index`. `Rcon` (`round constant`) adalah array konstan yang digunakan dalam proses kunci ekspansi pada setiap putaran kunci. Hasil operasi XOR ini disimpan dalam array yang diwakili oleh variabel `added`.

Fungsi Expand Key

```
def expand_key(key):
    st = state(key)
    added_st = [[0] * 4 for _ in range(4)]

    key_list = [st]

    for i in range(10):
        g_column = g_function([st[r][3] for r in range(4)], i)
        for r in range(4):
            added_st[r][3] = g_column[r]
        for c in range(4):
            for r in range(4):
                added_st[r][c] = st[r][c] ^ added_st[r][c-1]
        st = [row.copy() for row in added_st]
        key_list.append(st)

    return key_list
```

Fungsi ini bertujuan untuk melakukan proses *key expansion* pada kunci yang diberikan. Awalnya, fungsi mengonversi kunci ke dalam bentuk matriks dengan fungsi `'state()'`. Kemudian, dibuat matriks tambahan `'added_st'` yang diisi dengan nilai nol dan sebuah list `'key_list'` yang akan menyimpan setiap iterasi dari kunci yang diekspansi. Melalui iterasi sebanyak 10 kali, fungsi melakukan operasi `g-function` dengan parameter kolom terakhir dari `'st'` untuk menghasilkan nilai baru yang digunakan dalam proses ekspansi kunci. Setelah itu, dilakukan operasi XOR antara `'st'` dan `'added_st'` untuk menghasilkan nilai baru yang akan menjadi bagian dari kunci yang diekspansi. Hasil akhir dari setiap iterasi disimpan dalam `'key_list'`, yang berisi kunci yang diekspansi dalam setiap langkah iterasi, dan akan dikembalikan sebagai output dari

fungsi ini.

Fungsi Enkripsi Block

```
def block_encrypt(plain, list_key):
    st = state(plain)

    rk = add_round_key(st, list_key[0])

    for i in range(1, 10):
        sb = sub_bytes(rk, sbbox())
        sr = shift_rows(sb, -1)
        mc = mix_columns(sr, t2(), t3(), t1(), t1())
        rk = add_round_key(mc, list_key[i])

    sb = sub_bytes(rk, sbbox())
    sr = shift_rows(sb, -1)
    rk = add_round_key(sr, list_key[10])

    cipher = unstate(rk)
    return cipher
```

Fungsi "block_encrypt" ini digunakan untuk melakukan enkripsi blok data menggunakan algoritma Advanced Encryption Standard (AES). Fungsi menerima dua parameter, yaitu "plain" yang merupakan blok data plaintext yang akan dienkrpsi, dan "list_key" yang berisi kunci-kunci putaran yang digunakan selama proses enkripsi. Pertama, fungsi menginisialisasi keadaan awal "st" dengan menggunakan fungsi "state" pada blok plaintext yang diberikan. Selanjutnya, kunci putaran pertama ("rk") dihasilkan dengan menjalankan fungsi "add_round_key" menggunakan keadaan awal "st" dan kunci putaran pertama dari "list_key". Proses iteratif dilakukan sebanyak sembilan putaran enkripsi dengan menggunakan langkah-langkah substitusi byte ("sub_bytes"), pergeseran baris ("shift_rows"), mix columns ("mix_columns"), dan penambahan kunci putaran ("add_round_key"). Setelah sembilan putaran, langkah terakhir dilakukan tanpa operasi mix columns. Setelah kesepuluh putaran, keadaan akhir dihasilkan dari substitusi byte, pergeseran baris, dan penambahan kunci terakhir. Keadaan akhir ini kemudian diubah kembali menjadi bentuk blok data yang dienkrpsi menggunakan fungsi "unstate", dan hasilnya dikembalikan sebagai blok data ciphertext. Dengan demikian, fungsi "block_encrypt" secara keseluruhan merupakan implementasi lengkap dari algoritma enkripsi AES pada satu blok data.

Fungsi Block Decrypt

```
def block_decrypt(cipher, list_key):
    st = state(cipher)
    rk = add_round_key(st, list_key[10])
    sr = shift_rows(rk, 1)
    sb = sub_bytes(sr, inv_sbox())
```



```

for i in range(9, 0, -1):
    rk = add_round_key(sb, list_key[i])
    mc = mix_columns(rk, t14(), t11(), t13(), t9())
    sr = shift_rows(mc, 1)
    sb = sub_bytes(sr, inv_sbox())

rk = add_round_key(sb, list_key[0])

plain = unstate(rk)
return plain

```

Fungsi menerima dua parameter, yaitu cipher yang merupakan blok cipher yang akan didekripsi, dan list_key yang berisi kunci-kunci ronde yang akan digunakan selama proses dekripsi. Pertama, pembuat objek state st dari blok cipher menggunakan fungsi state(cipher). Selanjutnya, kunci ronde terakhir (list_key[10]) ditambahkan ke state menggunakan operasi add_round_key. Kemudian, dilakukan operasi shift pada baris-state (shift_rows) dengan parameter 1, dan dilanjutkan dengan substitusi byte terbalik (sub_bytes) menggunakan invers dari S-box (inv_sbox()). Selanjutnya, dilakukan iterasi dari ronde ke-9 hingga ronde ke-1. Pada setiap iterasi, kunci ronde (list_key[i]) ditambahkan ke state, kemudian dilakukan operasi mix columns (mix_columns) dengan matriks transformasi (t14(), t11(), t13(), t9()). Setelah itu, dilakukan shift rows dan substitusi byte terbalik seperti pada ronde terakhir. Setelah iterasi selesai, kunci ronde pertama (list_key[0]) ditambahkan ke state hasil iterasi terakhir. Terakhir, state tersebut diubah kembali menjadi bentuk blok teks biasa menggunakan fungsi unstate(rk) dan hasilnya dikembalikan sebagai blok teks terdekripsi.

Fungsi AES Encrypt

```

def aes_encrypt(plain, key):
    keys = expand_key(key)
    chunks = split_data(pad_data(plain))

    cipher = []

    for chunk in chunks:
        encrypted_chunk = block_encrypt(chunk, keys)
        cipher.extend(encrypted_chunk)

    return cipher

```

Fungsi ini berfungsi untuk melakukan enkripsi AES pada teks 'plain' menggunakan kunci 'key' yang diberikan. Pertama, fungsi melakukan ekspansi kunci dengan menggunakan 'expand_key(key)' untuk mendapatkan serangkaian kunci yang akan digunakan selama proses enkripsi. Selanjutnya, teks 'plain' dipad dan dibagi menjadi blok-blok data menggunakan 'pad_data()' dan 'split_data()'. Setelah itu, fungsi melakukan enkripsi pada setiap blok data menggunakan fungsi 'block_encrypt()' dengan menggunakan serangkaian kunci yang telah dihasilkan. Blok-blok data yang telah dienkripsi kemudian digabungkan menjadi satu kesatuan 'cipher' yang akan dikembalikan sebagai hasil enkripsi dari teks 'plain' menggunakan AES

dengan kunci yang diberikan.

Fungsi Dekripsi

```
def aes_decrypt(cipher, key):  
    keys = expand_key(key)  
    chunks = split_data(cipher)  
  
    plain = []  
  
    for chunk in chunks:  
        decrypted_chunk = block_decrypt(chunk, keys)  
        plain.extend(decrypted_chunk)  
  
    return unpad_data(plain)
```

Fungsi "aes_decrypt" digunakan untuk mendekripsi data ciphertext yang telah dienkripsi. Fungsi ini menerima dua parameter, yaitu "cipher" yang merupakan data ciphertext yang akan didekripsi, dan "key" yang merupakan kunci rahasia yang digunakan dalam proses dekripsi. Pertama, fungsi meng-expand kunci rahasia ("key") menjadi serangkaian kunci menggunakan fungsi "expand_key". Selanjutnya, data ciphertext ("cipher") dibagi menjadi potongan-potongan dengan menggunakan fungsi "split_data". Proses dekripsi dilakukan pada setiap potongan ciphertext dalam loop. Setiap potongan didekripsi menggunakan fungsi "block_decrypt", yang sebagian besar mengikuti langkah-langkah kebalikan dari langkah-langkah enkripsi AES. Hasil dekripsi dari setiap potongan ciphertext ditambahkan ke dalam daftar "plain". Setelah seluruh potongan ciphertext telah didekripsi, data plaintext hasil dekripsi diubah kembali menjadi bentuk semula dengan menggunakan fungsi "unpad_data", dan hasilnya dikembalikan oleh fungsi sebagai output.

Fungsi Nilai Acak

```
def generate_random(bit):  
    import random  
    return random.getrandbits(bit)
```

Fungsi tersebut digunakan untuk menghasilkan nilai acak berdasarkan jumlah bit yang ditentukan oleh argumen bit. Semakin besar nilai bit, semakin besar pula rentang nilai acak yang mungkin dihasilkan. Fungsi ini digunakan untuk menghasilkan nilai private key acak

Fungsi Check Prima

```
def is_prime(n, k=32):  
    import random  
  
    if n <= 1:  
        return False
```

```

if n <= 3:
    return True
if n % 2 == 0:
    return False
r, d = 0, n - 1
while d % 2 == 0:
    r += 1
    d //= 2
for _ in range(k):
    a = random.randint(2, n - 2)
    x = pow(a, d, n)
    if x == 1 or x == n - 1:
        continue
    for _ in range(r - 1):
        x = pow(x, 2, n)
        if x == n - 1:
            break
    else:
        return False
return True

```

Fungsi ini digunakan untuk melakukan pengecekan apakah bilangan 'n' merupakan bilangan prima atau bukan. Pertama, fungsi mengecek kondisi khusus seperti jika 'n' kurang dari atau sama dengan 1, maka langsung dikembalikan nilai False. Jika 'n' bernilai 2 atau 3, fungsi akan mengembalikan nilai True karena keduanya merupakan bilangan prima. Kemudian, dilakukan pengujian untuk mengecek apakah 'n' habis dibagi 2. Jika 'n' genap (habis dibagi 2), maka langsung dikembalikan nilai False karena bilangan genap selain 2 bukan bilangan prima. Selanjutnya, fungsi melakukan pengujian Miller-Rabin dengan menggunakan 'k' iterasi yang secara acak memilih bilangan 'a' dan melakukan perhitungan pada nilai 'x'. Jika nilai 'x' tidak memenuhi kriteria pada pengujian, maka 'n' dipastikan bukan bilangan prima dan dikembalikan nilai False. Jika selama proses pengujian tidak ditemukan bukti bahwa 'n' bukan prima, maka fungsi mengembalikan nilai True, menandakan bahwa 'n' merupakan bilangan prima berdasarkan pengujian Miller-Rabin.

Fungsi Generate Prima

```

def generate_prime(bit_length):
    import random

    while True:
        candidate = random.getrandbits(bit_length)
        candidate |= (1 << bit_length - 1) | 1
        if is_prime(candidate):
            return candidate

```

Fungsi "generate_prime" bertujuan untuk menghasilkan bilangan prima dengan panjang bit tertentu. Fungsi menggunakan library "random" untuk menghasilkan bilangan bulat acak dengan jumlah bit yang ditentukan oleh parameter "bit_length". Selama perulangan tak terbatas, fungsi membangun kandidat bilangan prima dengan mengatur bit paling signifikan

(MSB) dan bit paling kurang signifikan (LSB) untuk memastikan bahwa bilangan tersebut ganjil dan memiliki panjang bit yang diinginkan. Setelah itu, fungsi memeriksa apakah kandidat tersebut merupakan bilangan prima dengan memanggil fungsi "is_prime". Jika benar, kandidat tersebut dikembalikan sebagai hasil dari fungsi. Fungsi ini menggabungkan pendekatan pembangkitan bilangan prima dengan pengujian primalitas untuk memastikan bahwa bilangan yang dihasilkan adalah prima. Pendekatan pembangkitan menggunakan bit-wise operations untuk mengatur bit-bit tertentu pada kandidat bilangan, sementara pengujian primalitas dilakukan dengan memanggil fungsi "is_prime".

Fungsi Modulus Eksponen

```
def mod_exp(base, exponent, modulus):
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent = exponent // 2
    return result
```

Fungsi menginisialisasi variabel result dengan nilai 1. Kemudian, variabel base diubah menjadi sisa hasil bagi base terhadap modulus, memastikan bahwa nilai base tetap dalam batas modulus. Selanjutnya, dilakukan iterasi menggunakan loop while dengan kondisi bahwa exponent lebih besar dari 0. Dalam setiap iterasi, dilakukan pengecekan apakah exponent ganjil atau genap. Jika exponent ganjil, maka nilai result akan dikalikan dengan base dan diambil sisa hasil bagi terhadap modulus. Selanjutnya, nilai base diperbarui menjadi kuadrat dari nilai sebelumnya, juga diambil sisa hasil bagi terhadap modulus. Terakhir, eksponen dibagi dua setiap iterasi dengan menggunakan operasi floor division (/). Proses ini berlanjut hingga eksponen mencapai nilai 0, dan hasil akhir dari perhitungan eksponensial modulo disimpan dalam variabel result, yang kemudian dikembalikan sebagai output dari fungsi.

Fungsi Int to Hex

```
def int_to_hex(decimal):
    hex_string = hex(decimal)[2:]

    if len(hex_string) > 32:
        hex_string = hex_string[:32]

    while len(hex_string) < 32:
        hex_string = '0' + hex_string

    byte_list = [hex_string[i:i+2] for i in range(0, len(hex_string), 2)]

    hex_list = [int(byte, 16) for byte in byte_list]
```

```
return hex_list
```

Fungsi ini bertujuan untuk mengonversi bilangan desimal menjadi representasi heksadesimal dalam bentuk list bilangan bulat. Pertama, fungsi mengonversi bilangan desimal menjadi string heksadesimal menggunakan fungsi 'hex()', kemudian membuang awalan '0x' dengan mengambil substring dari indeks ke-2 ke depan. Fungsi kemudian memastikan bahwa panjang string heksadesimal tidak melebihi 32 karakter dengan memotongnya jika panjangnya lebih dari 32 karakter. Selanjutnya, dilakukan penambahan nol pada awal string heksadesimal sampai panjangnya mencapai 32 karakter. Setelah itu, string heksadesimal dibagi menjadi byte-byte dua karakter, dan setiap byte dikonversi kembali menjadi bilangan bulat dengan basis 16 menggunakan fungsi 'int()' dan argumen basis 16. Hasil akhir dari konversi ini adalah list yang berisi bilangan bulat yang merepresentasikan nilai heksadesimal dari bilangan desimal yang diberikan.

Fungsi DHKE

```
def DHKE(p, g, x):  
    y = mod_exp(g, x, p)  
    return y
```

Fungsi "DHKE" (Diffie-Hellman Key Exchange) menerapkan algoritma pertukaran kunci Diffie-Hellman. Fungsi ini menerima tiga parameter, yaitu bilangan prima "p", generator "g", dan eksponen pribadi "x". Dalam algoritma Diffie-Hellman, "y" dihitung dengan mengambil nilai "g" dipangkatkan pada eksponen "x" modulo "p". Nilai "y" ini kemudian dapat ditukar dengan pihak lain dalam suatu protokol pertukaran kunci. Pada potongan kode selanjutnya, bilangan prima ("p") dihasilkan menggunakan fungsi "generate_prime" dengan panjang bit 128, sementara generator ("g") dihasilkan dengan panjang bit 16.

9. Jelaskan langkah pengujian hasil Anda? (Gunakan salah satu parameter pengujian misalnya waktu eksekusi, keamanan dsb)

Langkah pengujian dilakukan dengan cara menyiapkan alat yang akan digunakan untuk pengujian. Disini digunakan google colab dan kode python untuk mengecek beberapa parameter dalam menguji aplikasi. Parameter yang diuji yaitu waktu eksekusi, keamanan, dan kompatibilitas.

Waktu Eksekusi

No	Nama File	Ukuran File	Waktu Enkripsi	Waktu Dekripsi
1	1	1 MB	18s	18s
2	5	5 MB	91s	92s
3	10	10 MB	188s	183s

Keamanan

Ketika file telah terenkripsi, maka byte pada isi file sudah teracak dan tidak dapat terlihat isi dari data aslinya. Nama file juga merupakan string acak yang lebih mengamankan konteks file tersebut. Penggunaan key dari pertukaran DHKE juga membuat file aman untuk dipertukarkan di jaringan publik. Key yang memiliki panjang 128 bit akan membutuhkan waktu sangat lama bagi penyerang untuk melakukan brute force.

Kompatibilitas

File berhasil mengenkripsi dan dekripsi semua jenis file maupun ekstensi.

10. Lengkapilah seluruh jawaban Anda berdasar rujukan, dan tulislah rujukannya.

[1] Y. Yusfrizal, A. Meizar, H. Kurniawan, and F. Agustin, "Key management using combination of Diffie–Hellman key exchange with AES encryption," *2018 6th International Conference on Cyber and IT Service Management (CITSM)*, 2018.
doi:10.1109/citsm.2018.8674278

[2] B.Ranganatha Rao, B. Sujatha. "A hybrid elliptic curve cryptography (HECC) technique for fast encryption of data for public cloud security," in *Measurement: Sensors*, vol. 29, pp. 100870, 2023.

[3] Ermatita, Y. Bayu Prastyo, I. W. W. Pradnyana, and M. Adrezo, 'Diffie-Hellman Algorithm for Securing Medical Record Data Encryption keys', in *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, 2020, pp. 296–300.

[4] A. Abusukhon, M. N. Anwar, Z. Mohammad, and B. Alghannam, "A hybrid network security algorithm based on Diffie Hellman and text-to-image encryption algorithm," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 22, no. 1, pp. 65–81, 2019. doi:10.1080/09720529.2019.1569821

[5] R. Patgiri, 'privateDH: An Enhanced Diffie-Hellman Key-Exchange Protocol using RSA and AES Algorithm'. 2021.

[6] Y. Yusfrizal, A. Meizar, H. Kurniawan, and F. Agustin, "Key management using combination of Diffie–Hellman key exchange with AES encryption," *2018 6th International Conference on Cyber and IT Service Management (CITSM)*, 2018.
doi:10.1109/citsm.2018.8674278

[7] R. Patgiri, 'privateDH: An Enhanced Diffie-Hellman Key-Exchange Protocol using RSA and AES Algorithm'. 2021.

[8] Ermatita, Y. Bayu Prastyo, I. W. W. Pradnyana, and M. Adrezo, 'Diffie-Hellman Algorithm for Securing Medical Record Data Encryption keys', in *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, 2020, pp. 296–300.

- [9] S. Ali *et al.*, "An efficient cryptographic technique using modified Diffie–Hellman in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 16, no. 6, p. 155014772092577, 2020. doi:10.1177/1550147720925772
- [10] B. Kumar Pandey *et al.*, 'Encryption and steganography-based text extraction in IoT using the EWCTS optimizer', *Imaging Sci. J.*, vol. 69, no. 1–4, pp. 38–56, May 2021.
- [11] L. Prathibha and K. Fatima, "A novel high-speed data encryption scheme for internet of medical things using modified elliptic curve Diffie–Hellman and advance encryption standard," *International Journal of Image and Graphics*, 2022.
doi:10.1142/s0219467823400041