Laporan Kuis 3 Kriptografi dan Keamanan Siber

IMPLEMENTASI DAN ANALISIS ENKRIPSI *DATA ENCRYPTION STANDARD* (DES) DALAM PYTHON

DOSEN PENGAMPU:

Dr. Muhammad Zaki Almuzakki, S. Si, M. Si, M. Sc

DISUSUN OLEH:

Alghifari Rasyid Zola (105222006)

PROGRAM STUDI ILMU KOMPUTER

FAKULTAS SAINS DAN ILMU KOMPUTER

UNIVERSITAS PERTAMINA

2024/2025

A. Tujuan

Menerapkan algoritma kriptografi *Data Encryption Standard* (DES) dalam bahasa pemrograman Python untuk melakukan enkripsi terhadap plaintext sepanjang 64-bit.

B. Source Code

Github: https://github.com/alghifrz/data-encryption-standard-algorithm

C. Tahapan Enkripsi

1. Konversi Teks ke Bit
    - Plaintext “computer” dan key “12345678” masing-masing dikonversi ke bit array 64-bit.
    - Hasilnya akan menjadi:
      computer => 01100011 01101111 01101101 01110000 01110101 01110100 01100101 01110010
      12345678 => 00110001 00110010 00110011 00110100 00110101 00110110 00110111 00111000
2. Inisial Permutasi
    - Dilakukan permutasi awal terhadap plaintext berdasarkan tabel IP (Initial Permutation).

```
IP = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9, 1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]
```

    - Tujuannya adalah untuk menyebarkan bit plaintext agar memperbesar efek perubahan bit pada proses selanjutnya.
3. Pembentukan 16 Kunci Ronde (Round Key)
    - Key sepanjang 64-bit dipermutasikan dengan tabel PC-1 → menjadi 56-bit.

```
PC1 = [57, 49, 41, 33, 25, 17, 9,
       1, 58, 50, 42, 34, 26, 18,
       10, 2, 59, 51, 43, 35, 27,
       19, 11, 3, 60, 52, 44, 36,
       63, 55, 47, 39, 31, 23, 15,
       7, 62, 54, 46, 38, 30, 22,
       14, 6, 61, 53, 45, 37, 29,
       21, 13, 5, 28, 20, 12, 4]
```

```
def generate_keys(key_bits):
    key_permuted = permute(key_bits, PC1)
    C, D = key_permuted[:28], key_permuted[28:]
    round_keys = []
    for shift in SHIFT:
        C, D = left_shift(C, shift), left_shift(D, shift)
        round_keys.append(permute(C + D, PC2))
    return round_keys
```

- Dibagi menjadi dua bagian: C dan D (masing-masing 28-bit).

```
def generate_keys(key_bits):
    key_permuted = permute(key_bits, PC1)
    C, D = key_permuted[:28], key_permuted[28:]
    round_keys = []
    for shift in SHIFT:
        C, D = left_shift(C, shift), left_shift(D, shift)
        round_keys.append(permute(C + D, PC2))
    return round_keys
```

- Setiap ronde, C dan D di-*left shift* dan dipermutasikan dengan tabel PC-2 → menghasilkan 16 subkey sepanjang 48-bit.

```
PC2 = [14, 17, 11, 24, 1, 5, 3, 28,
       15, 6, 21, 10, 23, 19, 12, 4,
       26, 8, 16, 7, 27, 20, 13, 2,
       41, 52, 31, 37, 47, 55, 30, 40,
       51, 45, 33, 48, 44, 49, 39, 56,
       34, 53, 46, 42, 50, 36, 29, 32]
```

4. Proses 16 Ronde Feistel
    - Di setiap ronde:
        - R diperluas menjadi 48-bit menggunakan tabel ekspansi E.

```
E = [32, 1, 2, 3, 4, 5, 4, 5,
     6, 7, 8, 9, 8, 9, 10, 11,
     12, 13, 12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21, 20, 21,
     22, 23, 24, 25, 24, 25, 26, 27,
     28, 29, 28, 29, 30, 31, 32, 1]
```

```
def feistel(R, K):
    expanded_R = permute(R, E)
    temp = xor(expanded_R, K)
    sbox_out = sbox_substitution(temp)
    return permute(sbox_out, P)
```

        - Dihitung XOR antara hasil ekspansi dengan subkey.
        - Dimasukkan ke S-box untuk substitusi → hasil 32-bit.

```
def sbox_substitution(bits):
    output = []
    for i in range(8):
        block = bits[i * 6:(i + 1) * 6]
        row = (block[0] << 1) | block[5]
        col = (block[1] << 3) | (block[2] << 2) | (block[3] << 1) | block[4]
        val = SBOX[i][row][col]
        output.extend([int(x) for x in format(val, '04b')])
    return output
```

        - Permutasi hasil S-box menggunakan tabel P.

```
P = [16, 7, 20, 21, 29, 12, 28, 17,
     1, 15, 23, 26, 5, 18, 31, 10,
     2, 8, 24, 14, 32, 27, 3, 9,
     19, 13, 30, 6, 22, 11, 4, 25]
```

        - XOR hasil P dengan L → hasil menjadi R selanjutnya, dan R sebelumnya menjadi L selanjutnya.

- Setelah 16 ronde, bagian L dan R terakhir disatukan dalam urutan R || L.

```
def des_encrypt_block(block_bits, key_bits):
    block = permute(block_bits, IP)
    L, R = block[:32], block[32:]
    round_keys = generate_keys(key_bits)

    for i in range(16):
        L, R = R, xor(L, feistel(R, round_keys[i]))
```

5. Final Permutation (FP)
    - Permutasi akhir menggunakan tabel FP untuk menghasilkan ciphertext.

```
FP = [40, 8, 48, 16, 56, 24, 64, 32,
      39, 7, 47, 15, 55, 23, 63, 31,
      38, 6, 46, 14, 54, 22, 62, 30,
      37, 5, 45, 13, 53, 21, 61, 29,
      36, 4, 44, 12, 52, 20, 60, 28,
      35, 3, 43, 11, 51, 19, 59, 27,
      34, 2, 42, 10, 50, 18, 58, 26,
      33, 1, 41, 9, 49, 17, 57, 25]
```

```
    final_block = permute(R + L, FP)
    return final_block
```

D. Hasil Enkripsi

1. Input
    - Plaintext: computer
    - Key: 12345678
2. Output
    - HEX: 3036a53c609d04a3
    - Biner: 00110000 00110110 10100101 00111100 01100000 10011101 00000100 10100011
    - Base64: MDbCpTxgwp0EwqM=

E. Kesimpulan

- Algoritma DES berhasil diterapkan dengan pendekatan modular dan sistematis.
- Proses enkripsi dilakukan dalam 16 ronde dengan fungsi Feistel, yang masing-masing menggunakan subkey unik.
- DES memberikan hasil ciphertext yang *tidak terbaca* dan berbeda jauh dari plaintext.