ENKRIPSI & DEKRIPSI MENGGUNAKAN ALGORITMA DES & RSA

by: Michael Christian P.L
1122024

Daftar Isi

DES (Data Encryption Standard)

- DES adalah algoritma enkripsi simetris berbasis blok 64-bit dengan kunci 56-bit, digunakan untuk mengamankan data melalui 16 tahap enkripsi.

- Kelebihan DES
  - Mudah dipahami
  - Cepat di hardware
  - Telah teruji secara luas

- Kekurangan DES:
  - Kunci terlalu pendek
  - Rentan brute force
  - Tidak cukup aman untuk era modern

RSA

- RSA adalah algoritma kriptografi asimetris yang menggunakan dua kunci: publik untuk enkripsi dan privat untuk dekripsi.

- Kelebihan RSA
  - Aman untuk komunikasi terbuka
  - Cocok untuk digital signature
  - Tidak perlu berbagi kunci rahasia

- Kekurangan RSA:
  - Lebih lambat dari algoritma simetris
  - Tidak cocok untuk data besar
  - Bergantung pada panjang kunci

Arsitektur Program

1

VIEW

- Bertugas untuk menampilkan antarmuka (Tombol, TextBox)
- Menangkap input dari pengguna

2

Model

- Merupakan kan core dari aplikasi ini yang berisi logika dan data
- Implementasi inti algoritma DES: tabel permutasi, S-Box, fungsi pergeseran, dan lainnya.

3

Controller

- Menjembatani View dan Model.
- Menerima event yang terjadi pada view, lalu metodenya akan dilakukan sesuai dengan logika yang ada pada model
- Mengambil hasil dari proses yang dilakukan model yang akan ditampilkan di view

Penjelasan Kode

```
1 reference
public void Encrypt(string plaintextHex, string keyHex)
{
    ClearPreviousResults();

    this.PlaintextHex = plaintextHex;
    this.KeyHex = keyHex;

    // Konversi dari heksadesimal ke biner
    PlaintextBinary = HexToBinary(plaintextHex);
    KeyBinary = HexToBinary(keyHex);

    // Hasilkan sub-kunci dari kunci yang diberikan
    GenerateSubKeys();

    // Proses enkripsi menggunakan sub-kunci urutan normal (K1..K16)
    ProcessRounds(PlaintextBinary, K_list);

    EncryptedHexOutput = BinaryToHex(FinalPermutation);
}
```

```
private static readonly int[] keyPermutationTable =
    {
    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
    };
private static readonly int[] shiftTable = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
private static readonly int[] compressionTable =
    {
    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
    };
```

```
2 references
private void GenerateSubKeys()
{
    PermutedKey = ApplyPermutation(KeyBinary, keyPermutationTable);
    C0 = PermutedKey.Substring(0, 28);
    D0 = PermutedKey.Substring(28, 28);

    C_list.Add(C0);
    D_list.Add(D0);

    string Cn = C0;
    string Dn = D0;

    for (int i = 0; i < 16; i++)
    {
        Cn = LeftShift(Cn, shiftTable[i]);
        Dn = LeftShift(Dn, shiftTable[i]);
        C_list.Add(Cn);
        D_list.Add(Dn);

        string CD = Cn + Dn;
        string K = ApplyPermutation(CD, compressionTable);
        K_list.Add(K);
    }
}
```

Proses:

C12D12 = 0101111 1111000 0110011 0010101 0001111 0101010 1011001 1001111
K12 = 011101 010111 000111 110101 100101 000110 011111 101001

C13D13 = 0111111 1100001 1001100 1010101 0111101 0101010 1100110 0111100
K13 = 100101 111100 010111 010001 111110 101011 101001 000001

C14D14 = 1111111 0000110 0110010 1010101 1110101 0101011 0011001 1110001
K14 = 010111 110100 001110 110111 111100 101110 011100 111010

C15D15 = 1111100 0011001 1001010 1010111 1010101 0101100 1100111 1000111
K15 = 101111 111001 000110 001101 001111 010011 111100 001010

C16D16 = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111
K16 = 110010 110011 110110 001011 000011 100001 011111 110101

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

```
private static readonly int[] initialPermutationTable =
    {
    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
    };
```

```
private static readonly int[] expansionTable =
    {
    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
    };
```

$$K_1 \oplus E(R_0)$$

```
private void ProcessRounds(string inputBinary, List<string> subKeys)
{
    InitialPermutation = ApplyPermutation(inputBinary, initialPermutationTable);
    L0 = InitialPermutation.Substring(0, 32);
    R0 = InitialPermutation.Substring(32, 32);

    L_list.Add(L0);
    R_list.Add(R0);

    for (int i = 0; i < 16; i++)
    {
        string Ln = L_list[i];
        string Rn = R_list[i];

        string expandedR = ApplyPermutation(Rn, expansionTable);
        ExpansionResults.Add(expandedR);

        // Gunakan sub-kunci dari list yang diberikan
        string xorWithKey = XOR(expandedR, subKeys[i]);
        XorWithKeyResults.Add(xorWithKey);

        string sBoxOutput = ApplySBoxes(xorWithKey);
        SBoxOutputs.Add(sBoxOutput);

        string pBoxOutput = ApplyPermutation(sBoxOutput, permutationP);
        PBoxOutputs.Add(pBoxOutput);

        string Rnext = XOR(Ln, pBoxOutput);
        string Lnext = Rn;

        L_list.Add(Lnext);
        R_list.Add(Rnext);
    }

    PreOutput = R_list[16] + L_list[16];
    FinalPermutation = ApplyPermutation(PreOutput, finalPermutationTable);
}
```

```
private static readonly int[,,] SBoxes =
    new int[8, 4, 16]
    {
        {
            { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
            { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
            { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
            { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 }
        },
        {
            { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
            { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
            { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
            { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 }
        },
        {
            { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
            { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
            { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
            { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 }
        },
        {
            { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
            { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
            { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 }
        },
        {
            { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
            { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
            { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
            { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 }
        },
        {
            { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
            { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
            { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
            { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 }
        },
        {
            { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
            { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
            { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
            { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 }
        },
        {
            { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
            { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
            { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
            { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 }
        }
    };
```

```
private static readonly int[] permutationP =
    {
    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
    };
private static readonly int[] finalPermutationTable =
    {
    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
    };
```

Putaran 16:
L15 = 1100 0010 1000 1100 1001 0110 0000 1101
R15 = 0100 0011 0100 0010 0011 0010 0011 0100
E(R15) = 001000 000110 101000 000100 000110 100100 000110 101000
K16 = 110010 110011 110110 001011 000011 100001 011111 110101
K XOR E = 111010 110101 011110 001111 000101 000101 011001 011101
S-Box Output = 1010 0111 1000 0011 0010 0100 0010 1001
P(S-Box Output) = 1100 1000 1100 0000 0100 1111 1001 1000
L16 = 0100 0011 0100 0010 0011 0010 0011 0100
R16 = L15 XOR P(S-Box) = 0000 1010 0100 1100 1101 1001 1001 0101

Gabungan R16 ‖ L16 : 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100
IP-1 : 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101
Ciphertext (Hex) : 85E813540F0AB405

```
1 reference
public void Decrypt(string ciphertextHex, string keyHex)
{
    ClearPreviousResults();

    this.CiphertextHex = ciphertextHex;
    this.KeyHex = keyHex;

    CiphertextBinary = HexToBinary(ciphertextHex);
    KeyBinary = HexToBinary(keyHex);

    GenerateSubKeys();

    // Buat list kunci terbalik untuk dekripsi
    var reversedKeys = new List<string>(K_list);
    reversedKeys.Reverse();

    // Proses dekripsi menggunakan sub-kunci urutan terbalik (K16..K1)
    ProcessRounds(CiphertextBinary, reversedKeys);

    DecryptedHexOutput = BinaryToHex(FinalPermutation);
}
```

Proses:

Putaran 1 sampai 16:
Putaran 1:
L 0 = 0000 1010 0100 1100 1101 1001 1001 0101
R 0 = 0100 0011 0100 0010 0011 0010 0011 0100
E(R 0) = 001000 000110 101000 000100 000110 100100 000110 101000
K16 = 110010 110011 110110 001011 000011 100001 011111 110101
K XOR E = 111010 110101 011110 001111 000101 000101 011001 011101
S-Box Output = 1010 0111 1000 0011 0010 0100 0010 1001
P(S-Box Output) = 1100 1000 1100 0000 0100 1111 1001 1000
L 1 = 0100 0011 0100 0010 0011 0010 0011 0100
R 1 = L 0 XOR P(S-Box) = 1100 0010 1000 1100 1001 0110 0000 1101

Proses:

Putaran 16:
L15 = 1110 1111 0100 1010 0110 0101 0100 0100
R15 = 1111 0000 1010 1010 1111 0000 1010 1010
E(R15) = 011110 100001 010101 010101 011110 100001 010101 010101
K 1 = 000110 110000 001011 101111 111111 000111 000001 110010
K XOR E = 011000 010001 011110 111010 100001 100110 010100 100111
S-Box Output = 0101 1100 1000 0010 1011 0101 1001 0111
P(S-Box Output) = 0010 0011 0100 1010 1010 1001 1011 1011
L16 = 1111 0000 1010 1010 1111 0000 1010 1010
R16 = L15 XOR P(S-Box) = 1100 1100 0000 0000 1100 1100 1111 1111

Gabungan R16 ‖ L16 : 11001100 00000000 11001100 11111111 11110000 10101010 11110000 10101010
IP-1 : 00000001 00100011 01000101 01100111 10001001 10101011 11001101 11101111
Plaintext (Hex) : 0123456789ABCDEF

Penjelasan Kode

```
1 reference
public bool GenerateKeys(string p_str, string q_str, string e_str)
{
    ErrorMessage = "";
    var processLog = new StringBuilder();
    processLog.AppendLine("====== PROSES PEMBANGKITAN KUNCI ======");

    if (!BigInteger.TryParse(p_str, out BigInteger p) || !BigInteger.TryParse(q_str, out BigInteger q) || !BigInteger.TryParse(e_str, out BigInteger e))
    {
        ErrorMessage = "Please enter valid integer values for p, q, and e.";
        return false;
    }

    processLog.AppendLine($"1. Pilih dua bilangan prima, p dan q."); //
    processLog.AppendLine($"    p = {p}");
    processLog.AppendLine($"    q = {q}");


    if (!IsPrime(p) || !IsPrime(q))
    {
        ErrorMessage = "p and q must be prime numbers.";
        processLog.AppendLine("\nERROR: p dan q harus bilangan prima.");
        ProcessOutput = processLog.ToString();
        return false;
    }

    P = p;
    Q = q;
    E = e;

    N = P * Q;
```

```
if (fpb != 1)
{
    ErrorMessage = "e is not relatively prime to phi(n).";
    processLog.AppendLine($"\nERROR: e tidak relatif prima terhadap ϕ(n).");
    ProcessOutput = processLog.ToString();
    return false;
}
```

```
if (!IsPrime(p) || !IsPrime(q))
{
    ErrorMessage = "p and q must be prime numbers.";
    processLog.AppendLine("\nERROR: p dan q harus bilangan prima.");
    ProcessOutput = processLog.ToString();
    return false;
}
```

Input

p 13

q 19

e 31

Key Generation

n 247

phi (n) 216

d 7

Generate

Public Key (31, 247)

Private Key (7, 247)

```
1 reference
private BigInteger Gcd(BigInteger a, BigInteger b, StringBuilder log)
{
    log.AppendLine($"   Mencari FPB({a}, {b}) menggunakan Algoritma Euclidean:"); /
    while (b != 0)
    {
        log.AppendLine($"   {a} mod {b} = {a % b}");
        BigInteger temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

Process

```
====== PROSES PEMBANGKITAN KUNCI ======
1. Pilih dua bilangan prima, p dan q.
   p = 13
   q = 19
2. Hitung n = p * q
   n = 13 * 19 = 247
3. Hitung φ(n) = (p - 1) * (q - 1)
   φ(n) = (13 - 1) * (19 - 1) = 216
4. Pilih kunci publik e = 31
   Syarat: FPB(e, φ(n)) = 1
   Mencari FPB(31, 216) menggunakan Algoritma Euclidean:
   31 mod 216 = 31
   216 mod 31 = 30
   31 mod 30 = 1
   30 mod 1 = 0
   Hasil FPB(31, 216) = 1
   Syarat terpenuhi
```

Encryption

Message (numeric) 60

Ciphertext 109

Encrypt

Process

```
====== PROSES ENKRIPSI ======
Pesan (m) = 60
Kunci Publik (e, n) = (31, 247)
Rumus Enkripsi: c = m^e mod n
c = 60^31 mod 247
c = 109
Hasil Ciphertext: 109
```

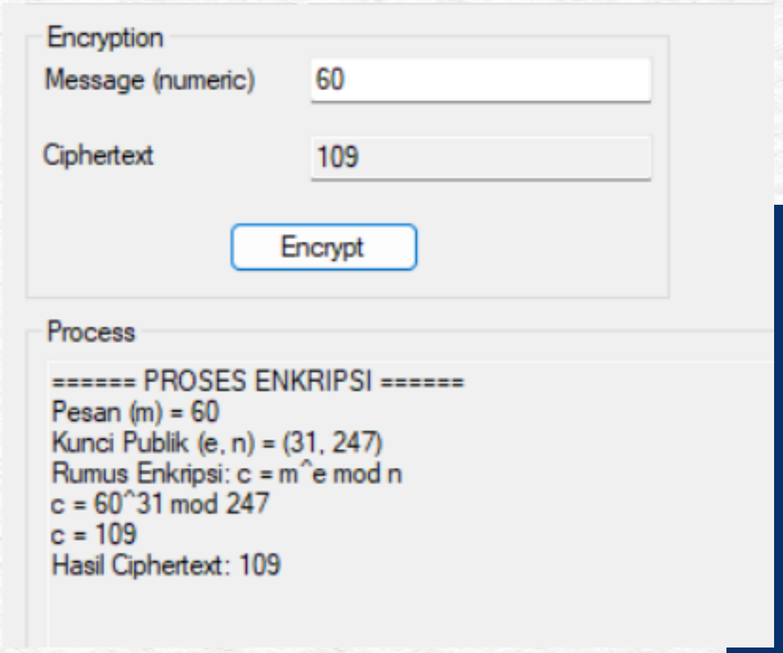
```
public string Encrypt(string message_str)
{
    var processLog = new StringBuilder();
    processLog.AppendLine("\n====== PROSES ENKRIPSI ======");
    if (BigInteger.TryParse(message_str, out BigInteger m))
    {
        processLog.AppendLine($"Pesan (m) = {m}");
        processLog.AppendLine($"Kunci Publik (e, n) = ({E}, {N})");
        processLog.AppendLine($"\nRumus Enkripsi: c = m^e mod n"); //

        BigInteger c = BigInteger.ModPow(m, E, N);

        processLog.AppendLine($"c = {m}^{E} mod {N}");
        processLog.AppendLine($"c = {c}");
        processLog.AppendLine($"\nHasil Ciphertext: {c}");

        ProcessOutput = processLog.ToString();
        return c.ToString();
    }

    ProcessOutput = processLog.AppendLine("Error: Pesan harus berupa angka desimal.").ToString();
    return "Invalid Message";
}
```



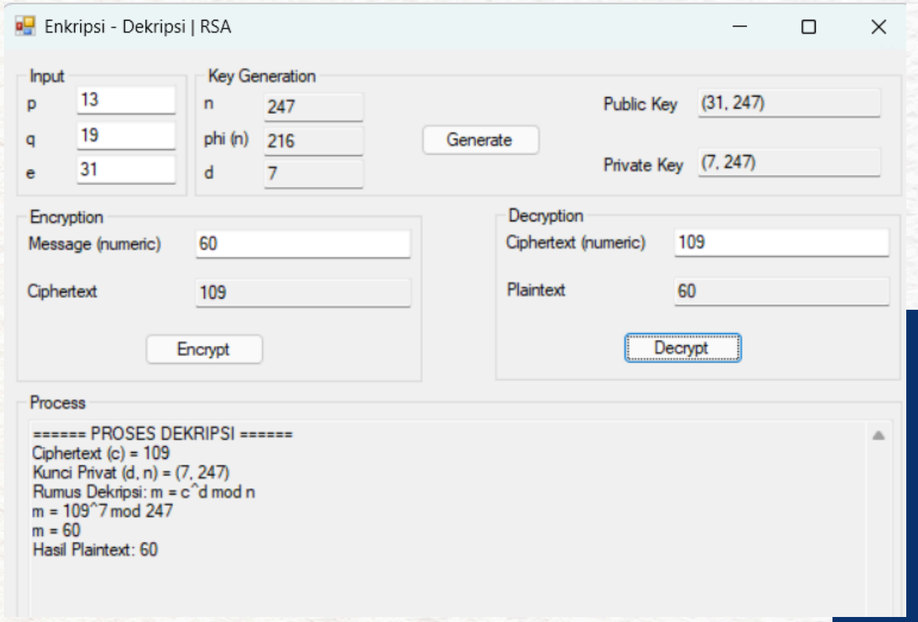
```
1 reference
public string Decrypt(string ciphertext_str)
{
    var processLog = new StringBuilder();
    processLog.AppendLine("\n====== PROSES DEKRIPSI ======");
    if (BigInteger.TryParse(ciphertext_str, out BigInteger c))
    {
        processLog.AppendLine($"Ciphertext (c) = {c}");
        processLog.AppendLine($"Kunci Privat (d, n) = ({D}, {N})");
        processLog.AppendLine($"\nRumus Dekripsi: m = c^d mod n"); //

        BigInteger m = BigInteger.ModPow(c, D, N);

        processLog.AppendLine($"m = {c}^{D} mod {N}");
        processLog.AppendLine($"m = {m}");
        processLog.AppendLine($"\nHasil Plaintext: {m}");

        ProcessOutput = processLog.ToString();
        return m.ToString();
    }

    ProcessOutput = processLog.AppendLine("Error: Ciphertext harus berupa angka desimal.").ToString();
    return "Invalid Ciphertext";
}
```



DEMO APLIKASI

SEKIAN TERIMA KASIH