Enkripsi KTM x *E-Money* Menggunakan Algoritma Rijndael

Syarifuddin Fakhri Al Husaini 13518095
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13518095@std.stei.itb.ac.id

Abstrak—Perkembangan teknologi semakin pesat, salah satunya adalah penggunaan KTM dan *E-Money* yang menggunakan sebuah cip. Data yang disimpan dalam cip perlu dijaga keamanannya agar tidak disalah gunakan. Pada makalah ini akan dibahas sebuah kombinasi antara KTM dan *E-Money* dengan menggunakan algoritma keamanan Rijndael.

Keyword—kriptografi; enkripsi; deskripsi; KTM; *E-Money*; Rijndael

I. PENDAHULUAN

Saat ini teknologi tidak dapat dipisahkan dengan kehidupan manusia. Dari bangun tidur hingga tidur kembali manusia akan sulit agar tidak menggunakan sebuah teknologi. Dengan teknologi seseorang dapat mencari suatu hal dengan mudah, dapat memprediksi naik turunnya saham, hingga digunakan dalam hal pembayaran. Salah satu metode pembayaran yang menggunakan teknologi adalah *E-Money*. Dengan *E-Money* seseorang dapat membayar makanan, minuman, biaya parkir, tiket bis, tiket kereta, dan lain sebagainya dengan hanya sekali tempel.

Karena pada umumnya *E-Money* berbentuk dalam sebuah kartu, sistem *E-Money* ini juga dapat dikombinasikan dengan hal yang lain seperti KTM atau Kartu Tanda Mahasiswa. Dengan kombinasi antara KTM dan *E-Money*, hanya dengan satu kartu saja mahasiswa dapat menggunakannya sebagai presensi atau keperluan kemahasiswaan yang lain dan digunakan sebagai alat pembayaran.

Namun, sistem kartu KTM x *E-Money* ini juga memunculkan beberapa persoalan, apalagi komponen yang digunakan merupakan sesuatu yang sensitif dan berharga yaitu uang dan data mahasiswa. Data yang disimpan dalam cip kartu harus dipastikan keamanannya agar transaksi berjalan dengan tepat. Salah satu cara yang digunakan adalah menggunakan kriptografi agar data dalam kartu dapat dipastikan keamanannya.

Oleh karena itu, pada makalah ini akan dibahas mengenai implementasi enkripsi data kartu KTM x *E-Money* menggunakan algoritma Rijndael. Dengan menggunakan algoritma Rijndael diharapkan data terjaga kerahasiaanya dan adanya data integritas di dalamnya sesuai tujuan dari kriptografi itu sendiri.

II. DASAR TEORI

A. Kriptografi

Kriptografi berasal dari bahasa Yunani yaitu *cryptos* dan *graphein*. *Cryptos* berarti rahasia dan *graphein* berarti tulisan, jika digabung akan memiliki makna tulisan rahasia. Definisi kriptografi adalah sebuah seni dan ilmu yang digunakan untuk menjaga keamanan suatu pesan [3]. Definisi kriptografi lainnya adalah ilmu yang mempelajari teknik-teknik matematika yang berkaitan dengan keamanan informasi seperti kerahasiaan, integritas data, otentikasi entitas, dan otentikasi keaslian.

Menurut Schneier [3], terdapat 4 dasar tujuan dari penggunaan kriptografi. Tujuan-tujuan tersebut sebagai berikut:

1) Terjaga Kerahasiaannya
   Artinya sebuah layanan digunakan untuk menyimpan informasi kecuali orang-orang yang berwenang.
2) Data integritas
   Artinya sebuah layanan digunakan untuk menangani perubahan-perubahan yang terjadi pada suatu data yang tidak sah.
3) Otentikasi
   Artinya sebuah layanan yang digunakan dapat diyakini bahwa pengirim pesan asli, bukan pihak ketiga yang menyamar.
4) *Non-Repudiation*
   Artinya sebuah layanan yang digunakan tidak dapat disangkal oleh pengirimnya setelah pesan itu dikirim

B. Algoritma Rijndael

Algoritma Rijndael merupakan sebuah algoritma keamanan yang diusulkan oleh Vincent Rijmen dan Joan Daemen dari Belgia pada perlombaan yang diadakan oleh *National Institute of Standards and Technology* (NIST). Perlombaan ini diselenggarakan untuk membuat standard algoritma kriptografi yang baru untuk menggantikan *Data Encryption Standard* (DES) yang tidak aman karena kunci dapat ditemukan secara *brute-force*. Standard tersebut kelak diberi nama *Advanced Encryption*

Standard (AES). Algoritma Rijndael dipilih menjadi pemenang pada perlombaan tersebut pada Otober 2000 dan ditetapkan sebagai algoritma AES pada bulan November 2001.

Berikut merupakan spesifikasi dari algoritma Rijndael:

1) Rijndael mendukung panjang kunci 128 bit sampai 256 bit dengan step 32 bit (yaitu 128 bit, 160 bit, 192 bit, …, 256 bit). Pada umumnya, panjang kunci yang digunakan adalah 128, 192, dan 256 bit. Oleh sebab itu, algoritma ini juga dikenal dengan AES-128, AES-192, dan AES-256.
2) Secara *de-fakto*, hanya ada dua varian AES, yaitu AES-128 dan AES-256, karena akan sangat jarang pengguna menggunakan kunci yang panjangnya 192 bit.
3) Panjang kunci dan ukuran blok dapat dipilih secara independen.
4) Setiap blok dienkripsi dalam sejumlah putaran tertentu, sebagaimana halnya pada DES.

Algoritma Rijndael merupakan salah satu algoritma yang cukup aman untuk saat ini. Dengan panjang kunci 128 bit, maka akan terdapat $3{,}4 \times 10^{38}$ kemungkinan kunci. Jika komputer tercepat yang ada saat ini dapat mencoba 1 juta kunci setiap milidetik, maka dibutuhkan waktu $5{,}4 \times 10^{18}$ tahun untuk mencoba seluruh kunci.

Algoritma Rijndael beroperasi dalam orientasi *byte*, bukan seperti DES yang berorientasi bit. Berikut akan dijelaskan garis besar algoritma enkripsi Rijndael yang beroperasi pada blok 128 bit dengan panjang kunci 128 bit:

1) AddRoundKey
   Yaitu melakukan XOR antara *state* awal (plainteks) dengan *cipher key*. Tahap ini disebut juga *initial round*.
2) Melakukan putaran sebanyak Nr – 1 kali
   Proses yang dilakukan pada setiap putaran adalah sebagai berikut:
   a) *SubBytes*
      Melakukan substitusi *byte* dengan menggunakan tabel substitusi (S-box).
   b) *ShiftRows*
      Melakukan pergeseran baris-baris *array state* secara *wrapping*.
   c) *MixColumns*
      Mengacak data di masing-masing kolom *array state*.
   d) *AddRoundKey*
      Melakukan XOR antara *state* sekarang *round key*. *Round key* akan dibangkitkan tiap putarannya sehingga akan berbeda pula tiap putarannya.
3) *Final round*
   Berikut proses untuk putaran terakhir:
   a) *SubBytes*
   b) *ShiftRows*
   c) *AddRoundKey*

Algoritma deskripsi Rijndael merupakan kebalikan dari algoritma enkripsinya. Urutan operasi akan dilakukan terbalik dari nomor 3 hingga 1. Proses yang dilakukan juga kebalikannya yaitu *InvSubBytes*, *InvShiftRows*, *InvMixColumns*, dan *InvAddRoundKey*.

III. RANCANGAN SOLUSI DAN IMPLEMENTASI

Dalam menyelesaikan permasalahan, terdapat langkah-langkah penyelesaian yaitu Deskripsi Umum, Rancangan, dan Implementasi Solusi.

A. Deskripsi Umum Solusi

Dalam perancangan kartu KTM x *E-Money*, data yang disimpan dalam cip kartu perlu dienkripsi terlebih dahulu. Saat kartu digunakan, mesin pembaca kartu akan mendeskripsi data yang ada dalam cip kartu. Algoritma keamanan yang digunakan dalam kartu adalah algoritma Rijndael. Dengan penggunaan algoritma Rijndael diharapkan dapat dipastikan keamanannya dan terjaga kerahasiaannya sesuai tujuan kriptografi itu sendiri.

Algoritma Rijndael yang akan diimplementasikan dalam perancangan KTM x *E-Money* ini adalah AES-128. Pertimbangannya adalah dengan kunci sepanjang 128 bit saja, algoritma Rijndael sulit untuk memecahkan pesan yang ada di dalamnya.

Solusi yang diimplementasikan memiliki dua sudut pandang yaitu pembuat kartu sebagai pemilik data dan mesin pembaca kartu sebagai pembaca data.

B. Rancangan Solusi

Solusi dirancang terdiri dari dua sudut pandang yaitu pembuatan kartu (enkripsi data) dan mesin pembaca kartu (deskripsi data). Arsitektur rancangan dapat dilihat pada Gambar 3.1.

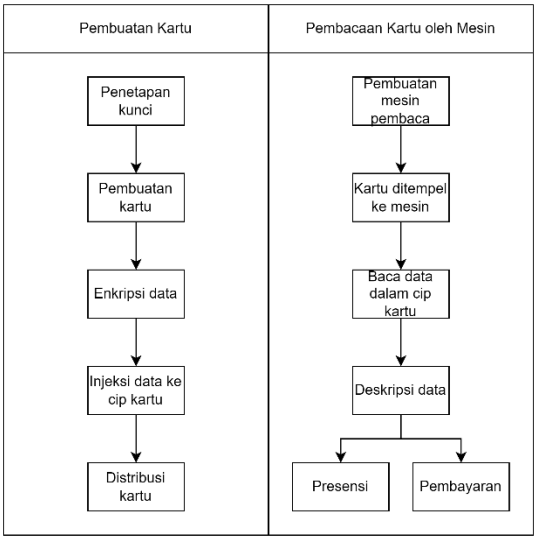
Tahapan proses pada susut pandang pembuat kartu (dalam konteks ini adalah kampus) adalah sebagai berikut:

1) Kampus menetapkan kunci untuk mengamankan kartu.
2) Membuat cip dan kartu.
3) Mengenkripsi data yang akan disimpan ke dalam kartu menggunakan algoritma Rijndael.
4) Memasukkan data ke dalam kartu.
5) Mendistribusikan kartu ke mahasiswa.

Tahapan proses pada susut pandang mesin pembaca kartu adalah sebagai berikut:

1) Pembuatan mesin pembaca kartu.
2) Saat kartu ditempel pada mesin pembaca, data pada cip kartu akan dibaca oleh mesin.

3) Mesin mendeskripsi data yang telah dibaca menggunakan algoritma Rijndael dengan kunci yang telah ditetapkan sebelumnya.
4) Melanjutkan proses selanjutnya (untuk presensi atau pembayaran).



Gambar 3.1 Arsitektur Rancangan Solusi Kartu KTM x *E-Money*

C. Implementasi Solusi

Solusi yang diimplementasikan memiliki beberapa buah batasan implementasi agar penelitian lebih terfokus dengan topik permasalahan yang ada. Beberapa batasan implementasi yang dilakukan adalah sebagai berikut:

1) Kartu beserta cipnya dianggap sudah ada
2) Tidak ada proses memasukkan data ke dalam kartu.
3) Mesin pembaca kartu dianggap sudah ada.
4) Tidak ada proses pembacaan data.
5) Implementasi berfokus pada enkripsi dan deskripsi data menggunakan algoritma Rijndael.
6) Kunci yang digunakan maksimal sepanjang 16 karakter.
7) Implementasi algoritma Rijndael menggunakan bahasa pemrograman Python dalam bentuk GUI.

Berikut implementasi algoritma Rijndael yang dibuat:

1) Fungsi enkripsi
   Fungsi ini digunakan untuk mengenkripsi plaintext dengan sebuah *cipher key* dan akan menghasilkan ciphertext.

```
def encrypt(message, key):
    input_bytes = bytes(message, 'utf-
8')
    state = [[] for j in range(4)]
    for r in range(4):
        for c in range(nb):
            state[r].append(
input_bytes[r + 4 * c])

    key_schedule = key_expansion(key)

    state = add_round_key(state,
key_schedule)

    for rnd in range(1, nr):
        state = sub_bytes(state)
        state = shift_rows(state)
        state = mix_columns(state)
        state = add_round_key(state,
key_schedule, rnd)

    state = sub_bytes(state)
    state = shift_rows(state)
    state = add_round_key(state,
key_schedule, rnd + 1)

    output = [None for i in range(4 *
nb)]
    for r in range(4):
        for c in range(nb):
            output[r + 4 * c] =
state[r][c]

    res = [chr(val) for val in output]
    return ''.join(res)
```

2) Fungsi deskripsi
   Fungsi ini digunakan untuk mendeskripsi ciphertext dengan sebuah *cipher key* dan akan menghasilkan plaintext.

```
def decrypt(ciphertext, key):
    cipher = [ord(val) for val in
ciphertext]

    state = [[] for i in range(nb)]
    for r in range(4):
        for c in range(nb):
            state[r].append(cipher[r + 4
* c])

    key_schedule = key_expansion(key)

    state = add_round_key(state,
key_schedule, nr)

    rnd = nr - 1
    while rnd >= 1:
        state = shift_rows(state,
inv=True)
        state = sub_bytes(state,
inv=True)
        state = add_round_key(state,
key_schedule, rnd)
        state = mix_columns(state,
inv=True)

        rnd -= 1

    state = shift_rows(state, inv=True)
    state = sub_bytes(state, inv=True)
    state = add_round_key(state,
key_schedule, rnd)

    output = [None for i in range(4 *
nb)]
    for r in range(4):
        for c in range(nb):
            output[r + 4 * c] =
state[r][c]

    res = [chr(val) for val in output]
```

```
    return ''.join(res)
```

3) Fungsi sub_byte
   Fungsi ini digunakan untuk melakukan substitusi *byte* dengan menggunakan tabel substitusi (S-box).

```
def sub_bytes(state, inv=False):
    if inv == False:  # encrypt
        box = sbox
    else:  # decrypt
        box = inv_sbox

    for i in range(len(state)):
        for j in range(len(state[i])):
            row = state[i][j] // 0x10
            col = state[i][j] % 0x10

            box_elem = box[16 * row +
col]
            state[i][j] = box_elem

    return state
```

4) Fungsi shift_rows
   Fungsi ini digunakan untuk melakukan pergeseran baris-baris *array state* secara *wrapping*.

```
def shift_rows(state, inv=False):
    count = 1

    if inv == False:  # encrypting
        for i in range(1, nb):
            state[i] =
left_shift(state[i], count)
            count += 1
    else:  # decryptionting
        for i in range(1, nb):
            state[i] =
right_shift(state[i], count)
            count += 1

    return state
```

5) Fungsi mix_columns
   Fungsi ini digunakan untuk mengacak data di masing-masing kolom *array state*.

```
def mix_columns(state, inv=False):
    for i in range(nb):

        if inv == False:  # encryption
            s0 = mul_by_02(state[0][i])
^ mul_by_03(state[1][i]) ^ state[2][i] ^
state[3][i]
            s1 = state[0][i] ^
mul_by_02(state[1][i]) ^
mul_by_03(state[2][i]) ^ state[3][i]
            s2 = state[0][i] ^
state[1][i] ^ mul_by_02(state[2][i]) ^
mul_by_03(state[3][i])
            s3 = mul_by_03(state[0][i])
^ state[1][i] ^ state[2][i] ^
mul_by_02(state[3][i])
        else:  # decryption
            s0 = mul_by_0e(state[0][i])
^ mul_by_0b(state[1][i]) ^
mul_by_0d(state[2][i]) ^
mul_by_09(state[3][i])
            s1 = mul_by_09(state[0][i])
^ mul_by_0e(state[1][i]) ^
mul_by_0b(state[2][i]) ^
mul_by_0d(state[3][i])
            s2 = mul_by_0d(state[0][i])
^ mul_by_09(state[1][i]) ^
mul_by_0e(state[2][i]) ^
mul_by_0b(state[3][i])
            s3 = mul_by_0b(state[0][i])
^ mul_by_0d(state[1][i]) ^
mul_by_09(state[2][i]) ^
mul_by_0e(state[3][i])

        state[0][i] = s0
        state[1][i] = s1
        state[2][i] = s2
        state[3][i] = s3

    return state
```

6) Fungsi add_row_key
   Fungsi ini digunakan untuk melakukan XOR antara *state* sekarang *round key*.

```
def add_round_key(state, key_schedule,
round=0):
    for col in range(nk):
        s0 = state[0][col] ^
key_schedule[0][nb * round + col]
        s1 = state[1][col] ^
key_schedule[1][nb * round + col]
        s2 = state[2][col] ^
key_schedule[2][nb * round + col]
        s3 = state[3][col] ^
key_schedule[3][nb * round + col]

        state[0][col] = s0
        state[1][col] = s1
        state[2][col] = s2
        state[3][col] = s3

    return state
```

7) Fungsi key_expansion
   Fungsi ini digunakan untuk membangkitan *round key* selanjutnya.

```
def key_expansion(key):
    key_symbols = [ord(symbol) for
symbol in key]

    if len(key_symbols) < 4 * nk:
        for i in range(4 * nk -
len(key_symbols)):
            key_symbols.append(0x01)

    key_schedule = [[] for i in
range(4)]
    for r in range(4):
        for c in range(nk):

key_schedule[r].append(key_symbols[r + 4
* c])

    for col in range(nk, nb * (nr + 1)):
        if col % nk == 0:
            tmp = [key_schedule[row][col
- 1] for row in range(1, 4)]

tmp.append(key_schedule[0][col - 1])
```

```
            for j in range(len(tmp)):
                sbox_row = tmp[j] //
0x10
                sbox_col = tmp[j] % 0x10
                sbox_elem = sbox[16 *
sbox_row + sbox_col]
                tmp[j] = sbox_elem

            for row in range(4):
                s =
(key_schedule[row][col - 4]) ^
(tmp[row]) ^ (rcon[row][int(col / nk -
1)])

key_schedule[row].append(s)

        else:
            for row in range(4):
                s =
key_schedule[row][col - 4] ^
key_schedule[row][col - 1]

key_schedule[row].append(s)

    return key_schedule
```

IV. PENGUJIAN DAN PEMBAHASAN

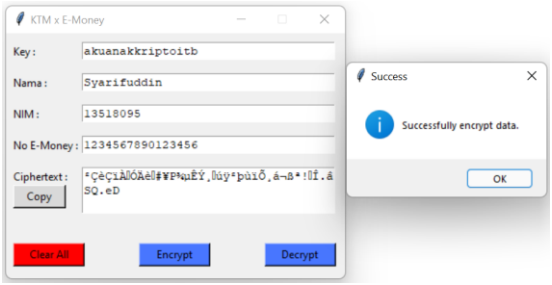
A. Pengujian

Berikut spesifikasi laptop yang digunakan untuk pengujian:

1) CPU Ryzen 5600U dengan AMD Radeon (TM) Graphic
2) RAM 16GB
3) Sistem operasi Windows 11

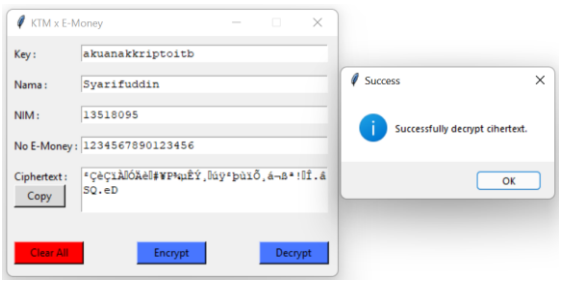
Terdapat empat pengujian yang akan dilakukan yaitu enkripsi valid, deskripsi valid, enkripsi tidak valid, dan deskripsi tidak valid.

1) Enkripsi valid



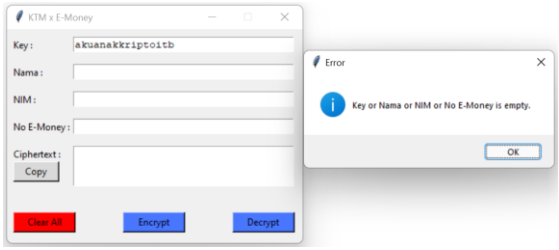
Gambar 4.1 Hasil Enkripsi Valid

2) Deskripsi valid

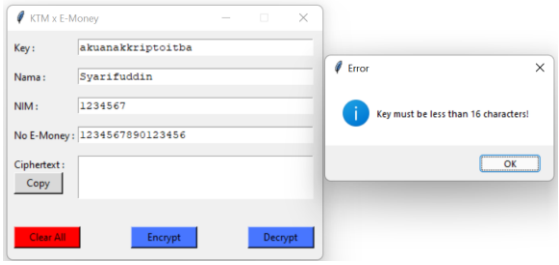


Gambar 4.2 Hasil Deskripsi Valid

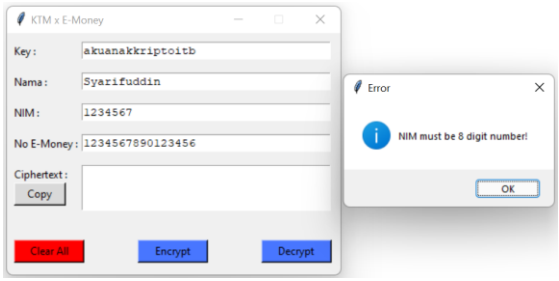
3) Enkripsi tidak valid



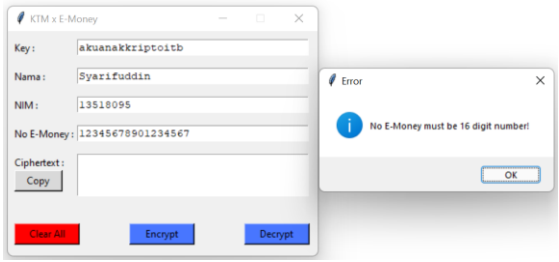
Gambar 4.3 Hasil Enkripsi Tidak Valid (1)



Gambar 4.4 Hasil Enkripsi Tidak Valid (2)

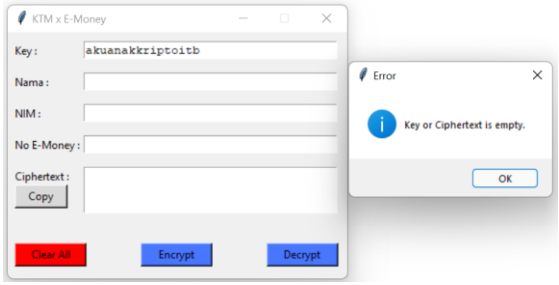


Gambar 4.5 Hasil Enkripsi Tidak Valid (3)

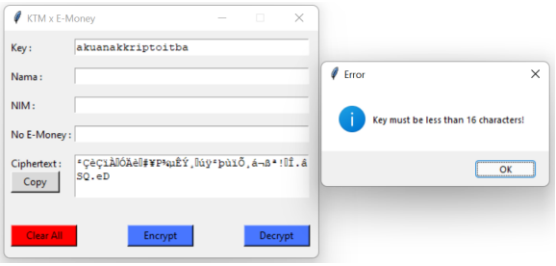


Gambar 4.6 Hasil Enkripsi Tidak Valid (4)

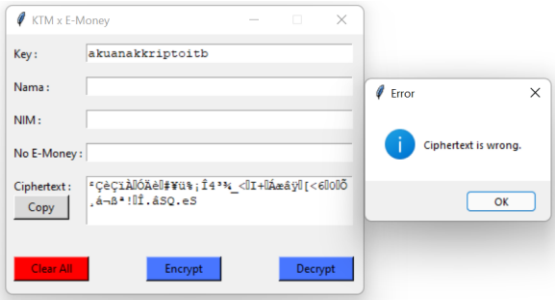
4) Deskripsi tidak valid



Gambar 4.7 Hasil Deskripsi Tidak Valid (1)



Gambar 4.8 Hasil Deskripsi Tidak Valid (2)



Gambar 4.9 Hasil Deskripsi Tidak Valid (3)

B. Pembahasan

Berdasarkan pengujian yang telah dilakukan, didapatkan hasil sebagai berikut:

1) Enkripsi valid
   Pengujian menunjukkan respons sukses dan data telah terenkripsi
2) Deskripsi valid
   Pengujian menunjukkan respons sukses dan data telah terdeskripsi sesuai data yang dimasukkan sebelumnya
3) Enkripsi tidak valid
   Terdapat beberapa kondisi yang dapat meng-akibatkan enkripsi tidak valid yaitu:
   a) Terdapat *field* yang tidak diisi
   b) *Key* yang dimasukkan terlalu panjang
   c) Format NIM tidak sesuai
   d) Format Nomor *E-Money* tidak sesuai
4) Deskripsi tidak valid
   Terdapat beberapa kondisi yang dapat meng-akibatkan deskripsi tidak valid yaitu:
   a) Terdapat *field* yang tidak diisi
   b) *Key* yang dimasukkan terlalu panjang
   c) Input ciphertext salah

V. KESIMPULAN DAN SARAN

Solusi yang diimplementasikan berhasil untuk mengenkripsi dan mendeskripsi data pada kartu KTM x *E-Money*. Hal ini dapat memastikan kerahasiaan data dan adanya data integritas pada kartu tersebut.

Kedepannya, solusi yang dapat dikembangkan adalah penggunaan algoritma keamanan Rijndael pada kartu KTM x E-Money dengan dikombinasikan dengan tanda tangan digital. Hal ini dapat menambah keamanan dari kartu KTM x E-Money dan memenuhi aspek otentikasi.

VI. UCAPAN TERIMA KASIH

Ucapan terimakasih penulis nyatakan kepada Tuhan Yang Maha Esa, karena karunia-Nya penulis bisa diberikan kesempatan untuk menyelesaikan dan bisa memberikan kontribusi nyata dalam memberikan ide yang dituliskan pada makalah ini.

Penulis juga mengucapkan terimakasih kepada Dr. Rinaldi Munir atas dedikasinya dalam memberikan ilmu pengetahuan tentang kriptografi kepada penulis

REFERENSI

[1] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Pengantar Kriptografi
[2] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Review Beberapa Block Cipher dan Stream Cipher (Bagian 4: Advanced Encryption Standard (AES))
[3] Schneier, B. 1997. Applied Cryptography. CRC Press, Inc.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2021

Syarifuddin Fakhri Al Husaini