# Penerapan Aritmatika Modular dan *Hashing* dalam Validasi *Basic Access Control* pada Keamanan *e-Passport*

Azfa Radhiyya Hakim - 13523115[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
[1]*aradhihakim@gmail.com*, *13523115@std.stei.itb.ac.id*

***Abstract*—Aritmatika Modular adalah salah satu topik dari teori bilangan yang di implementasikan pada banyak hal di dunia keinformatikaan. Bilangan prima, yang banyak digunakan dalam Arimatika Modular, merupakan salah satu konsep dasar dalam ilmu kriptografi. Kriptografi dapat digunakan untuk menjaga keamanan dalam menggunakan teknologi, salah satunya adalah *Basic Access Control* pada Keamanan *e-Passport*. Proses ini dilakukan dengan mengubah 24-*byte* informasi pada *e-Passport* menjadi dua kunci utama enkripsi, yang kemudian akan memanfaatkan pembangkit bilangan acak untuk memverifikasi kevalidan.**

***Keywords*—Algoritma Modular, *Basic Access Control*, Kriptografi, Pembangkit Bilangan Acak**

## I. PENDAHULUAN

Matematika Diskrit, merupakan ilmu yang menjadi konsep dasar dan solusi dari berbagai algoritma informatika dan permasalahan teknologi. Teori bilangan, adalah salah satu konsep dalam Matematika Diskrit yang menjadi fondasi dari berbagai teknologi yang ada di sekitar kita. Di dalam konsep ini, dipelajari bilangan prima dan operasi modulo. Kedua hal tersebut membangun suatu konsep baru, yaitu Arimatika Modular. Konsep ini dapat dimanfaatkan dalam mengenkripsi pesan.

Di dunia yang telah dipenuhi oleh teknologi, akan timbul banyak keuntungan yang dapat mempermudah manusia dalam menjalani kehidupan. Namun di sisi lain, akan timbul pula kerugian yang disebabkan oleh orang-orang tidak bertanggung jawab. Imigrasi adalah suatu proses perpindahan seseorang (atau sekelompok) dari suatu negara ke negara lain. Seseorang harus memiliki paspor untuk melakukan imigrasi. Sebelum seseorang dapat berpindah, akan di cek terlebih dahulu kevalidan dari paspor tersebut, untuk menghindari pemalsuan identitas. Setiap paspor akan memiliki suatu *string* unik, yaitu MRZ (*Machine Readable Zone*) yang berisi beberapa data pribadi.

Dalam makalah ini, akan dibahas penerapan algoritma modular, bilangan prima, pembangkit bilangan acak, dan kriptografi dalam mengecek validitas kode unik yang ada pada tiap paspor.

## II. LANDASAN TEORI

### A. Bilangan Prima

Bilangan prima, merupakan suatu bilangan bulat, lebih besar dari 1, yang hanya memiliki 2 faktor, yaitu 1 dan dirinya sendiri. Bilangan ini menjadi konsep dasar dan fundamental dalam teori bilangan, terutama dalam bidang-bidang yang menghubungkan sifat dari dua atau lebih bilangan. Beberapa contoh bilangan prima adalah $2,3,5,7,11,\ldots$ Secara matematika, suatu bilangan dapat dikatakan bilangan prima, jika

$$\forall d \in \mathbb{Z}^+, d|p \Longrightarrow (d = 1 \; or \; d = p)$$

### B. Algoritma Modular

Algoritma Modular, merupakan suatu konsep dalam matematika, yang banyak digunakan dalam menyelesaikan permasalahan yang berkaitan dengan bilangan besar. Diperkenalkan istilah **mod** ($\equiv$), yang melambangkan sisa pembagian dari suaatau operasi. Misalkan $a$ adalah bilangan bulat dan $m$ adalah bilangan bulat $> 0$. Operasi $a$ **mod** $m$ (dibaca "$a$ modulo $m$") memberikan sisa apabila $a$ dibagi dengan $m$ [1]. Bilangan $m$ disebut **modulo** dan hasil operasi modulo $m$ terletak pada himpunan $\{0,1,2,\ldots,m-1\}$. Dengan kata lain, akan terdapat suatu $r \in Z^+$ sedemikian sehingga,

$$a = mq + r, 0 \leq r < m$$

### C. Pembangkit Bilangan Acak

Pembangkit bilangan acak, akrab disebut LCG (*Linear Congruential Generator*), adalah salah satu metode sederhana untuk menghasilkan bilangan acak. Algoritma ini berbasis kekongruenan lanjar, yang dapat dituliskan dalam bentuk berikut.

$$X_n = (aX_{n-1} + b) \; \boldsymbol{mod} \; m$$

dengan,

$X_n$ = bilangan acak ke-n dari deret
$X_{n-1}$ = bilangan acak sebelumnya
$a$ = factor pengali
$b$ = *increment*
$m$ = modulus

dan $X_0$ merupakan kunci pembangkit (*seed).*

Ada beberapa keuntungan yang didapatkan dalam menggunakan pembangkit bilangan acak, yaitu implementasi yang tidak kompleks dan komputasi yang cepat. Adapun kerugian yang didapat adalah bilangan yang dihasilkan *predictable* karena pola yang berulang hingga periode tertentu.

## *D. SHA-1*

SHA-1 (*Secure Hash Algorithm 1)* adalah sebuah metode kriptografi yang menerima *input* data dengan ukuran berapa pun, dan diproses agar menghasilkan *output* berupa *hash* dengan panjang 20 *bytes*. Dengan memanfaatkan *hashing*, berikut adalah langkah-langkah dalam mengenkripsi pesan menggunakan SHA-1.

1. *Padding*, yaitu menambahkan bit pada akhir *input* yang dimasukkan agar panjang data menjadi kelipatan 512 bit. Sebagai contoh, akan dihitung *hash* dari pesan “rad”. Pertama, ubah terlebih dahulu setiap huruf yang ada pada kata “rad” dalam representasi kode ASCII, yaitu
$$r = 01110010,$$
$$a = 01100001$$
$$d = 01100100$$
Selanjutnya, akan ditambahkan *padding* hingga panjang data tersebut menjadi 512 bit (yang kemudian dinamakan 1 blok), yaitu dengan menambahkan bit 1 di akhir sebagai penanda akhir pesan, dan sisanya diberikan bit 0. Pada bagian akhir, ditambahkan kode *hex* dari panjang data asli, yaitu 24 bit. Sehingga, pesan tersebut sekarang akan menjadi sebagai berikut.
```
01110010 01100001 01100100 10000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00011000 00000000
```
2. Blok data yang telah dihasilkan, kemudian dipecah menjadi 16 kata 32-bit pertama ($W_0$ hingga $W_{15}$), dan expansi kemudian dilakukan untuk menghasilkan kata 16 ($W_{16}$) hingga 79 ($W_{79}$), yaitu dengan rumus berikut.
$$W_t = W_{t-3} \wedge W_{t-8} \wedge W_{t-14} \wedge W_{t-16}$$
3. Dalam proses SHA-1, terdapat 5 variabel yang telah diinisialisasi yaitu H0 = 0x67452301, H1 = 0xEFCDAB89, H2 = 0x98BADCFE, H3 = 0x10325476, H4 = 0xC3D2E1F0 yang kemudian nilai-nilai ini akan ditambahkan dengan nilai A,B,C,D, dan E yang diperoleh dari iterasi yang ditentukan oleh fungsi yang bergantung pada nilai $t$.

| F | F(x) | t |
|---|---|---|
| F1 | (B & C) \| (!B & D) | $0 \le t < 20$ |
| F2 | B ^ C ^ D | $20 \le t < 40$ |
| F3 | (B & C) \| (B & D) \| (C & D) | $40 \le t < 60$ |
| F4 | B ^ C ^ D | $60 \le t < 80$ |

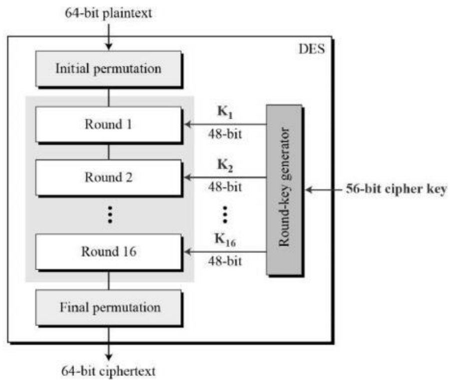
*Gambar 1. Fungsi pada SHA-1*

4. Setelah melakukan iterasi sebanyak 80 kali, akhirnya akan diperoleh nilai *hash* dari *input* “rad”, yaitu “bade5d26b78d94e5efb27f8cf03d43b298f69915”, yang berisikan 20 *bytes*.

## *E. DES*

*DES* (*Data Encryption Standard*) adalah algoritma cipher blok yang telah dijadikan standar algoritma enkripsi kunci-simetri. *DES* beroperasi pada pesan yang dibagi menjadi blok berukuran 64 bit. *DES* mengenkripsi 64 bit *plainteks* menjadi 64 bit *ciphertext* dengan menggunakan 56 bit kunci internal. Kunci ini diperoleh dari kunci eksternal yang diberikan oleh pengguna, yang memiliki panjang 64 bit [1].

Secara garis besar, terdapat 3 langkah utama dalam DES. Pertama, data 64-bit yang dimasukkan di permutasi dengan matriks permutasi awal. Hasil permutasi tersebut kemudian di-*enchipering* sebanyak 16 kali, dan setiap putaran memiliki *subkey* yang berbeda dari kunci utama. Hasil dari Langkah tersebut akan kembali di permutasi menggunakan matriks permutasi balikan, dan menghasilkan blok *cipherteks*. Pada makalah ini, penulis tidak akan terlalu fokus menjelaskan cara kerja DES (karena cukup panjang).




*Gambar 2. Cara kerja DES*

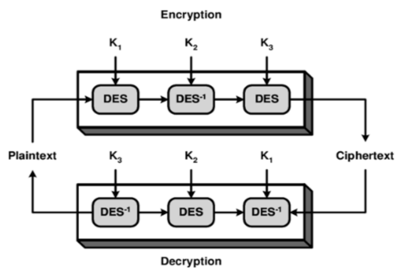
*(Sumber: https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm)*

## *F. Triple DES*

Algoritma DES sering kali memiliki potensi kelemahan. DES dapat dipecah dengan melakukan *brute force attack* dengan alasan panjang *key* yang relative pendek. Untuk mencegah hal tersebut, dibuatlah variasi baru dari DES, yaitu Double DES dan Triple DES. Double DES dilakukan dengan mengeksekusi DES sebanyak dua kali, sedangkan Triple DES sebanyak tiga kali.

Bentuk sederhana dari penggunaan Triple DES adalah sebagai berikut.

$$Enkripsi: C = E_{k3}\left(E_{k2}\left(E_{k1}(P)\right)\right)$$
$$Dekripsi: P = D_{k1}\left(D_{k2}\left(D_{k3}(C)\right)\right)$$

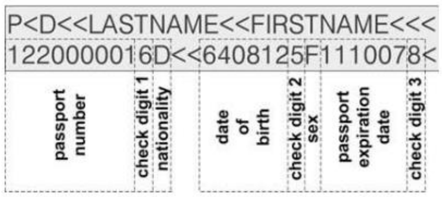



*Gambar 3. Cara kerja 3DES*

*(Sumber: https://www.splunk.com/en_us/blog/learn/triple-des-data-encryption-standard.html)*

## *G.* MRZ *e-Passport*

Pada setiap *e-Passport*, akan terdapat 24 *byte code* yang disebut MRZ (*Machine Readable Zone).* MRZ terdiri dari 2 baris yang berisi informasi terkait pemilik dari paspor tersebut, seperti nama, jenis kelamin, tanggal lahir, dan kewarganegaraan [2]. Data dalam MRZ diformat sedemikian rupa agar dapat dibaca oleh mesin dengan kemampuan standar di seluruh dunia. Dalam pengecekan kevalidan, baris kedua merupakan kode yang sangat krusial, karena merupakan penentu dari kunci BAC.




*Gambar 4. MRZ pada e-passport (Sumber: Yifei Liu, 2007, hlm. 1533)*

Adapun hal-hal yang tercantum pada baris kedua adalah sebagai berikut:

- Kode Paspor yang terdiri atas simbol *alphanumeric* dan bersifat unik.
- Tanggal kelahiran dari pemilik paspor, berisi 6 karakter $(Y_{1a}Y_{1b}M_{1a}M_{1b}D_{1a}D_{1b})$
- Tanggal kadaluwarsa dari paspor $(Y_{2b}M_{2a}M_{2b}D_{2a}D_{2b}C_3)$
- Ketiga informasi tersebut memiliki cek digit masing-masing, untuk validasi kebenaran kode $(C_1, C_2, C_3)$

| |
|---|
| $P < D < LASTNAME << FIRSTNAME <<<$ |
| $A_1A_2A_3 \dots A_9C_1N \ll Y_{1a}Y_{1b}M_{1a}M_{1b}D_{1a}D_{1b}C_2SY_{2a}Y_{2b}M_{2a}M_{2b}D_{2a}D_{2b}C_3 <$ |

*Gambar 5. Notasi MRZ secara umum*

Untuk memastikan **kebenaran kode** (bukan kebenaran identitas), kita dapat menerapkan Algorithma Luhn dengan bobot 7,3, dan 1 untuk setiap informasi yang ada. Misalkan dipunyai kode paspor yang terdiri atas 9 digit, maka cara perhitungannya adalah sebagai berikut.

$$(7A_1 + 3A_2 + A_3 + 7A_4 + 3A_5 + A_6 + 7A_7 + 3A_8 + A_9)\ \boldsymbol{mod}\ 10 = C_1$$

Jika muncul suatu huruf, maka nilai $A_i$ dari huruf yang bersangkutan adalah representasi ASCII $(A = 10, B = 11, \dots Z = 35)$. Jika persamaan tersebut benar, maka kode e-paspor yang diberikan valid. Sebagai contoh, jika dipunyai kode *e-passport* "1220000016", maka jika dicek menggunakan persamaan diatas, akan menghasilkan

$$(7 + 6 + 2 + 1)\ \boldsymbol{mod}\ 10 = 6 = C_1\ [valid].$$

## *H. Basic Access Control*

Dalam memvalidasi kebenaran paspor, diperlukan protokol *three-pass challenge-response* untuk autentikasi dan pembentukan kunci antara dua pihak, yaitu IFD (*Inspection System)* dan *Contactless IC (*chip pada e-passport). Protokol ini digunakan agar memastikan komunikasi yang aman menggunakan algoritma kriptografi 3DES dan MAC.

Pertama-tama, akan dikenalkan istilah RND.IC dan RND.IFD yang secara berturut-turut adalah *nonce* yang dihasilkan oleh IC dan IFD. Nonce adalah angka atau *string* acak yang digunakan untuk mencegah serangan berulang. Secara detail, langkah-langkah yang akan ditempuh adalah sebagai berikut.

1. **IC menghasilkan *nonce***
   IFD akan mengirimkan perintah "*GET CHALLENGE"* untuk meminta kode RND.IC, dan IC merespons dengan mengirimkan *nonce* tersebut. *Nonce* ini akan berisi 8 *byte* secara acak.
2. **IFD mengirimkan *challenge*.**
   Tahap selanjutnya yaitu IFD akan membuat RND.IFD (bilangan acak) dan K.IFD. Data tersebut kemudian digabungkan dalam format berikut.
   $$S = RND.IFD\ ||RND.IC||K.IFD$$
   Penggabungan tersebut kemudian akan dienkripsi menggunakan $K_{enc}$, menghasilkan EIFD yang kemudian hasilnya akan di MAC (Message Authentication Control) dengan $K_{mac}$. Terakhir, IFD akan mengirimkan hasil penggabungan EIFD dengan MIFD ke IC.

3. **Verifikasi dan respons IC**
   Setelah mendapatkan data yang dibutuhkan, IC kemudian akan mendekripsi pesan-pesan yang telah di enkripsi sebelumnya, dan memeriksa apakah RND.IC yang didapat sesuai. Selanjutnya, data/pesan yang telah ada akan digabungkan sebagai berikut
   $$R = RND.IC\ ||RND.IFD||K.IC$$
   Dengan cara yang sama seperti sebelumnya, R akan dienkripsi menjadi EIC menggunakan kunci enkripsi $K_{enc}$, yang kemudian hasilnya akan di MAC menggunakan $K_{mac}$. Kemudian, dikirimkan hasil dari penggabungan EIC dengan MIC ke IFD.

4. **Verifikasi respons IC**
   Setelah mendapatkan pesan terbaru, pesan tersebut kemudian akan didekripsi untuk mendapatkan pesan yang terenkripsi, dan memeriksa apakah RND.IFD sesuai.

5. **Derivasi *session key***
   IFD dan IC kemudian akan menghitung kunci $K_{SENC}$ dan $K_{SMAC}$ masing-masing yang kemudian akan diverifikasi apakah nilai keduanya sama.

Untuk memastikan kevalidan antara kode MRZ dengan *chip* yang terdapat pada *e-passport,* semua Langkah yang dilalui harus memberikan kebenaran, yaitu,

- RND.IC yang dikirim IC sama dengan RND.IC yang diterima IFD

- RND.IFD yang dikirim IFD sama dengan RND.IFD yang diterima IC
- $K_{SENC}$ dan $K_{SMAC}$ yang dihitung oleh IFD dan IC haruslah sama.

## III. IMPLEMENTASI DAN PEMBAHASAN

Berikut adalah implementasi dari fungsi-fungsi yang digunakan. Untuk memudahkan contoh, penulis akan menganggap bahwa paspor yang akan dicek memiliki ID MRZ “L898902C<369080619406236”

### A. *check_digit_test*

Fungsi ini akan mengecek kevalidan dari kode yang tertera pada e-Passport. Input berupa string yang akan di cek tersebutm, dan output berupa Boolean, yang mengahasilkan true jika data yang diberikan valid. Untuk setiap informasi yang tersedia (kode *e-passport*, tanggal lahir, dan tanggal kadaluarsa), akan di cek kesamaan *check_digit* masing-masing informasi dengan bobot 7,3, dan 1. Berikut adalah implementasinya.

```
def check_digit_test(data: str) -> bool:
    # bobot untuk cek digit
    weights = [7, 3, 1]

    # CEK VALIDITAS KODE PASSPORT
    # mehitung total bobot dari data
    total = 0
    i = 0
    while (i < len(data)):
        if (data[i] == '<'):
            i += 1
            break
        elif (data[i].isdigit()):
            total += int(data[i]) * weights[i % 3]
        elif (ord(data[i]) >= 65 and ord(data[i]) <= 90):
            total += (ord(data[i]) - 55 + 10) * weights[i % 3]
        i += 1
    cek_valid_kode_passport = (total % 10) == int(data[i])
    print("Validitas kode passport: ", cek_valid_kode_passport)
```

*Gambar 6. Validasi kode passport*

```
    # CEK VALIDITAS TANGGAL LAHIR
    i += 1
    counter = 0
    total = 0
    while (counter < 6):
        total += int(data[i]) * weights[counter % 3]
        counter += 1
        i += 1
    cek_valid_birth = (total % 10) == int(data[i])
    print("Validitas tanggal lahir: ", cek_valid_birth)
```

*Gambar 7. Validasi tanggal lahir*

```
    # CEK VALIDITAS TANGGAL KADALUARSA

    i += 1
    counter = 0
    total = 0
    while (counter < 6):
        total += int(data[i]) * weights[counter % 3]
        counter += 1
        i += 1

    cek_valid_expired = (total % 10) == int(data[i])
    print("Validitas tanggal kadaluarsa: ", cek_valid_birth)

    return cek_valid_kode_passport and cek_valid_birth and cek_valid_expired
```

*Gambar 8. Validasi tanggal kadaluarsa*

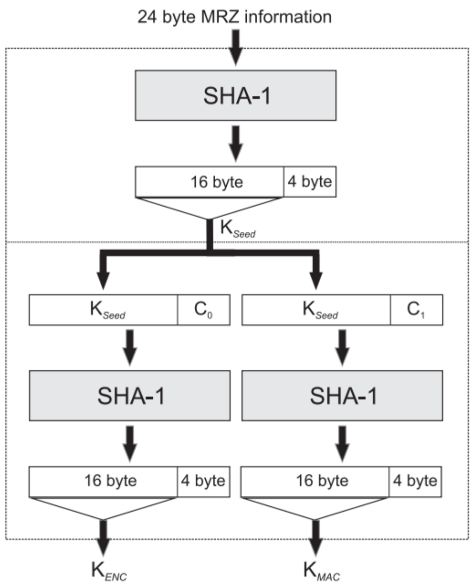
Jika diinput “L898902C<369080619406236” pada kode tersebut, berikut adalah hasilnya, yang menandakan bahwa kode tersebut valid.

```
Masukkan data: L898902C<369080619406236
Validitas kode passport:  True
Validitas tanggal lahir:  True
Validitas tanggal kadaluarsa:  True
```

*Gambar 9. Hasil cek*

### B. *generate_kenc_kmac*

Terdapat dua kunci utama yang akan digunakan dalam proses selanjutnya, yaitu $K_{enc}$ (*Encryption Key*) dan $K_{mac}$ (*Message Authentication Code Key*). $K_{enc}$ digunakan untuk mengenkripsi data saat pembacaan paspor, sedangkan $K_{mac}$ digunakan untuk memastikan integritas pesan dan autentikasi data yang diperoleh. Ada beberapa cara untuk memperoleh kedua nilai *key* tersebut, namun pada makalah ini penulis akan menggunakan metode SHA-1 untuk memperolehnya. Setelah diproses menggunakan SHA-1, akan diperoleh suatu hash dengan panjang 20 *bytes*. Dari hasil tersebut, $K_{enc}$ diperoleh dengan mengambil 8 *bytes* pertama, sedangkan $K_{mac}$ diperoleh dengan mengambil 8 *bytes* berikutnya. Untuk memperolehnya, data yang dimasukkan sebagai input adalah 24-*byte* MRZ yang tertera. Secara garis besar, berikut adalah langkah utama dalam menentukan $K_{enc}$ dan $K_{mac}$.




*Gambar 10. Cara memperoleh kenc dan kmac(Sumber: Yifei Liu, 2007, hlm. 1534)*

Nilai $C_0$ dan $C_1$ secara berturut-turut adalah “00000001“ dan “00000002”. Kedua nilai tersebut perlu ditambahkan untuk membuat proses deviasi lebih fleksibel dan aman. Sebelum memberikan implementasi pembuatan $K_{enc}$ dan $K_{mac}$, penulis akan terlebih dahulu memberikan implementasi fungsi *change_parity* yang akan digunakan kemudian. Fungsi ini menerima input berupa string, yang kemudian akan diubah paritasnya. Setiap *byte* yang ada, akan dikonversi menjadi bilangan biner. Jika banyak bit ‘1’ adalah ganjil, maka akan dicari bit ‘0’ paling akhir dan diubah nilainya menjadi ‘1’.

```python
def change_parity(key: str) -> str:
    result = ""
    for i in range(0, len(key), 2):
        temp = ""
        hex_value = int(key[i:i+2], 16)
        binary = format(hex_value, '08b')
        count_ones = binary.count('1')
        if count_ones % 2 == 0:
            for j in range(7, -1, -1):
                if binary[j] == '0':
                    binary = binary[:j] + '1' + binary[j+1:]
                    break
        temp += format(int(binary, 2), '02x')
        result += temp
    return result
```

*Gambar 11. Fungsi change_parity*

Berikut adalah implementasi dari fungsi *generate_kenc_kmac*.

```python
def generate_kenc_kmac(data: str) -> tuple:
    hash_result = hashlib.sha1(data.encode('utf-8')).digest()

    kseed = ''.join([format(byte, '02x') for byte in hash_result[:16]])
```

*Gambar 12. Memperoleh nilai kseed*

```python
    # GENERATE KENC DENGAN PADDING 00000001

    kenc_padding = "00000001"
    kseed += kenc_padding
    kseed_bytes = bytes.fromhex(kseed)

    # HASH KEMBALI KSEED UNTUK MEMPEROLEH NILAI BARU
    key = hashlib.sha1(kseed_bytes).digest()
    key_string = ''.join([format(byte, '02x') for byte in key])

    # MENGAMBIL 16 BYTES PERTAMA DAN KEDUA
    Ka = key_string[:16]
    Kb = key_string[16:32]

    # MENYESUAIKAN PARITAS ka dan kb
    result_ka = change_parity(Ka)
    result_kb = change_parity(Kb)

    Kenc = ""
    Kenc += result_ka + result_kb
```

*Gambar 13. Memperoleh nilai Kenc*

```python
    # GENERATE KMAC DENGAN PADDING 00000002

    kseed = ''.join([format(byte, '02x') for byte in hash_result[:16]])
    kmac_padding = "00000002"
    kseed += kmac_padding
    kseed_bytes = bytes.fromhex(kseed)

    # HASH KEMBALI KSEED UNTUK MEMPEROLEH NILAI BARU
    key = hashlib.sha1(kseed_bytes).digest()
    key_string = ''.join([format(byte, '02x') for byte in key])

    # MENGAMBIL 16 BYTES PERTAMA DAN KEDUA
    Ka = key_string[:16]
    Kb = key_string[16:32]

    # MENYESUAIKAN PARITAS ka dan kb
    result_kam = change_parity(Ka)
    result_kbm = change_parity(Kb)

    Kmac = ""
    Kmac += result_kam + result_kbm

    return Kenc, Kmac
```

*Gambar 14. Memperoleh nilai Kmac*

Jika diinput "L898902C<369080619406236" pada kode tersebut, berikut adalah hasilnya, yang menunjukkan nilai-nilai yang berkaitan

```
Masukkan data: L898902C<369080619406236
kseed:  239ab9cb282daf66231dc5a4df6bfbae
Ka (enc):  ab94fcedf2664edf
Kb (enc):  b9b291f85d7f77f2
result_ka (enc):  ab94fdeff2674fdf
result_kb (enc):  b9b391f85d7f7ff2
Ka (mac):  7862d9ece03c1bcd
Kb (mac):  4d77089dcf131442
result_ka (mac):  7962d9ece03d1fcd
result_kb (mac):  4f7f089ddf131543
Kenc:  ab94fdeff2674fdfb9b391f85d7f7ff2
Kmac:  7962d9ece03d1fcd4f7f089ddf131543
```

*Gambar 15. Contoh output fungsi generate_kenc_kmac*

## *C. lcg_next dan lcg_main*

Kedua fungsi ini digunakan untuk menghasilkan suatu bilangan hex acak dengan ukuran *bytes* tertentu. Bilangan ini kemudian akan digunakan untuk menentukan nilai dari RND.IC, RND.IFD, dan K.IFD. Dalam mengimplementasikannya, penulis menggunakan nilai $a = 1597, c = 51479,$ dan $m = 244944$. Perbedaan utama kedua fungsi tersebut adalah *lcg_next* menghasilkan integer, sedangkan *lcg_main* menghasilkan hex. Penulis membutuhkan *library* time yang digunakan untuk menentukan nilai dari seed yang digunakan. Berikut adalah implementasinya.

```python
def lcg_next(seed, a=1597, c=51749, m=244944):
    return (a * seed + c) % m
```

*Gambar 16. Fungsi lcd_next*

```python
def lcg_main(length):

    # MENGGUNAKAN WAKTU SEBAGAI SEED
    current_seed = int(time.time() * 1000000)

    hex_chars = '0123456789ABCDEF'
    result = ''

    # DIKALIKAN 2 KARENA SETIAP 2 KARAKTER HEXA MEMBUTUHKAN 1 BYTE
    iteration = (length*2 + 1) // 2

    for _ in range(iteration):
        current_seed = lcg_next(current_seed)
        hex1 = hex_chars[current_seed % 16]
        hex2 = hex_chars[(current_seed // 16) % 16]
        result += hex1 + hex2

    return result[:length*2]
```

*Gambar 17. Fungsi lcg_main*

## *D. send_challange_IFD*

Saat IFD memberikan challenge, ada beberapa hal yang harus diinisialisasi menggunakan random number generator, yaitu RND_IC (8 *bytes*), RND_IFD (16 *bytes*), dan K_IFD (16 *bytes*). Ketiga nilai tersebut kemudian digabungkan dan akan menghasilkan $E_{IFD}$ dan $M_{IFD}$ menggunakan algoritma 3DES dengan mode ECB dan CBC. Fungsi *send_challange_IFD* menerima masukkan berupa kedua nilai $K_{enc}$ dan $K_{mac}$ dalam tipe *string*, nilai dari RND.IC dan menghasilkan keluaran berupa RND.IFD, K.IFD, dan data yang merupakan gabungan dari $E_{IFD}$ dan $M_{IFD}$. Dalam implementasinya, penulis menggunakan *library crypto*. Berikut adalah kodenya.

```python
def send_challange_IFD(kenc: str, kmac: str, rnd_ic:str) -> tuple:

    rnd_ifd = lcg_main(8)
    k_ifd = lcg_main(16)

    s = ""
    s += rnd_ic + rnd_ifd + k_ifd

    data = bytes.fromhex(s)

    # PADDING DATA YANG DIBUTUHKAN
    kenc_24 = kenc + kenc[:16]

    # MENYESUAIKAN PARITAS
    kenc_24 = change_parity(kenc_24)

    # CONVERT KE BYTES
    key_bytes_enc = bytes.fromhex(kenc_24)

    # ENKRIPSI DATA MENGGUNAKAN DES3
    cipher = DES3.new(key_bytes_enc, DES3.MODE_ECB)
    Eifd = cipher.encrypt(data)
```

*Gambar 18. Memperoleh nilai Eifd*

```python
## ULANGI CARA YANG SAMA UNTUK KMAC
kmac_24 = kmac + kmac[:16]

# MENYESUAIKAN PARITAS
kmac_24 = change_parity(kmac_24)

# CONVERT KE BYTES
key_bytes_mac = bytes.fromhex(kmac_24)

# ENKRIPSI DATA MENGGUNAKAN DES3
iv = b'\x00' * 8
cipher = DES3.new(key_bytes_mac, DES3.MODE_CBC,iv)

Mifd = cipher.encrypt(Eifd)
Mifd = Mifd[-8:]

res = ""
Eifd_string = Eifd.hex().upper()
Mifd_string = Mifd.hex().upper()
res += Eifd_string + Mifd_string
return rnd_ifd,k_ifd,res
```

*Gambar 19. Memperoleh nilai Mifd*

### E. *create_eic*

Fungsi ini menerima masukkan berupa K.IFD, RND.IC, dan RND.IFD, dan memberikan *output* berupa EIC yang nantinya akan dibandingkan nilainya dengan RND.IC.

```python
def create_eic(kifd: str, rnd_ic: str, rnd_ifd:str):

    kic = lcg_main(16)
    kifd_bytes = bytes.fromhex(kifd)
    kic_bytes = bytes.fromhex(kic)
    kseed = bytes(a ^ b for a, b in zip(kifd_bytes, kic_bytes)).hex()
    ksenc,ksmac = generate_kenc_kmac(kseed,True)

    # MENGGABUNGKAN RND.IC, RND.IFD, DAN K.IC
    r = ""
    r += rnd_ic + rnd_ifd + kic
    print("r: ",r)
    data = bytes.fromhex(r)

    # PADDING DATA YANG DIBUTUHKAN
    kenc_24 = ksenc + ksenc[:16]

    # CONVERT KE BYTES
    key_bytes_enc = bytes.fromhex(kenc_24)

    # ENKRIPSI DATA MENGGUNAKAN DES3
    cipher = DES3.new(key_bytes_enc, DES3.MODE_ECB)
    Eic = cipher.encrypt(data)

    Eic_string = Eic.hex().upper()

    return Eic_string
```

*Gambar 20. Fungsi create_eic*

### F. *decrypt_eic*

Setelah mendapakan nilai EIC, langkah terakhir yang diperlukan adalah mendekripsi nilai tersebut, untuk memperoleh RND.IC yang telah dibuat pada awal program. Fungsi ini menerima masukkan nilai EIC dan *key* $K_{senc}$, sedangkan keluaran dari fungsi ini adalah ketiga nilai dari RND.IC, RND.IFD, dan KIC.

```python
def decrypt_eic(eic: str, ksenc: str) -> tuple:
    kenc_24 = ksenc + ksenc[:16]
    key_bytes_enc = bytes.fromhex(kenc_24)

    # DEKRIPSI EIC
    cipher = DES3.new(key_bytes_enc, DES3.MODE_ECB)
    decrypted_data = cipher.decrypt(bytes.fromhex(eic))

    r = decrypted_data.hex().upper()
    rnd_ic = r[:16]
    rnd_ifd = r[16:32]
    kic = r[32:]

    return rnd_ic, rnd_ifd, kic
```

*Gambar 21. Fungsi decrypt_eic*

### G. *check_validitas*

Setelah diperoleh ketiga nilai tersebut, langkah selanjutnya adalah mengecek kesamaan nilainya dengan angka-angka yang telah di-*generate* di awal. Jika ketiga nilainya sama, maka kode passport tersebut berhasil.

```python
def check_validitas(eic,ksenc,rnd_ic, rnd_ifd, kic):
    to_check = decrypt_eic(eic,ksenc)
    return to_check[0] == rnd_ic and to_check[1] == rnd_ifd and to_check[2] == kic
```

*Gambar 22. Fungsi check_validitas*

## IV. Kesimpulan Dan Saran

Berdasarkan implementasi yang telah dilakukan, penggunaan aritmatika modular dan hashing dalam proses enkripsi serta verifikasi terbukti mampu meningkatkan keamanan e-Passport secara signifikan. Teknik transformasi data menjadi kunci enkripsi, ditambah dengan penerapan pembangkit bilangan acak, berhasil menjaga validitas data sekaligus mencegah akses tidak sah. Hasil ini mengindikasikan bahwa pendekatan kriptografi berbasis aritmatika modular memiliki potensi besar untuk diterapkan dalam sistem keamanan modern.

Pada makalah ini, penulis belum sepenuhnya memanfaatkan informasi *message* yang didapat berdasarkan fungsi *send_challange_IFD*. Penelitian selanjutnya dapat lebih berfokus dan memanfaatkan hasil dari pesan tersebut.

## V. Ucapan Terima Kasih

Saya mengucapkan rasa syukur kepada Allah SWT, yang karena Rahmat-Nya, saya diberi kemudahan dalam menyelesaikan makalah ini dengan baik dan tepat waktu. Selain itu, saya mengucapkan terima kasih kepada Dr. Ir. Rinaldi, M.T. selaku dosen mata kuliah IF 1220 Matematika Diskrit yang sudah mengajarkan materi semester 3 ini dengan sangat baik dan menyenangkan. Tak lupa saya ucapkan terima kasih kepada kedua orang tua saya yang telah banyak membantu dan mendukung saya selama proses pengerjaan makalah. Saya juga ingin mengucapkan terima kasih kepada teman-teman yang tidak dapat saya sebutkan satu-satu, atas bantuan dan dukungannya dalam pembuatan makalah ini.

## DAFTAR PUSTAKA


[1] Munir, Rinaldi. *Kriptografi*, 2nd ed., Bandung: Penerbit INFORMATIKA, 2019.

[2] Munir, Rinaldi, 2024. "Teori Bilangan (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/15-Teori-Bilangan-Bagian1-2024.pdf

[3] Munir, Rinaldi, 2024. "Teori Bilangan (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/16-Teori-Bilangan-Bagian2-2024.pdf

[4] Munir, Rinaldi, 2024. "Teori Bilangan (Bagian 3)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/17-Teori-Bilangan-Bagian3-2024.pdf

[5] Y. Liu, T. Kasper, K. Lemke-Rust, and C. Paar, *"E-Passport: Cracking Basic Access Control Keys,"* Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany.

[6] *Doc 9303: Machine Readable Travel Documents, Part 3: Security Mechanisms for MRTDs*, 8th ed., ORGANIZACIÓN DE AVIACIÓN CIVIL INTERNACIONAL, 2021.

[7] *Doc 9303: Machine Readable Travel Documents, Part 11: Security Mechanisms for MRTDs*, 8th ed., ORGANIZACIÓN DE AVIACIÓN CIVIL INTERNACIONAL, 2021.

[8] "Data Encryption Standard," *TutorialsPoint*. Diakses 6 Januari 2025. https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm


## 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, 6 Januari 2025

Azfa Radhiyya Hakim, 13523115