Trident Cipher

Nixon Andika, Jan Meyer Saragih

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika (STEI), Institut Teknologi Bandung (ITB), Jalan Ganesha 10, Bandung 40132
E-mail: 13517059@std.stei.itb.ac.id, 13517131@stei.itb.ac.id

Abstrak. Dalam jurnal ini, kami mengajukan sebuah cipher blok baru, yaitu Trident Cipher. Trident Cipher merupakan cipher blok baru dengan ukuran blok 64-bit dan ukuran kunci 64-bit. Trident cipher didesain dengan mengutamakan keamanan melalui penggunaan berbagai operasi modulo, *xor*, rotasi, substitusi, dan transposisi. Algoritma ini menghubungkan jaringan Feistel pada umumnya dengan algoritma pembangkitan key dan fungsi-f yang baru. Algoritma ini melakukan substitusi dan transposisi yang tergolong kompleks terkait merupakan *block cipher*. **Kata Kunci**: kriptografi, *Trident*, kunci, substitusi, transposisi.

1. Pendahuluan

Dalam dunia *modern*, kebutuhan akan keamanan data terus meningkat. Terdapat banyak data *personal* maupun sensitif yang tidak boleh diketahui oleh pihak lain. Salah satu metode untuk mengamankan data adalah dengan mengenkripsi data tersebut menjadi data yang tidak dapat dibaca oleh pihak lain. Pengenkripsian data dilakukan menggunakan sebuah *cipher*, baik berupa *stream cipher* maupun *block cipher*. Jurnal ini memperkenalkan sebuah *block cipher* baru dengan nama ***Trident Cipher.*** *Trident cipher* menggunakan operasi modulo sederhana, *xor*, *shift*, substitusi dan transposisi. *Trident cipher* didesain secara aman dengan menggunakan berbagai variasi reduksi, ekspansi, substitusi, rotasi, dan transposisi.

Terdapat banyak teknik kriptografi *modern* yang telah dihasilkan hingga sekarang ini. Beberapa jenis standar atau algoritma kriptografi *modern* yang merupakan inspirasi dari *Trident Cipher* adalah DES (*Data Encryption Standard*), AES (*Advanced Encryption Standard*), dan *MARC Cipher*.

DES merupakan salah satu standar kriptografi blok *modern* yang berasal dari IBM pada tahun 1972[1]. Algoritma dari DES dinamakan DEA (*Data Encryption Algorithm*). Algoritma ini merupakan algoritma yang memanfaatkan jaringan *Feistel* yang berorientasi bit1. Pada algoritma ini, blok berukuran 64-bit dan dilakukan 16 kali putaran enkripsi dengan panjang kunci internal 48-bit. Kunci enkripsi untuk setiap putaran dibangkitkan melalui kunci eksternal dengan memanfaatkan algoritma ekspansi, permutasi, dan shift. Setelah itu, kunci digunakan untuk mendapatkan hasil fungsi *f* dari jaringan *Feistel*. Kunci digunakan untuk melakukan eksklusif *or* dengan hasil ekspansi teks yang akan dienkripsi. Setelah itu dilakukan substitusi menggunakan kotak-S dan permutasi lagi untuk mendapatkan hasil dari fungsi *f*.

AES juga merupakan salah satu standar kriptografi blok *modern*. Algoritma yang dijadikan standar untuk AES adalah algoritma *Rijndael* yang ditetapkan pada November 2001. Algoritma ini juga merupakan algoritma yang memanfaatkan jaringan *Feistel* yang berorientasi byte. Pada algoritma ini, blok berukuran 128-bit dan dilakukan 10 kali putaran enkripsi dengan panjang kunci internal 128-bit.

Algoritma ini membagi byte-byte ke dalam bentuk matriks[2]. Kunci enkripsi untuk setiap putaran dibangkitkan melalui kunci eksternal. Setelah itu, kunci tersebut digabungkan dengan plainteks menggunakan algoritma substitusi byte, pergeseran baris, pencampuran kolom matriks, dan *xor* (eksklusif *or*).

MARC merupakan salah satu algoritma kriptografi blok *modern* yang diciptakan oleh Jeffrey S.-W. Hsiao. Algoritma ini merupakan algoritma yang dibuat dengan tujuan menghasilkan algoritma yang cepat namun tetap aman[3]. Cipher blok ini juga memanfaatkan jaringan *Feistel* dengan melakukan enkripsi antara 8 hingga 32 putaran. Operasi-operasi matematika yang digunakan pada MARC adalah *xor*, rotasi yang bergantung terhadap kunci, penjumlahan modulo, dan perkalian modulo.

Dari inspirasi ketiga *cipher* tersebut, dibentuk *Trident cipher*. *Trident cipher* merupakan algoritma kriptografi blok yang menggunakan 64-bit kunci dan 64-bit blok. Algoritma ini juga menggunakan jaringan *Feistel* sebanyak 8 putaran. *Trident cipher* mendapatkan inspirasi dari operasi-operasi matematika dari DES, AES, dan MARC. Operasi yang digunakan seperti S-Box pada AES, algoritma permutasi pada DES, dan algoritma penjumlahan, perkalian modulo, serta rotasi yang bergantung terhadap kunci pada MARC. Selain itu, *Trident cipher* juga menambahkan algoritma baru seperti substitusi pada bit yang bergantung pada kunci, penambahan kunci pada akhir plainteks, penjumlahan modulo dan perkalian modulo yang bergantung pada putaran, serta pembagian bit kunci pada saat menjalankan fungsi *f*.

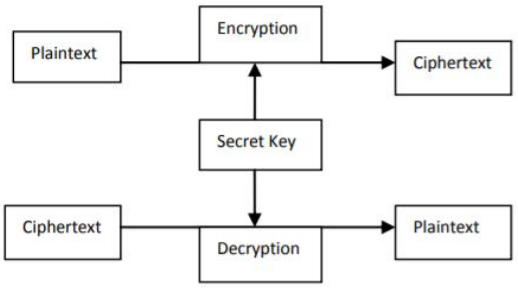
2. Studi Pustaka

2.1. Kunci Simetri

Terdapat dua kategori kriptografi tergantung dari jenis kunci yang digunakan untuk mengenkripsi atau mendekripsi data, yaitu teknik enkripsi simetri dan asimetri[4].

2.1.1. Kriptografi Simetri / Single Key Cryptography

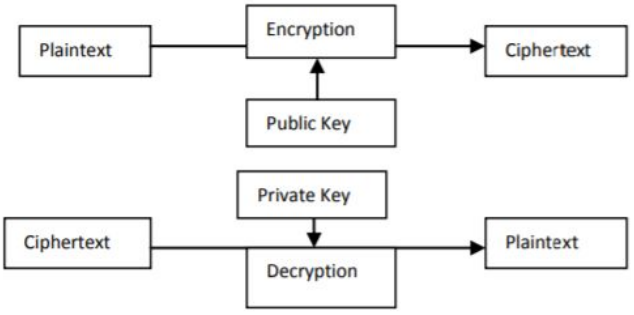
Jenis kriptografi ini menggunakan satu kunci yang sama untuk melakukan enkripsi dan dekripsi. Pengirim dan penerima pesan harus setuju dengan sebuah kunci rahasia yang mereka berdua akan gunakan untuk enkripsi dan dekripsi. Proses kriptografi simetri dapat dilihat pada gambar 2.1.



Gambar 2.1. Proses kriptografi simetri.

2.1.2. Kriptografi Asimetri / Public Key Cryptography

Jenis kriptografi ini menggunakan dua kunci, yaitu kunci *public* dan kunci *private*. Kunci *public* merupakan kunci yang diketahui oleh orang lain sedangkan kunci *private* merupakan kunci yang hanya diketahui oleh pemilik kunci. Kunci *public* digunakan untuk mengenkripsi pesan dan kunci *private* digunakan untuk mendekripsi pesan. Data yang dienkripsi oleh suatu kunci *public* hanya dapat didekripsi oleh kunci *private* yang cocok. Proses kriptografi asimetri dapat dilihat pada gambar 2.2.



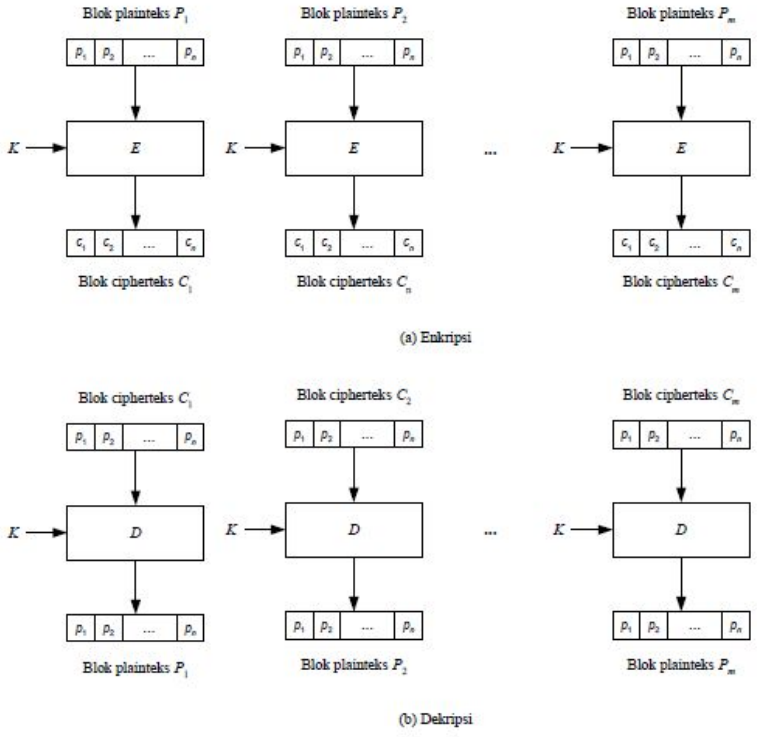
Gambar 2.2. Proses kriptografi asimetri.

2.2. Mode Enkripsi Dekripsi

Terdapat lima *mode* enkripsi dekripsi untuk *block cipher*, yaitu *ECB*, *CBC*, *CFB*, *OFC*, dan *counter*[4].

2.2.1. ECB (Electronic Code Book)

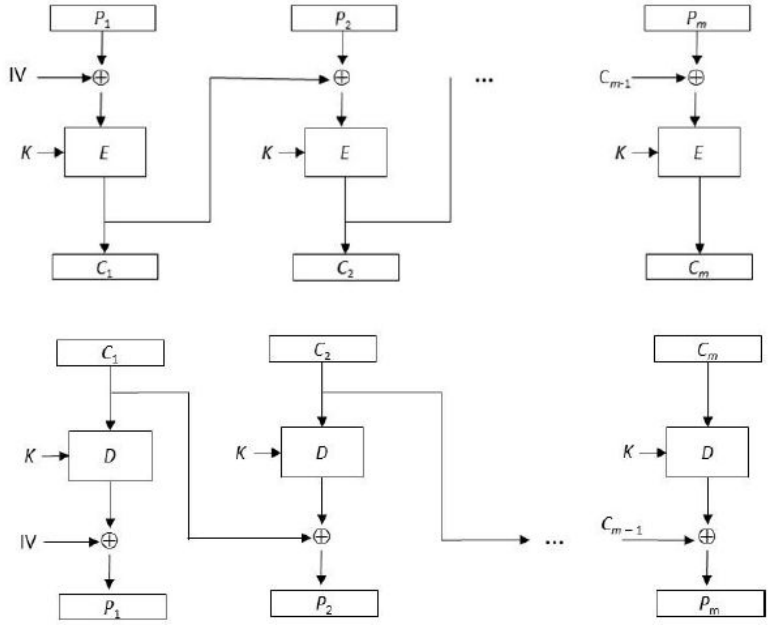
Pada *mode* ini, data dibagi menjadi blok berukuran 64 bit dan setiap blok dienkripsi atau didekripsi satu per satu. Karena dienkripsi atau didekripsi satu per satu, setiap blok independen dari blok lain. Jika terjadi *error*, hanya blok itu saja yang terpengaruh sedangkan blok lain tidak ikut mengalami *error* tersebut. Karena independen pula, blok *plaintext* yang sama akan selalu dienkripsi menjadi blok *ciphertext* yang sama sehingga terdapat kemungkinan pembuatan buku kode *plaintext* dan *ciphertext* yang berkoresponden. ECB merupakan *mode* paling lemah dibandingkan *mode* lainnya karena tidak ada keamanan tambahan, tetapi ECB merupakan *mode* yang paling cepat dan paling mudah untuk diimplementasi. Proses *ECB* dapat dilihat pada gambar 2.3.



Gambar 2.3. Proses *ECB*.

2.2.2. CBC (Cipher Block Chaining)

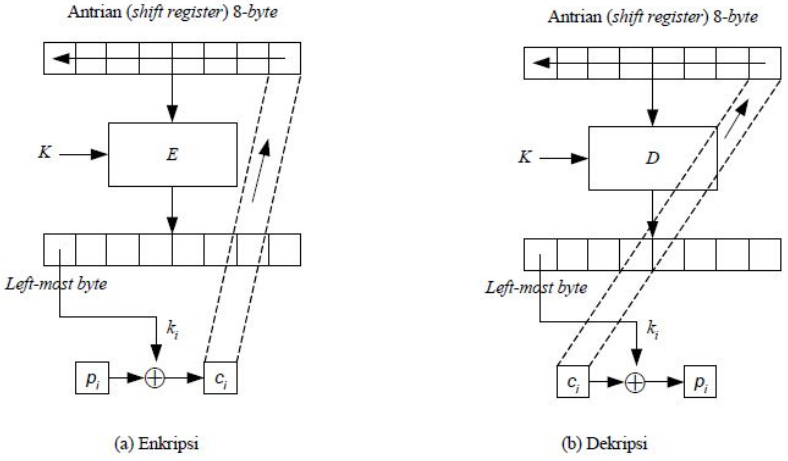
Pada *mode* ini, setiap blok hasil ECB akan di-*xor*-kan dengan blok *plaintext* selanjutnya yang akan dienkripsi. Khusus untuk blok pertama, dilakukan *xor* dengan sebuah blok *Initialization Vector* (IV) yang dapat dibentuk secara bebas. Karena dilakukan *xor* dengan hasil blok *ciphertext* sebelumnya, blok-blok pada CBC bersifat dependen terhadap hasil blok sebelumnya. Hal ini berarti jika ingin menemukan *plaintext* dari suatu blok, perlu diketahui blok *ciphertext*, kunci, dan blok *ciphertext* sebelumnya. Jika terjadi *error*, kesalahan *error* ini akan mempengaruhi semua blok selanjutnya. CBC lebih aman dari ECB karena penambahan langkah *xor* terhadap blok *ciphertext* sebelumnya. Proses CBC dapat dilihat pada gambar 2.4.



Gambar 2.4. Proses *CBC*.

2.2.3. CFB (Cipher Feedback)

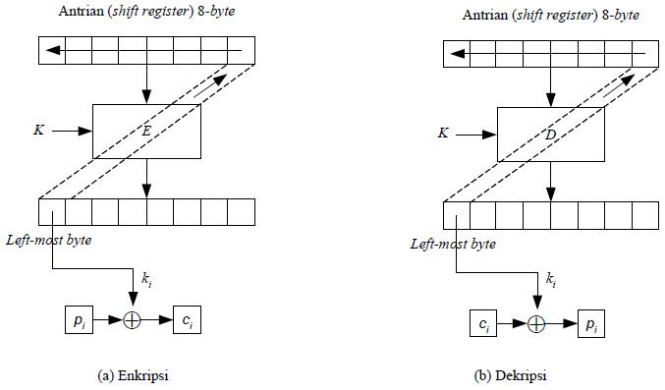
Pada *mode* ini, dilakukan enkripsi data dalam *unit* yang lebih kecil daripada ukuran blok. Pada CFB, digunakan *shift register* yang digunakan sebagai nilai *input*. M-bit dari *shift register* akan dienkripsi dan nilainya digunakan untuk melakukan XOR dengan *plaintext* untuk menghasilkan *ciphertext*. *Ciphertext* yang dihasilkan akan digunakan pula sebagai nilai *shift register* yang baru untuk menghasilkan *ciphertext* blok selanjutnya. Seperti CBC, terjadinya *error* akan menyebabkan semua blok selanjutnya juga mengalami *error*. Proses CFB dapat dilihat pada gambar 2.5.



Gambar 2.5. Proses CFB.

2.2.4. OFC (Output FeedBack)

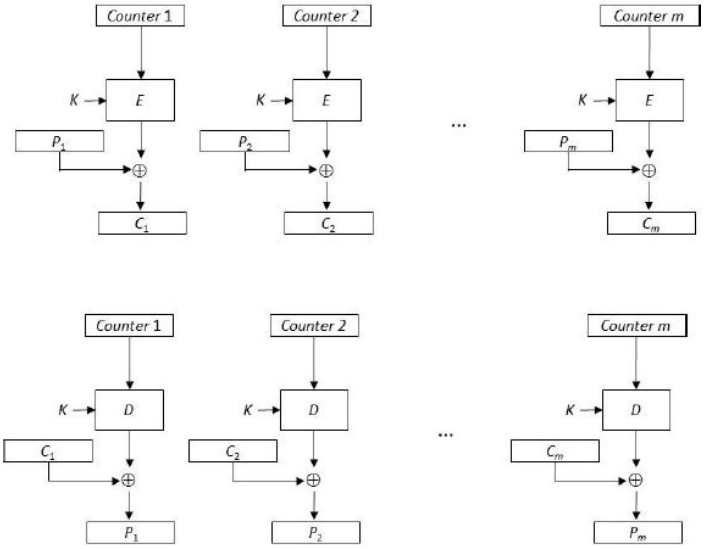
OFC mirip dengan *CFB* dengan perbedaannya hanya terletak dari pengisian *shift register*. Pada *OFC*, *ciphertext* yang didapatkan dari hasil enkripsi *shift register* yang akan digunakan untuk mengisi *shift register* kembali, bukan hasil *ciphertext* asli seperti pada CFB. Proses OFC dapat dilihat pada gambar 2.6.



Gambar 2.6. Proses OFC.

2.2.5. Counter

Berbeda dari *mode* lain, *counter mode* menggunakan sebuah angka yang akan ditambah secara *incremental* untuk setiap ronde. Setiap blok dienkripsi menggunakan sebuah angka yang berbeda dengan angka tersebut diinisialisasi dengan sebuah nilai. Pada setiap enkripsi, nilai *counter* ditambah satu. Nilai *counter* akan dienkripsi menggunakan kunci dan hasilnya akan di-*xor* dengan *plaintext* untuk menghasilkan *ciphertext*. *Counter mode* tidak melakukan *block chaining* seperti ada *CBC*, *CFB*, dan *OFC* sehingga blok-bloknya bersifat independen. Proses *counter mode* dapat dilihat pada gambar 2.7.



Gambar 2.7. Proses *counter mode*.

2.3. Jaringan Feistel

Jaringan *feistel* merupakan struktur jaringan simetris yang umumnya digunakan dalam *block cipher*. Jaringan *feistel* diciptakan oleh Horst Feistel dan pertama kali diperkenalkan dalam algoritma *block cipher* Lucifer pada tahun 1973[5]. Jaringan *feistel* bersifat *invertible* untuk proses enkripsi dan dekripsi yang berarti tidak perlu algoritma baru untuk enkripsi dan dekripsi. Sebuah jaringan *feistel* menggunakan *round function*, sebuah fungsi yang menerima dua *input* berupa blok data dan *subkey*, dan mengembalikan blok *output* dengan ukuran yang sama. Pada setiap ronde, *round function* dilakukan pada setengah data dan hasilnya akan di-*xor* dengan setengah data sisanya. Pada setiap ronde, hasil nilai blok kiri adalah blok kanan dan hasil nilai blok kanan hasil *round function* blok kanan yang di-*xor* dengan blok kiri. Jaringan *feistel* pada enkripsi putaran ke-*i* dapat dirumuskan sebagai berikut.

$$L_i = R_{i-1}$$

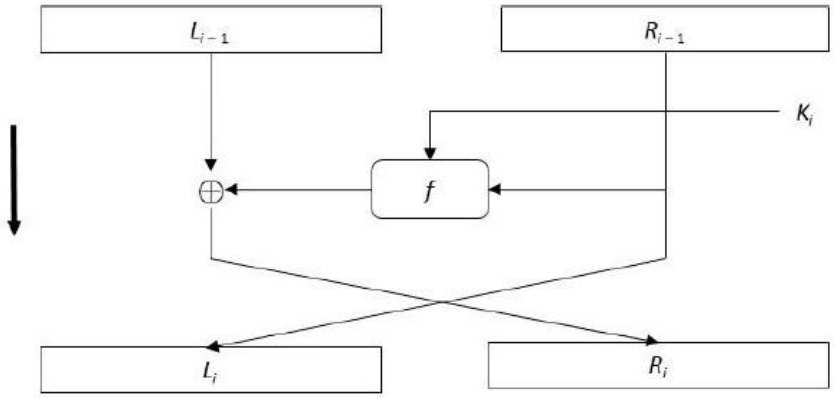
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Khusus untuk ronde terakhir, tidak dilakukan substitusi kiri dan kanan. Rumus untuk ronde terakhir adalah sebagai berikut.

$$L_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

$$R_i = R_{i-1}$$

Round function akan dilakukan sejumlah kali dan hasil akhirnya merupakan data yang telah dienkripsi. *Round function* yang digunakan dapat dibuat serumit mungkin dan tidak akan mempengaruhi sifat *invertible* dari jaringan *feistel*. Struktur jaringan *feistel* dapat dilihat pada gambar 2.8.



Gambar 2.8. Jaringan Feistel.

2.4. Operasi Matematis

Terdapat beberapa operasi matematis yang digunakan dalam sebuah *cipher*, yaitu operasi *xor*, penjumlahan modulo, perkalian modulo, dan *rotation* (*shift left* dan *shift right*). Operasi *xor* merupakan operasi yang menghasilkan ***True*** jika kedua *input* berbeda. Jika bit dari *input* pertama dan *input* kedua sama, *xor* menghasilkan nilai ***False*** sedangkan jika kedua bit berbeda maka dihasilkan nilai ***True***. Karena sifat operasi *xor* tersebut, sulit ditemukan pasangan bit yang benar. Operasi penjumlahan modulo merupakan operasi yang mencari nilai *r* bukan negatif terkecil ketika *a*+*b* dibagi dengan *m*. Operasi penjumlahan modulo dapat dirumuskan sebagai berikut.

$$a +_m b = r, \text{ untuk } 0 \leq r < m$$

Mirip dengan penjumlahan modulo, operasi perkalian modulo juga mencari nilai *r* ketika *a* x *b* dibagi dengan *m*. Operasi perkalian modulo dapat dirumuskan sebagai berikut.

$$a \times_p b = r, \text{ untuk } 0 \leq r \leq m$$

Operasi penjumlahan dan perkalian modulo dilakukan untuk meningkatkan *confusion* dari *cipher*. Operasi *rotation* merupakan operasi yang menggeser bit sebanyak *n* kali ke kiri (*shift left*) atau ke kanan (*shift right*). Dengan melakukan pergeseran, keamanan *cipher* semakin baik dengan menyulitkan proses kriptanalisis.

2.5. Permutasi dan S-Box

Pada *cipher*, permutasi digunakan untuk melakukan transposisi dan S-Box digunakan untuk melakukan substitusi. Pada algoritma DES, dibentuk kotak permutasi yang berisi 56 indeks yang diacak. Blok kiri C merupakan 28 huruf pada 28 indeks pertama dari kotak permutasi sedangkan blok kanan D merupakan 28 huruf pada 28 indeks kedua dari kotak permutasi. Pada DES, permutasi dilakukan untuk mengacak hasil proses substitusi S-Box. Proses substitusi dengan S-Box dilakukan dengan mencari nilai dari tabel S-Box pada baris dan kolom tertentu. Nilai baris dan kolom didapatkan dari blok data dengan mengambil bit-bit tertentu dari blok. Pada DES, nilai baris diambil bit pertama dan bit terakhir sedangkan nilai kolom diambil dari bit kedua hingga bit kelima. Penggunaan permutasi dan S-Box dilakukan untuk mengimplementasi prinsip *confusion* dan *diffusion* untuk memastikan *cipher* aman.

2.6. Prinsip Confusion dan Diffusion

Menurut *Shannon*, terdapat dua prinsip yang diperlukan agar sebuah *cipher* menjadi aman, yaitu prinsip *confusion* dan *diffusion*[6]. Prinsip *confusion* menyebutkan bahwa setiap bit dari *ciphertext* sebaiknya dependen terhadap beberapa bagian dari kunci sehingga mengaburkan hubungan di antara keduanya. Prinsip *confusion* menyembunyikan hubungan dari *ciphertext* dengan kunci agar kunci menjadi sulit dicari dari *ciphertext*. Prinsip *confusion* dapat direalisasikan dengan menggunakan algoritma substitusi yang kompleks. Berbeda dari prinsip *confusion*, prinsip *diffusion* menyembunyikan hubungan antara *ciphertext* dengan *plaintext*. Pada prinsip *diffusion*, pengaruh satu bit *plaintext* disebarkan ke sebanyak mungkin *ciphertext* agar perubahan kecil dapat memberikan pengaruh yang bssar. Prinsip *diffusion* dapat direalisasikan dengan menggunakan transposisi.

3. Trident Cipher

Trident Cipher merupakan algoritma kriptografi yang berbasis bit dengan ukuran blok 64-bit dan ukuran kunci 64-bit. *Trident Cipher* terinspirasi dari algoritma-algoritma kriptografi *modern* sebelumnya, yaitu DEA yang merupakan algoritma untuk standar DES, Rijndael yang merupakan algoritma untuk standar AES, dan MARC. Seperti ketiga jenis algoritma kriptografi yang disebutkan di atas, *Trident Cipher* juga menggunakan jaringan *Feistel* seperti yang dijelaskan sebelumnya. Yang merupakan keunikan dari *Trident Cipher* merupakan fungsi *f* dan algoritma untuk pembangkitan kunci putaran.

3.1. Algoritma Pembangkitan Kunci Putaran

Algoritma pembangkitan kunci putaran dimulai dengan memasukkan kunci 64-bit. Kunci ini merupakan kunci eksternal apabila sedang berada pada putaran pertama dan merupakan hasil gabungan kunci sebelumnya apabila tidak berada pada putaran pertama.

Tahap berikutnya adalah membelah kunci menjadi 2 buah kunci 32-bit. 32-bit pertama merupakan bit 0 - 31 dari kunci dan 32-bit kedua merupakan bit 32 - 63 dari kunci. Pada 32-bit pertama dilakukan penjumlahan modulo 2^{32} terhadap nomor putaran. Pada 32-bit berikutnya dilakukan perkalian modulo dependen $2^{32} + 1$ terhadap putaran. Pada penjumlahan modulo terhadap putaran, nilai key akan bertambah sebanyak nomor putaran (0 - 15) lalu dilakukan modulo dengan 32. Pada perkalian modulo terhadap putaran, nilai key akan dikalikan dengan nomor putaran sebelum lalu dilakukan modulo dengan 32.

Pada tahap berikutnya, dilakukan penambahan bit dari hasil 32-bit pertama hasil penjumlahan modulo dan 32-bit kedua hasil perkalian modulo. 16-bit pertama dari hasil penjumlahan modulo akan ditambahkan ke akhir dari hasil perkalian modulo. Sementara itu, 16-bit pertama dari hasil perkalian modulo akan ditambahkan ke akhir dari hasil penjumlahan modulo. Hal ini menghasilkan dua buah 48-bit kunci.

Tahap berikutnya merupakan tahap substitusi kunci. Terdapat 2 buah kotak-S yang digunakan untuk kedua 48-bit kunci. Kunci pertama diproses dengan kotak-S pertama dan kunci kedua diproses dengan kotak-S kedua. Proses pada kedua kotak-S sama. Proses kotak-S dimulai dengan membagi kunci menjadi 8-bit sehingga dihasilkan 6 x 8-bit. Untuk setiap 8-bit, dilakukan pengecekan pada S-Box. Bit-bit ganjil dihubungkan dan menjadi nomor baris (yang bernilai 0 - 15) dan bit-bit genap digabungkan dan menjadi nomor baris (yang bernilai 0 - 15). Nilai pada kedelapan bit tersebut ditukar dengan nilai 4-bit yang terdapat dalam kotak-S. Hasil yang didapatkan dari pemrosesan kotak-S pada setiap 48-bit kunci adalah 6 x 4-bit, yaitu 24 bit kunci. Adapun kotak -S yang digunakan adalah sebagai berikut.

Tabel 3.1 S-Box 1.

10	5	12	9	14	3	0	8	13	2	15	6	11	1	4	7
2	3	4	13	9	1	6	7	0	15	10	14	12	5	11	8

13	1	8	3	6	10	9	5	14	15	11	0	4	12	7	2
3	14	7	9	13	11	4	5	12	8	1	0	15	6	2	10
14	10	8	15	9	4	13	11	12	1	0	3	2	6	5	7
3	1	6	8	10	15	12	7	13	0	11	2	4	5	9	14
2	8	14	10	11	15	5	4	12	13	3	0	9	7	6	1
10	5	0	1	9	4	6	2	3	15	11	7	12	8	13	14
1	7	14	11	12	3	9	0	6	2	5	13	8	10	4	15
7	15	0	14	1	6	5	9	11	3	12	2	13	10	4	8
7	6	1	14	0	5	2	13	4	15	8	10	12	11	9	3
4	9	7	10	1	5	15	13	12	3	2	14	11	8	0	6
10	8	12	1	2	5	9	15	0	13	3	6	4	14	7	11
6	10	14	9	3	7	0	1	11	8	2	15	5	4	12	13
4	0	6	11	2	9	14	8	12	13	3	15	7	10	1	5
10	11	0	2	9	14	7	4	5	6	3	13	15	8	12	1

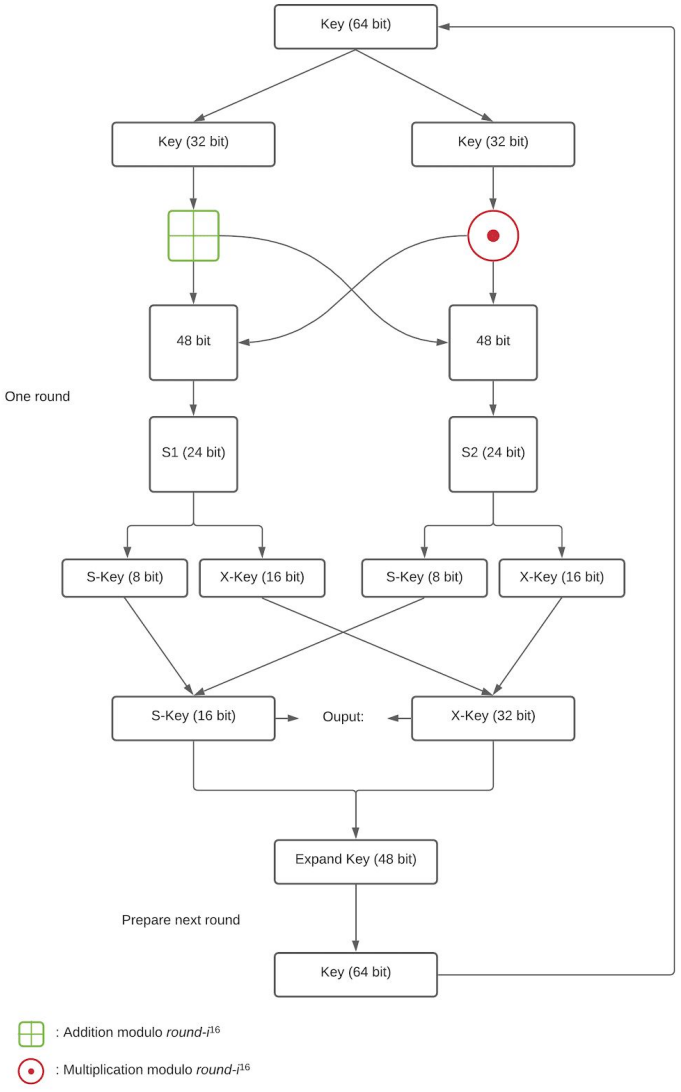
Tabel 3.2 S-Box 2.

0	11	13	8	4	7	3	6	10	5	9	14	2	12	15	1
0	15	13	8	14	9	6	10	1	5	2	12	7	11	4	3
5	14	13	6	7	1	11	15	3	2	4	0	12	9	8	10
0	3	14	1	7	6	13	4	10	8	5	15	11	9	12	2
10	0	5	11	13	6	3	14	4	12	2	9	7	15	1	8
8	0	7	10	11	6	9	13	14	15	2	4	12	1	5	3
4	3	15	13	11	2	10	1	5	12	8	0	7	6	9	14
9	14	10	1	5	8	2	15	12	13	0	4	3	6	7	11
2	15	6	0	14	9	10	11	7	4	1	12	13	3	5	8
7	8	12	9	14	3	5	2	0	15	13	11	10	4	1	6
13	12	15	1	6	8	0	11	14	10	3	2	9	5	7	4
1	7	9	3	4	0	2	12	13	10	5	14	11	15	6	8
11	10	1	9	4	15	3	12	2	0	7	6	13	8	5	14
1	5	13	14	3	0	6	9	11	10	2	15	8	7	12	4
10	12	3	4	9	11	5	8	0	6	2	1	7	14	13	15
1	10	15	2	4	6	11	3	5	8	13	7	0	12	14	9

Setelah melewati tahap tersebut, maka didapatkan 2 buah kunci 24-bit hasil enkripsi. 8-bit pertama pada kedua kunci tersebut akan digabungkan untuk membentuk kunci-S yang berukuran 16-bit

sementara 16-bit berikutnya akan digunakan untuk membentuk kunci-X yang berukuran 32-bit. Kedua kunci tersebut merupakan kunci putaran yang akan dimasukkan ke dalam fungsi *f*.

Tahap terakhir merupakan tahap untuk membentuk 64-bit kunci untuk menjadi masukan pada putaran berikutnya. Pertama digabungkan kunci-S dan kunci-X menghasilkan kunci yang berukuran 48-bit. Kemudian, nilai dari 48-bit tersebut akan dibagi tiap 3-bit sehingga menghasilkan 16 x 3-bit. Kemudian nomor putaran sekarang dilakukan modulus dengan 3, sehingga mendapatkan nilai 0, 1, atau 2. Jika nilai yang didapatkan 0, maka bit pertama yang diambil. Jika nilai 1, maka bit kedua yang diambil. Jika nilai yang didapatkan 2, maka bit ketiga yang diambil. Hasilnya didapatkan 16 bit. Kemudian, 16-bit tersebut ditambahkan ke belakang 48-bit gabungan sebelumnya sehingga didapatkan 64-bit yang merupakan masukan untuk putaran berikutnya.



Gambar 3.1. Algoritma Pembangkitan Kunci Putaran pada *Trident cipher*.

3.2. Algoritma Fungsi f / Round Function

Algoritma fungsi *f* pada *Trident cipher* menerima masukan 32-bit blok masukan, yaitu sisi kanan dari jaringan *feistel*. Selain itu, fungsi *f* juga menerima masukan berupa kunci putaran, yaitu kunci-S (16-bit) dan kunci-X (32-bit).

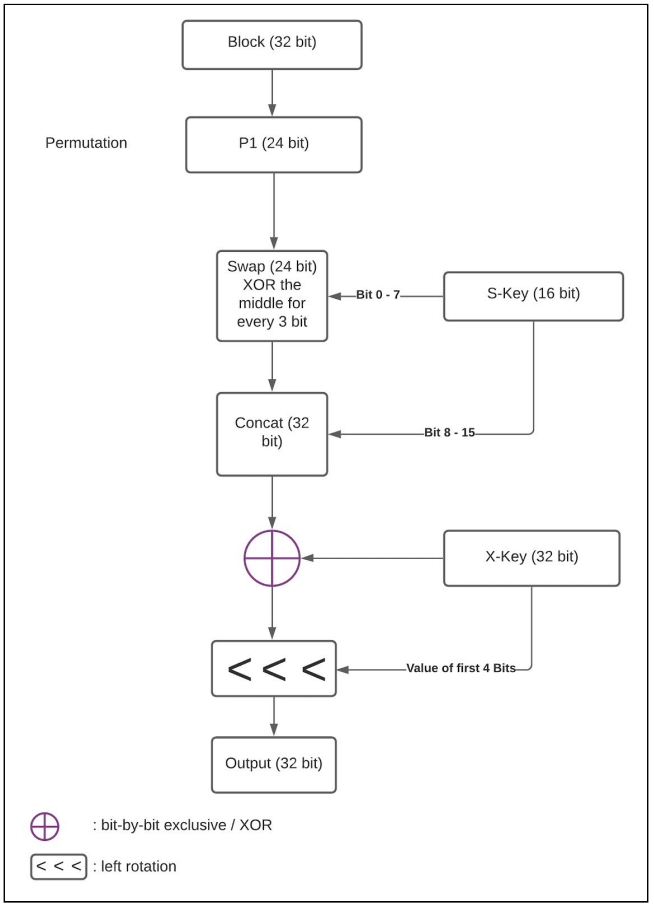
Pertama, dilakukan permutasi pada 32 bit. Permutasi dilakukan dengan memasukkan bit sesuai dengan yang ada dalam kotak permutasi. Dalam kotak permutasi, nilai pada setiap kotak diganti dengan bit yang berada pada urutan tersebut dari masukan 32-bit. Hasil dari setiap kotak dihubungkan menjadi 24-bit teks hasil permutasi. Kotak-P yang digunakan adalah sebagai berikut.

Tabel 3.3 P-Box 2

16	15	13	27	21	25	14	18	2	8	7	19
3	24	28	4	10	5	12	0	6	1	29	31

Tahap berikutnya adalah melakukan pertukaran kunci. Pada tahap ini, diambil 8-bit pertama dari kunci-S dan dilakukan pemisahan 24-bit hasil permutasi ke dalam blok berukuran 3-bit. Kemudian, nilai paling tengah dari 3-bit ini ditukar dengan nilai yang terdapat pada bit kunci-S yang bersangkutan. Hasil yang didapatkan adalah 24-bit yang sudah disubstitusi dengan kunci-S.

Tahap berikutnya adalah melakukan penambahan bit dengan 8-bit sisa dari kunci-S, yaitu bit 8 hingga bit 15. Dengan demikian masukan 24-bit akan diubah menjadi 32-bit dari bit sisa kunci-S. Tahap terakhir adalah melakukan *xor* dengan kunci-X yang menghasilkan 32-bit hasil *xor*. Kemudian, diambil 4-bit pertama dari kunci-X dan didapatkan nilai yang berkisar antara 0 hingga 15. Dilakukan rotasi ke kiri sebanyak jumlah nilai tersebut dan hasil tersebut merupakan hasil dari fungsi *f*.



Gambar 3.2. Fungsi *f* pada *Trident cipher*.

3.3. Eksekusi Trident Cipher

Hal yang dilakukan pertama kali pada *Trident cipher* adalah membagi masukan menjadi 64-bit. Apabila jumlah bit bukan merupakan 64-bit, dilakukan *padding* pada bagian depan dengan bit 0 hingga panjang merupakan kelipatan 64-bit. Lalu didapatkan blok-blok berukuran 64-bit

Kemudian, setiap blok dimasukkan ke dalam algoritma enkripsi yang memiliki 3 mode, yaitu *EBC, CFC, dan Counter*. Ketiga mode enkripsi ini menggunakan algoritma *feistel* seperti yang dijelaskan di bagian 2.3 dengan algoritma pembangkitan kunci dan fungsi *f* sesuai dengan yang dijelaskan pada bagian 3.1 dan 3.2.

3.4. Confusion dan Diffusion

Prinsip *confusion* dan *diffusion* dipenuhi oleh operasi-operasi matematika dan tahap-tahap yang dilakukan saat enkripsi menggunakan *Trident cipher*. Prinsip *confusion* dipenuhi dengan *s-box* dan substitusi dan penambahan *key* pada plainteks. Sementara itu, prinsip *diffusion* dipenuhi dengan permutasi, rotasi, penjumlahan modulo, dan perkalian modulo.

4. Eksperimen dan Analisa Hasil

4.1. Contoh Enkripsi Byte

Tabel 4.1. Hasil uji coba *trident cipher* dalam *mode* CBC.

Kunci: Trident3
Teks Awal: Percobaan untuk mencari tahu hasil enkripsi dari Trident Cipher dan hasil yang didapatkan.
Bit Sebelum Enkripsi 00000000000000000000000000000000000000000000000001010000011001010111001001100011 01101111011000100110000101100000101101110001000000111010101101110011101000111010 1 01101011001000000110110101100101011011100110001101100001011100100110100100100000 01110100011000010110100001110101001000000110100001100001011100110110100101101100 00100000011001010111001101011011100100110100101110000011100110110100100100000 01100100011000010111001001101001001000000101010001110010011010010110010001100101 01101110011101000010000001000011011010010111000001101000011001010111001000100000 01100100011000010110111000100000011010000110000101110011011010010110110000100000 01111001011000010110111001100111001000000110010001101001011001000110000101110000 01100001011101000110101101100001011011110001011
Bit Sesudah Enkripsi 11000101110101010100001100000111110111110101110111001100001010010101010011001100 11000000000011000100100110001101100101010011111000011010000111011001101001011110 11111101100110000100000100100010010010000100100001010111010001100011101101001101 01001010011101111111010011001110010000010001111000011000000001101110001001110101 11100011110111111111000000000011011110101101100111111010001011100001001001000000 10111000011111110101001110001100001010010011100101000001010010000110101101111111 10101101000100110010010101000001000100101000001100100011001001011001011100001010 11011010010000010111001111010011111101000110010001110111100100001001011000011111

1111011001000100101111010111111011010110110010010110111101011000101000100101100001100011011100110101101000000111011010010101000 1

Dari hasil di atas dapat dibuktikan bahwa terdapat perubahan bit yang signifikan antara hasil sebelum enkripsi dan hasil sesudah enkripsi. Selain itu, seperti yang dilihat dari contoh bit sebelum enkripsi, telah dilakukan *null padding* agar total bit sebelum enkripsi berjumlah kelipatan 64.

4.2. Analisis Confusion dan Diffusion

Gambar 4.1. Citra *starry night*.

Dilakukan uji coba dengan citra *starry night* dengan ukuran 220.224 *bytes* yang akan dienkripsi dengan kunci 8 karakter (64 bit) = ‘h4l0dun!’.

Tabel 4.2. Hasil uji coba dengan pergantian satu *byte* kunci.

Kunci	Jumlah Perubahan Byte	Persentase Perubahan Byte
h**a**l0dun!	219344 dari 220224	99,6%
h4l0dun**i**	220224 dari 220224	100%
h4l**o**dun!	219803 dari 220224	99,8%
H4l0dun!	219580 dari 220224	99,7%
h4**L**0dun!	220224 dari 220224	100%
Rata-rata	219835 dari 220224	99,8%

Dari hasil tabel di atas, dapat dibuktikan bahwa perubahan satu *byte* kunci dapat menghasilkan persentase perubahan *byte* yang signifikan dari hasil enkripsi, yaitu dengan rata-rata sebesar 99,8%. Hal ini memenuhi prinsip *confusion* karena perubahan kunci yang sedikit menghasilkan banyak perubahan *byte* pada hasil enkripsi. Hal ini menyebabkan proses di balik layar sulit deketahui oleh seorang kriptoanalis karena kunci yang berbeda sedikit akan menghasilkan hasil enkripsi yang sangat berbeda.

Tabel 4.3. Hasil uji coba dengan pergantian satu *byte plaintext*.

Posisi Pergantian *Byte*	Jumlah Perubahan Byte	Persentase Perubahan *Ciphertext*
0	220224 dari 220224	100%
2499	211457 dari 220224	96.01%
4999	202707 dari 220224	92.04%
9999	185207 dari 220224	84.09%
14999	167707 dari 220224	76.15%
19999	150207 dari 220224	68.20%
24999	132707 dari 220224	60.26%

Berdasarkan eksperimen perubahan satu *byte plaintext* pada posisi *byte* yang berbeda dapat terlihat bahwa perubahan satu *byte* dapat menyebabkan perubahan *ciphertext* yang besar tergantung dari posisi perubahan. Hal ini dikarenakan prinsip *block chaining* yang menggunakan hasil blok *ciphertext* sebelumnya untuk menghasilkan blok *ciphertext* baru. Semakin awal perubahan *byte*, semakin banyak blok-blok setelahnya yang akan terpengaruh. Sebaliknya, semakin akhir perubahan *byte*, semakin sedikit blok-blok setelahnya yang akan terpengaruh. Karena dari eksperimen perubahan satu *byte* awal menyebabkan perubahan *ciphertext* yang sangat besar maka *trident cipher* memenuhi prinsip *diffusion*.

4.3. Analisis Kecepatan

Tabel 4.4. Hasil uji coba kecepatan *trident cipher* dengan berbagai ukuran pesan dan *mode*.

Ukuran pesan (KB)	Mode	Operasi	Kecepatan (detik)
8	ECB	Enkripsi	1.071
		Dekripsi	1.071
	CBC	Enkripsi	1.072
		Dekripsi	1.072
	Counter	Enkripsi	1.108
		Dekripsi	1.108
16	ECB	Enkripsi	2.166
		Dekripsi	2.166

	CBC	Enkripsi	2.218
		Dekripsi	2.218
	Counter	Enkripsi	2.325
		Dekripsi	2.325
32	ECB	Enkripsi	4.364
		Dekripsi	4.364
	CBC	Enkripsi	4.436
		Dekripsi	4.436
	Counter	Enkripsi	4.475
		Dekripsi	4.475
64	ECB	Enkripsi	8.809
		Dekripsi	8.809
	CBC	Enkripsi	9.035
		Dekripsi	9.035
	Counter	Enkripsi	9.211
		Dekripsi	9.211
113	ECB	Enkripsi	15.377
		Dekripsi	15.377
	CBC	Enkripsi	16.511
		Dekripsi	16.511
	Counter	Enkripsi	17.303
		Dekripsi	17.303
232	ECB	Enkripsi	32.368
		Dekripsi	32.368
	CBC	Enkripsi	37.558
		Dekripsi	37.558
	Counter	Enkripsi	41.408
		Dekripsi	41.408

Gambar 4.2. Grafis hubungan ukuran pesan dan waktu enkripsi dengan *trident cipher*.

Berdasarkan hasil eksperimen kecepatan yang terlihat pada tabel 4.4 dan gambar 4.2, kecepatan *trident cipher* dalam melakukan enkripsi dan dekripsi meningkat secara *linear* dengan meningkatnya ukuran pesan. Seperti terlihat pada gambar 4.2, *mode* ECB memiliki gradien yang lebih rendah, diikuti oleh CBC, dan *counter*. Perbedaan gradien disebabkan oleh perbedaan jumlah operasi yang harus dilakukan oleh setiap *mode*. Pada ECB, *plaintext* langsung dimasukkan ke jaringan *feistel* untuk dienkripsi. Pada CBC, perlu dilakukan *xor* dengan blok *ciphertext* sebelumnya. Pada *counter*, perlu dilakukan perubahan angka menjadi *binary* untuk setiap ronde. Dengan bertambahnya operasi, waktu yang diperlukan menjadi semakin lama sehingga menyebabkan perbedaan gradien dari ketiga *mode*.

5. Kesimpulan

5.1. Kesimpulan

Trident cipher merupakan algoritma kriptografi yang menghasilkan hasil yang memuaskan pada proses kriptografi walaupun dengan algoritma yang relatif sederhhana. *Trident Cipher* memenuhi prinsip *confusion* dan *diffusion* dari jumlah perubahan blok hasil enkripsi apabila 1-*byte* kunci atau plainteks diubah. *Trident cipher* juga cukup *scalable* dengan peningkatan waktu yang *linear* dengan bertambahnya ukuran pesan.

5.2. Saran

Saran yang dapat kami berikan diharapkan mampu meningkatkan kecepatan enkripsi dari *Trident cipher*. Pertama adalah dengan melakukan paralelisasi enkripsi untuk mode *ECB* dan *Counter* untuk mempercepat proses enkripsi. Selain itu, dapat juga dilakukan operasi langsung dalam level bit dikarenakan pada saat ini *Trident cipher* menggunakan *string* yang merupakan representasi dari bit yang akan dioperasikan sehingga menghasilkan *overhead* yang tinggi dalam hal *casting* antara tipe data *string* dan *integer*.

6. Referensi

[1] Smid, M. E., & Branstad, D. K. (1988). The Data Encryption Standard: Past and Future. Proceedings of the IEEE, 76(5), 550–559. https://doi.org/10.1109/5.4441.

[2] Daemen, J., & Rijmen, V. (1999). AES proposal: Rijndael.

[3] Hsiao, J. S. (2012). MARC – A New Block Cipher Algorithm. Contemporary Engineering Sciences, 5(6), 281–286.

[4] Thakur, J., & Kumar, N. (2011). DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International Journal of Emerging Technology and Advanced Engineering*, *1*(2), 6–12.

[5] Feistel, H. (1973). Cryptography and Computer Privacy. *Scientific American*. https://doi.org/10.1038/scientificamerican0573-15

[6] Shannon, C. E. (1998). Communication theory of secrecy systems. 1945. *M.D. Computing : Computers in Medical Practice*, *15*(1), 57–64.

Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada semua orang yang telah membantu dalam proses desain dan implementasi dari *trident cipher*. Penulis mengucapkan terima kasih khususnya kepada Dr. Ir. Rinaldi Munir, MT. selaku dosen pengajar, orang tua penulis dan teman-teman penulis yang telah memberikan dukungan dan saran.