RANCANGAN,IMPLEMENTASI DAN PENGUJIAN *ZENARC SUPER CIPHER* SEBAGAI IMPLEMENTASI ALGORITMA KUNCI SIMETRI

Ozzi Oriza Sardjito – NIM 13503050

Program Studi Teknik Informatika, STEI Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
E-mail : if13050@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang rancangan, implementasi serta pengujian *Zenarc Super Cipher* untuk menyandikan pesan dan data yang dimasukkan agar kerahasiaannya tetap terjaga. *Zenarc Super Cipher* merupakan sebuah algoritma kriptografi kunci simetri yang dirancang secara independen memanfaatkan penggabungan beberapa algoritma *cipher* klasik seperti substitusi abjad majemuk, metoda ekspansi, metoda permutasi sederhana, dan teknik pengolahan kunci yang saya implementasikan sendiri. *Cipher* ini juga menggunakan metode *block cipher* yang membagi-bagi pesan menjadi blok yang panjangnya sama sesuai dengan kunci yang dimasukkan, dan menggunakan mode *Electronic Code Book* (ECB) yang memproses pesan pada blok yang sama tanpa ada referensi ke blok sebelum dan setelahnya. *Cipher* ini bekerja di tingkat *character* dan *byte* (bukan bit), sehingga sangat efisien untuk digunakan dalam menyandikan pesan yang ingin dikirim secara rahasia. Tetapi, cipher ini dapat juga digunakan untuk mengenkripsikan tipe *file* lainnya seperti gambar, suara, dan masih banyak lagi.

Untuk mengimplementasikan penggunaan *cipher* ini, dibangun suatu perangkat lunak sederhana yang bias mengenkripsikan dan mendekripsikan suatu file dengan *Zenarc Super Cipher*, bernama *ZenarcEnc*. Perangkat lunak ini dibangun menggunakan *tool* pengembangan *Microsoft Visual Studio .Net 2005* dalam lingkungan sistem operasi *Windows XP*.

Program ini kemudian digunakan untuk menguji keamanan *cipher* ini dan mencoba penggunaannya untuk berbagai tipe arsip, dokumen dan pesan. Karena menggunakan teknik ekspansi, *ciphertext* yang dihasilkan ukurannya lebih besar dari *plaintext* awalnya. Selain itu, *cipher* ini menggunakan dua kunci yang harus dimasukkan pada saat enkripsi dan dekripsi. Salah satu merupakan kunci utama, satu lagi merupakan nilai *override* yang mempersulit tingkat enkripsi pesan tersebut. Dengan implementasi algoritma sederhana, *Zenarc Super Cipher* ini dapat secara efisien digunakan di mana saja dengan ukuran yang kecil, yang menjadi salah satu kelebihannya. Tentu saja, *cipher* ini juga memiliki berbagai kelebihan dan kelemahan lainnya.

Kata Kunci : *Zenarc Super Cipher, ZenarcEnc, override,* metode ekspansi, enkripsi, dekripsi, substitusi abjad majemuk, *block cipher, Electronic Code Book.*

1. Pendahuluan

Sejak berabad-abad yang lalu, pengiriman dan penyandian pesan merupakan suatu hal yang sangat penting bagi kehidupan manusia. Apakah itu dalam keadaan perang ataupun damai. Manusia tetap memrlukan berbagai cara untuk berkomunikasi secara rahasia dan aman. Sampai sekarang, di zaman elektronik dan informasi ini, cara pengiriman pesan yang benar-benar aman tetap dicari, tetapi sekarang kebutuhan semakin bertambah karena data pun harus disimpan dengan aman, sehingga dibutuhkan cara pengamanan data dan penyandian pesan yang benar-benar akurat.

Untuk memenuhi hal tersebut, teknik kriptografi dikembangkan secara ekstensif. Teknik kriptografi ini meliputi metode enkripsi dan dekripsi terhadap pesan yang akan dikirim dan data yang akan diamankan. Enkripsi dilakukan sebelum data disimpan atau pesan dikirim, sehingga pesan teracak menjadi sebuah kode aneh yang tak bisa dibaca dengan cara biasa. Data pun tersimpan menjadi semacam *junk data* yang tak berharga bila dilihat secara biasa, dan tak dapat dimanfaatkan lagi sampai didekripsi kembali menggunakan cara yang sama. Data dan pesan tersebut hanya dapat dibaca kembali menggunakan kunci rahasia yang benar-benar rahasia dan hanya orang-orang tertentu yang dituju yang bisa mengetahuinya.

Seiring berjalannya waktu, berbagai jenis algoritma kriptografi bermunculan. Perkembangan teknologi informasi dan kecepatan perangkat keras pun melaju melebihi apa yang sempat diperkirakan di masa lalu. Algoritma yang dahulu sangat sulit dipecahkan, sekarang menjadi mudah dipecahkan, karena keberadaan internet yang membuat sistem komputer terdistribusi sangat mudah diimplementasikan, serta kecepatan pemrosesan data sekarang ini. Salah satu contoh algoritma yang tumbang di tangan perkembangan zaman ini adalah DES, *Data Encryption Standard*. Sudah ada berbagai perangkat keras dan perangkat lunak yang dapat memecahkan kunci DES dengan waktu yang relatif cepat.

Karena itulah, dikembangkan *Zenarc Super Cipher*, suatu algoritma *cipher* baru yang memanfaatkan kombinasi penggunaan *cipher block*, algoritma substitusi abjad majemuk dan teknik pengolahan kunci khusus yang membuatnya unik dan lebih sulit diserang secara biasa. Algoritma ini bekerja pada mode *character*, sehingga data diproses *character per character* dalam blok yang sudah ditentukan panjangnya berdasarkan kunci yang dimasukkan. Algoritma ini juga memanfaatkan tabel ASCII sehingga setiap *character* yang ada diubah dengan memanfaatkan nilainya di tabel ASCII.

Kekuatan algoritma ini bergantung pada panjang kunci yang dimasukkan, serta lebih diperkuat lagi dengan penggunaan *magic number*, kunci khusus yang dimasukkan bersama dengan kunci utama yang bertujuan untuk meng-override salah satu nilai kunci tertentu sehingga mempersulit tingkat enkripsi data atu pesan tersebut. Makin panjang kuncinya, panjang blok makin panjang, dan penyerangan makin sulit dilakukan. Dengan metode ekspansi, *plaintext* semakin sulit diketahui karena panjang pesan lebih kecil dari *ciphertext* yang dihasilkan. Implementasi *Zenarc Super Cipher* pada program *ZenarcEnc* ini dilakukan dengan modifikasi mode ECB pada *block cipher* dengan level implementasi per karakter dan byte(bukan level bit), tetapi tidak tertutup kemungkinan implementasi dengan mode-mode lain dapat dilakukan.

2. Dasar Teori yang Digunakan

Algoritma *Zenarc Super Cipher* ini memiliki landasan pada berbagi teori *cipher* yang sudah ada, seperti *cipher* substitusi abjad-majemuk, *cipher block*, *Electronic Code Book*(ECB), metode ekspansi, dan metode permutasi sederhana.

2.1 *Cipher* Substitusi Abjad-Majemuk

Cipher substitusi abjad-majemuk (*polyalphabetic-substitution cipher*) merupakan *cipher* substitusi-ganda (*multiple-substituion cipher*) yang melibatkan penggunaan kunci yang berbeda untuk tiap *character* pesan atau data. *Cipher* ini ditemukan pertama kali oleh Leon Battista pada tahun 1568, dan digunakan oleh tentara AS selama Perang Sipil Amerika.

Cipher ini kebanyakan merupakan *cipher* substitusi periodik yang didasarkan pada periode m, sehingga kekuatan dan pengulangannya tergantung panjang kunci dan panjang pesan. Misalkan *plaintext* P adalah :

$P = p_1 p_2 \ldots p_m p_{m-1} \ldots p_{2m} \ldots$

Maka *ciphertext* hasil enkripsi adalah :

$E_k(P) = f_1(p_1)\, f_2(p_2) \ldots f_m(p_m) f_{m-1}(p_{m-1}) \ldots f_{2m}(p(_{2m})$

Sebagai contoh, biasanya digunakan *Vigenere Cipher* yang ditemukan oleh kriptolog Prancis, Blaise de Vigenere pada abad ke-16. Misalkan K adalah deretan kunci :

$K = k_1 k_2 \ldots k_m$

Dimana k_i untuk $1 \leq i \leq m$ menyatakan jumlah pergeseran pada huruf ke-i. Maka, karakter *ciphertext* $y_i(p)$ adalah :

$y_i(p) = (p + k_i) \bmod n$

Misalkan periode m = 23, maka 23 karakter pertama dienkripsi dengan cara tersebut, dimana setiap karakter ke-I menggunakan kunci k_i, tetapi untuk 23 karakter berikutnya, kembali menggunakan pola enkripsi yang sama.

Misalkan enkripsi pada suatu pesan dengan kunci TWILIGHT yang panjangnya m = 8 :
P : NOONECOULDSHAREOURFATE
K: TWILIGHTTWILIGHTTWILIGHT
C: HL…

Maka huruf N pertama dienkripsi dengan kunci T sebagai :

(N+T) mod 26 = (14+20) mod 26 = 8 = H

Dan huruf O berikutnya dienkripsi dengan kunci W sebagai :

(O+W) mod 26 = (15+23) mod 26 = 12 = L

Cipher-cipher yang lain biasanya

diimplementasikan menggunakan pengembangan dari substitusi abjad-majemuk ini, tetapi dengan pembangkitan kunci yang berbeda. *Cipher block* termasuk salah satu di antaranya, tetapi fungsi yang digunakan jauh lebih rumit.

2.2 *Block Cipher*

Pada *cipher block*, rangkaian *plaintext* dibagi menjadi blok-blok bit dengan panjang sama, biasanya 64 bit atau lebih. Enkripsi dilakukan terhadap blok bit *plaintext* menggunakan bit-bit kunci yang ukurannya sama dengan blok bit *plaintext*. Dekripsi dan enkripsi dilakukan dengan cara yang sama, sehingga masih termasuk algoritma kunci simetri. Misalkan blok *plaintext* P yang berukuran m bit dinyatakan sebagai vektor :

$P = (p_1, p_2, \ldots, p_m)$

Yang dalam hal ini, p_i adalah 0 atau 1 untuk i = 1,2,..,m, dan blok *ciphertext* adalah :

$C = (c_1, c_2, \ldots, c_m)$

Yang dalam hal ini c_i adalah 0 atau 1 untuk i = 1,2,..,m.

Bila plainteks dibagi menjadi *n* buah blok, barisan blok-blok plainteks dinyatakan sebagai

$$(P_1, P_2, \ldots, P_n)$$

Untuk setiap blok plainteks P_i, bit-bit penyusunnya dapat dinyatakan sebagai vektor

$$P_i = (p_{i1}, p_{i2}, ..., p_{im})$$

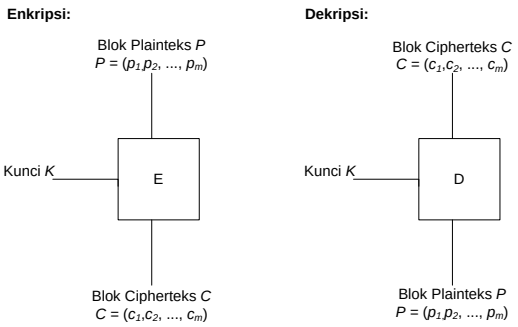
Enkripsi dengan kunci *K* dinyatakan dengan persamaan

$$E_k(P) = C,$$

sedangkan dekripsi dengan kunci *K* dinyatakan dengan persamaan

$$D_k(C) = P$$

Skema enkripsi dan dekripsi dengan *cipher* blok dapat dilihat pada Gambar 1.



Gambar 1 Skema Enkripsi dan dekripsi pada cipher block

Pada algoritma *Zenarc Super Cipher* ini, digunakan *cipher block* yang bekerja pada mode byte dan karakter, bukan mode bit.

2.3 *Electronic Code Book(ECB)*

Kata *code book* berasal dari fakta bahwa blok *plaintext* yang sama akan dienkripsi menjadi *ciphertext* yang sama.sehingga secara teoritis dapat dimungkinkan pembuatan buku kode *plaintext* dan *ciphertext* yang berkoresponden.

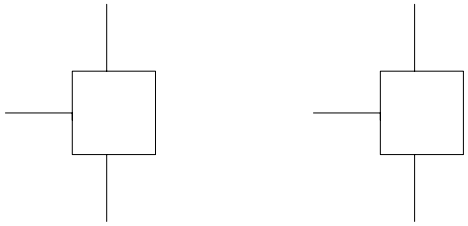
Pada mode ini, setiap blok plainteks P_i dienkripsi secara individual dan independen menjadi blok cipherteks C_i. Secara matematis, enkripsi dengan mode *ECB* dinyatakan sebagai

$$C_i = E_k(P_i)$$

dan dekripsi sebagai

$$P_i = D_k(C_i)$$

yang dalam hal ini, P_i dan C_i masing-masing blok plainteks dan cipherteks ke-*i*. Skema enkripsi dan dekripsi dengan mode *ECB* dapat dilihat pada Gambar 2.



Gambar 2 Skema enkripsi dan dekripsi menggunakan mode ECB

Ada kemungkinan panjang *plaintext* tidak habis dibagi dengan panjang ukuran blok yang ditetapkan, sehingga blok terakhir berukuran lebih pendek dari blok sebelumnya. Salah satu cara untuk mengatasi ini adalah dengan *padding* yaitu menambahkan blok terakhir dengan pola

bit teratur yang sama panjangnya agar ukurannya sama dengan panjang blok yang ditetapkan.

Di *Zenarc Super Cipher*, diterapkan ECB karena pesan dibagi-bagi menjadi blok yang panjangnya sama dan dienkripsi per blok secara independen dan terpisah antar blok yang berbeda satu sama lain.

2.4 Metode ekspansi

Metode ini merupakan salah satu metode kriptografi klasik yang mengekspansi panjang pesan atau data yang diinginkan menjadi lebih besar dari asalnya, sehingga *ciphertext* panjangnya lebih besar dari *plaintext.*

Di *Zenarc Super Cipher*, digunakan metode ini sedemikian rupa sehingga satu byte atau satu karakter pada pesan atau data akan menjadi dua byte atau dua karakter yang berbeda.

2.5 Metode permutasi sederhana

Teknik ini memindahkan posisi bit pada *plaintext* berdasar aturan tertentu. Secara matematis, ditulis sebagai :

$C = PM$

Dimana C adalah blok *ciphertext*, P adalah blok *plaintext*, dan M adalah matriks permutasi.

3. *Zenarc Super Cipher* (ZSC)

3.1 Pengolahan Kunci

ZSC menggunakan substitusi berulang sebagai inti algoritmanya, tetapi kekuatan ZSC ini terletak pada sistem pengolahan kuncinya yang berubah-ubah sesuai nilai *override* yang dimasukkan.

ZSC menggunakan dua macam kunci sebagai input. Kunci pertama adalah kunci yang digunakan untuk mengenkripsikan *plaintext* secara umum, dengan fungsi tertentu. Kunci kedua adalah *magic number*, sebuah nilai *override* yang digunakan untuk memodifikasi nilai kunci yang dimasukkan dan juga digunakan sebagai nilai *padding* di akhir file.

Pada awalnya, diterima dua nilai kunci, K_1 dan O, dalam bentuk integer. K_1 bisa dimasukkan dalam bentuk alfanumerik(*case sensitive*), tetapi pada prosesnya diubah menjadi byte yang sesuai dengan nilai ASCII-nya. Sebagai catatan, angka tetap menjadi angka tersebut Misalkan saja, K_1 yang dimasukkan adalah

“Twilight21“

Sesuai tabel ASCII, huruf T diubah menjadi 84, angka 2 tetap menjadi 2(memakai pembacaan string), sehingga didapat :

“84119105108105103104411621“

dan nilai itu yang dimasukkan sebagai nilai kunci K_1 yang sebenarnya. Sebagai catatan, panjang kunci minimal 16 integer agar efektif. Karena dengan panjang kunci awal yang pendek dapat dihasilkan kunci berbentuk integer yang panjang, maka sebaiknya digunakan kunci awal alfanumerik atau kunci numerik panjang.

Selanjutnya, nilai kunci *override*, O, berada pada domain $1 \leq O \leq 9$, dan dimasukkan sebagai integer. Nilai ini digunakan untuk meng-override nilai 0 yang ada di K_1, sehingga didapat kunci K_2 yang nilai 0-nya sudah diganti dengan nilai O. Misalkan dimasukkan

$O = 3$

Maka nilai K_2 dari K_1 yang sebelumnya adalah

“84119135138135133134116213”

Nilai K_2 ini kemudian yang digunakan sebagai kunci utama, sedangkan nilai K_1 tak diperlukan lagi.

Selanjutnya, dibangkitkan kunci bayangan dari K_2, dengan fungsi

$K_b(i) = (10 – K_2(i))$

Dengan i adalah posisi integer di kunci ke-i.

Kunci bayangan ini merupakan kunci yang digunakan untuk membangkitkan nilai karakter/huruf kedua dari *ciphertext* yang akan dihasilkan.

Jadi, apabila digunakan K_2 seperti di contoh sebelumnya, akan dihasilkan K_b :

“269919759729759779769948 9”

Untuk tahap pengolahan kunci terakhir, nilai ‘9’ di kunci tersebut diganti menjadi nilai O. Alasan penggantian ini karena nilai 9 akan banyak sekali muncul di kunci bila digunakan *lowercase* sebagai kuncinya, sehingga lebih mudah ditebak.

Sehingga, dihasilkan K_m adalah :

“263313753723753773763348 3”

Dan K_m ini adalah kunci bayangan yang digunakan dan dimasukkan ke dalam algoritma utama, beserta K_2 dan O.

Seperti terlihat, dihasilkan 3 kunci yang masing-masing memiliki panjang yang lumayan besar, kecuali nilai *Override* dengan panjang satu integer saja. Panjang kunci ini(25 byte untuk kasus ini) menjadi panjang blok *character* atau *byte* yang digunakan selanjutnya.

3.2 Algoritma Enkripsi Utama

Dengan masukan K_2, K_m, O dan *plaintext* algoritma enkripsi ZSC dapat langsung mengenkripsikan *plaintext* tersebut. Untuk file teks, digunakan mode *character*, sehingga pembagian blok nya per *character*, sedangkan untuk binary file seperti image, suara, *word document* dan sebagainya, digunakan mode *byte*, sehingga pembagian blok nya per *byte* dan cara enkripsinya sedikit berbeda

3.2.1 Mode *Character*

Mode *character* ini digunakan untuk *file-file* teks sederhana yang mengandung pesan yang ingin dienkripsikan. Pada mode ini, pesan diambil dan dibagi dalam blok-blok karakter (1 char = 2 byte) dan dienkripsi per blok.

Setelah membagi pesan dalam blok-blok, blok terakhir, bila panjangnya tidak mencukupi, di-padding dengan nilai O. Selanjutnya, buat satu blok setelah blok terakhir berisi nilai O, tetapi tiga *character* terakhir dari blok tersebut diisi dengan jumlah *character* padding di blok terakhir. Misalnya saja, pesannya adalah :

P = “I will go to somewhere around here.”

Pesan ini dibagi menjadi dua blok dengan panjang 25 karakter :

1 : I will go to somewhere a
2 : round here.

Kita lihat bahwa blok 2 panjangnya bukan 25 karakter, maka kita tambahkan karakter O di sana :

2 : round here.33333333333333

Kita tambahkan 14 karakter O di sana, maka di akhir kita tambahkan blok ketiga :

3: 33333333333333333333333014

Dengan 014 di akhirnya sebagai jumlah karakter padding di sana.

Bila panjang blok sudah sesuai, tambahkan satu blok terakhir di akhir pesan dengan aturan yang sama : nilai O dan 3 karakter akhir 000 sebagai jumlah karakter padding yang ada(0).

Selanjutnya, proses enkripsi awal dimulai. Setiap karakter dienkripsikan dengan kunci di posisi yang sama dari P yang sudah ditambah padding menjadi P_x, dengan fungsi

$C_1(i) = chrw(asc(P_x(i)) + K_2(i))$

dimana i adalah 1,3,5,7,…(bilangan ganjil), dan

$C_1(j) = chrw(asc(P_x(j)) – K_m(j))$

Dimana j adalah 2,4,6,8,…(bilangan genap), dan i adalah posisi karakter ke-i. Fungsi asc berfungsi untuk mengubah suatu karakter menjadi nilai indeks ASCII-nya, sedang chrw berfungsi sebaliknya, mengubah nilai indeks ASCII menjadi karakter.

Seperti terlihat di sini, satu karakter di *plaintext* dienkripsi menjadi dua karakter yang berbeda. Karakter pertama dihasilkan dari pertambahan karakter *plaintext* dengan nilai K_2, sedangkan karakter kedua dihasilkan dari pengurangan karakter *plaintext* dengan nilai K_m.

Selanjutnya, dihasilkan C_1, sebagai *ciphertext* sementara.

Setelah ini, C_1 dipermutasikan secara sederhana dengan lebar kolom O, sehingga didapatkan C_2. Misalkan O = 3, maka karakter ke-4 di C_1 adalah karakter ke-2 di C_2, karakter ke-7 adalah karakter ke-5, dan demikian seterusnya.

Terakhir, di C_2, karakter ke-(O-1, bila O =1 atau 2 maka karakter ke-2) dan kelipatannya di blok tersebut dienkripsikan sekali lagi dengan fungsi :

$C_3(i) = chrw(asc(C_2(i)) + O)$

Dimana

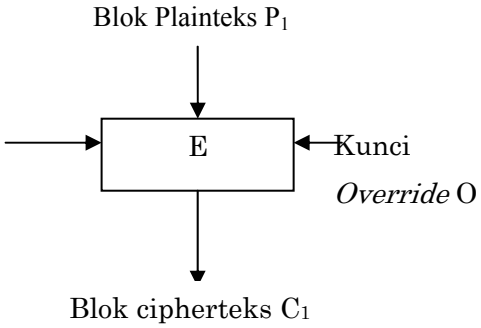
i = (O-1), 2(O-1),…

atau

I = 2,4,6,… untuk O = 1 atau 2

Sehingga dihasilkan C_3 sebagai *ciphertext* akhir dari ZSC ini.

Panjang C_3 yang dihasilkan adalah dua kali panjang P_x, dan panjang P_x adalah



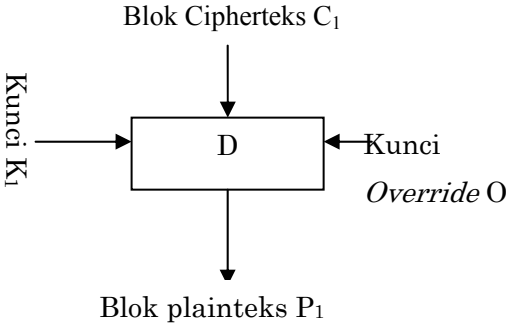
$P_x = P + m + pd$

m = panjang kunci
pd = panjang padding

Sehingga, panjang C_3 secara keseluruhan adalah :

$C_3 = 2 (P + m + pd)$ karakter.

Berikut adalah skema enkripsi dan dekripsi menggunakan ZSC.



Gambar 3 Skema enkripsi dan dekripsi menggunakan ZSC

3.2.2 Mode *Byte*

Mode *byte* ini digunakan untuk data-data dan pesan dalam bentuk non-teks, seperti suara, gambar dan *word document*. Pada mode ini, pesan diambil dan dibagi-bagi dalam blok-blok *byte* dan dienkripsi per blok tersebut secara independen.

Setelah membagi pesan dalam blok-blok, blok terakhir, bila panjangnya tidak mencukupi, di-padding dengan nilai O. Selanjutnya, buat satu blok setelah blok terakhir berisi nilai O, tetapi tiga *byte* terakhir dari blok tersebut diisi dengan jumlah *byte* padding di blok terakhir. Misalnya saja, pesannya adalah :

P = 444400257532

Dan panjang kunci, m adalah 8, maka dibagi dua blok :

1 : 44440025
2: 7532
Kita tambahkan 4 karakter O(misal O=3) di akhir blk kedua karena jumlahnya tidak sesuai, menjadi :

2 : 75323333

Dan tambahkan satu blok padding terakhir, blok ketiga, dengan 004 sebagai akhirnya :

3 : 33333004

Bila panjang blok sudah sesuai, tambahkan satu blok terakhir di akhir pesan dengan aturan yang sama : nilai O dan 3 karakter akhir 000 sebagai jumlah karakter padding yang ada(0).

Selanjutnya, proses enkripsi awal dimulai. Setiap byte dienkripsikan dengan kunci di posisi yang sama dari P yang sudah ditambah padding menjadi P_x, dengan fungsi

$C_1(i) = (P_x(i) + K_2(i)) \bmod 10$

dimana i adalah 1,3,5,7,…(bilangan ganjil), dan

$C_1(j) = (P_x(j)) + K_m(j)) \bmod 10$

Dimana j adalah 2,4,6,8,…(bilangan genap), dan i adalah posisi byte ke-i.

Seperti terlihat di sini, satu byte di *plaintext* dienkripsi menjadi dua byte yang berbeda. byte pertama dihasilkan dari hasil mod 10 dari pertambahan byte *plaintext* dengan nilai K_2, sedangkan byte kedua dihasilkan dari hasil mod 10 dari pertambahan byte *plaintext* dengan nilai K_m.

Selanjutnya, dihasilkan C_1, sebagai *ciphertext* sementara.

Setelah ini, C_1 dipermutasikan secara sederhana dengan lebar kolom O, sehingga didapatkan C_2. Misalkan O = 3, maka karakter ke-4 di C_1 adalah karakter ke-2 di C_2, karakter ke-7 adalah karakter ke-5, dan demikian seterusnya.

Terakhir, di C_2, byte ke-(O-1, apabila O=1 atau 2 maka byte ke 2) dan kelipatannya di blok tersebut dienkripsikan sekali lagi dengan fungsi :

$C_3(i) = (C_2(i) + O) \bmod 10$

Dimana

i = (O-1), 2(O-1),…

atau

i = 2,4,6,.. untuk O=1 atau 2

Sehingga dihasilkan C_3 sebagai *ciphertext* akhir dari ZSC ini.

Panjang C_3 yang dihasilkan adalah dua kali panjang P_x, dan panjang P_x adalah

$P_x = P + m + pd$

m = panjang kunci
pd = panjang padding

Sehingga, panjang C_3 secara keseluruhan adalah :

$C_3 = 2 (P + m + pd)$ byte.

3.3 Cara dekripsi

Seperti algoritma kunci simetri lainnya, ZSC ini dapat didekripsikan dengan menggunakan kunci yang sama, yaitu K_1 dan O. Apabila satu saja dari kunci ini salah, maka hasil dekripsinya akan kacau karena kedua kunci ini sangat krusial dalam pendekripsian ZSC ini.

Seperti biasa, pada awalnya, dilakukan pengolahan kunci oleh sistem sehingga didapat K_2, K_m dan nilai O. Kemudian, kita harus memilih apakah mau mendekripsi dengan mode *byte* atau mode *character*, karena kedua mode itu memiliki perbedaan di dalamnya. Kemudian, *ciphertext* didekripsikan awalnya dengan fungsi kebalikan yang pertama, yaitu

$C_2(i) = chrw(asc(C_3(i)) - O)$ (mode karakter)

$C_2(i) = (C_3(i) - O) \bmod 10$ (mode byte)

Dengan

i = (O-1), 2(O-1),…

atau

i = 2,4,6,.. untuk O=1 atau 2

Setelah itu, C_2 dipermutasikan balik dengan panjang kolom = O, di transpos balik, sehingga didapat C_1.

Terakhir, lakukan proses dekripsi akhir dengan invers fungsi sebelumnya, yaitu :

<<Mode karakter>>

$P_x(j) = chrw(asc(C_1(i)) - K_2(i))$

dimana i adalah 1,3,5,7,…(bilangan ganjil), dan j adalah 1,2,3,4,…(bilangan asli). I dan j adalah posisi karakter. Karakter dummy yang dibuat untuk memusingkan tidak dipakai lagi. Fungsi asc berfungsi untuk mengubah suatu karakter menjadi nilai indeks ASCII-nya, sedang chrw berfungsi sebaliknya, mengubah nilai indeks ASCII menjadi karakter.

<<Mode byte>>

$Px(j) = (C_1(i) - K_2(i)) \bmod 10$

dimana i adalah 1,3,5,7,…(bilangan ganjil), dan j adalah 1,2,3,4,…(bilangan asli). I dan j adalah posisi byte. Byte dummy yang dibuat untuk memusingkan tak dipakai lagi.

Terakhir, kita lihat nilai tiga karakter terakhir, setelah itu kita hilangkan m+x karakter terakhir dari P_x, dimana m adalah panjang kunci dan x adalah nilai 3 karakter terakhir tersebut dalam integer. Itu adalah karakter padding yang tak kita perlukan lagi di sini, sehingga bisa langsung dibuang dan kita dapatkan P, *plaintext* asal.

4. Pengujian

4.1 Perancangan Kasus Uji Pengujian Perangkat Lunak *ZenarcEnc*

Berdasarkan teknik pengujian yang telah dijelaskan, maka dirancang kasus-kasus uji sebagai berikut:

1. Kasus Uji 1
   Kasus Uji 1 bertujuan untuk menguji kebenaran proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi dengan menggunakan algoritma kriptografi *ZSC* menggunakan mode karakter dengan panjang kunci 25 dan 30 byte.

2. Kasus Uji 2
   Kasus Uji 2 bertujuan untuk menguji kebenaran proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi dengan menggunakan algoritma kriptografi *ZSC* menggunakan mode byte dengan panjang kunci 25 dan 30 byte.

3. Kasus Uji 3
   Kasus Uji 3 bertujuan untuk menguji tingkat keamanan data algoritma kriptografi *ZSC* menggunakan mode karakter terhadap pengubahan satu bit atau lebih blok cipherteks, penambahan blok cipherteks semu, dan penghilangan satu atau lebih blok cipherteks.

4. Kasus Uji 4
   Kasus Uji 4 bertujuan untuk menguji tingkat keamanan data algoritma kriptografi *ZSC* menggunakan mode byte terhadap pengubahan satu bit atau lebih blok cipherteks, penambahan blok cipherteks semu, dan penghilangan satu atau lebih blok cipherteks.

4.2 Evaluasi Hasil Pengujian Perangkat Lunak *ZenarcEnc*

Dari hasil pengujian Kasus Uji 1 dan 2, diketahui bahwa perangkat lunak *ZenarcEnc* telah melakukan proses enkripsi dan dekripsi algoritma kriptografi *ZSC* menggunakan mode karakter dan byte dengan panjang kunci 25 dan 30 byte dengan benar.

Proses enkripsi dengan menggunakan kunci tertentu dengan panjang tertentu akan menyandikan isi arsip asal. Proses dekripsi dengan menggunakan kunci yang sama dengan kunci yang digunakan dalam proses enkripsi (kunci simetris) akan mengembalikan isi arsip hasil dekripsi menjadi isi arsip asal.

Sedangkan, kesalahan penggunaan kunci mengakibatkan isi arsip hasil dekripsi tidak sama dengan arsip asal. Kesalahan memasukkan salah satu kunci (Override maupun utama) akan menghasilkan arsip yang berbeda jauh dengan arsip asal.

Kasus Uji 1 dan 2 juga menunjukkan bahwa:

1. Lama waktu yang digunakan untuk proses enkripsi dan dekripsi algoritma kriptografi *ZSC* menggunakan mode byte lebih lama dibandingkan dengan menggunakan mode karakter untuk satu file teks sederhana.

2. Bila mode karakter digunakan untuk file non-teks, maka hasilnya kadang-kadang tidak sesuai dengan perkiraan, karena tidak semua file non-teks mengandung karakter. Dan, mungkin byte terakhir tak terambil, karena satu karakter = 2 byte.

3. Sebaliknya, mode byte aman dipakai untuk file-file dengan tipe teks sederhana, karena relatif sama pembacaannya.

Dari hasil pengujian Kasus Uji 3, diketahui bahwa tingkat keamanan algoritma kriptografi *ZSC* menggunakan mode karakter terhadap manipulasi cipherteks adalah sebagai berikut:

1. Pengubahan satu bit atau lebih blok *ciphertext* akan mengakibatkan terjadinya perubahan terhadap sebuah blok *plaintext* pada arsip hasil dekripsi yang letaknya berkoresponden dengan sebuah blok *ciphertext* yang diubah.

2. Penambahan sebuah blok cipherteks semu akan mengakibatkan terjadinya penambahan sebuah blok plainteks pada arsip hasil dekripsi yang letaknya berkoresponden dengan sebuah blok cipherteks yang ditambahkan. Hal ini juga dapat merusak data dengan baik karena merusak susunan struktur urutan data yang dihasilkan dalam ekspansi *plaintext*.

3. Penghilangan satu atau lebih blok cipherteks akan mengakibatkan terjadinya penghilangan satu atau lebih blok plainteks pada arsip hasil dekripsi yang letaknya berkoresponden dengan sebuah blok cipherteks yang dihilangkan. Hal ini juga dapat merusak data dengan baik karena merusak susunan struktur urutan data yang dihasilkan dalam ekspansi *plaintext*.

Dari hasil pengujian Kasus Uji 4, diketahui bahwa tingkat keamanan algoritma kriptografi *ZSC* menggunakan mode byte terhadap manipulasi cipherteks adalah sebagai berikut:

1. Pengubahan satu bit atau lebih blok *ciphertext* akan mengakibatkan terjadinya perubahan terhadap sebuah blok *plaintext* pada arsip hasil dekripsi yang letaknya berkoresponden dengan sebuah blok *ciphertext* yang diubah.

2. Penambahan sebuah blok cipherteks semu akan mengakibatkan terjadinya penambahan sebuah blok plainteks pada arsip hasil dekripsi yang letaknya berkoresponden dengan sebuah blok cipherteks yang ditambahkan. Hal ini juga dapat merusak data dengan baik karena merusak susunan struktur urutan data yang dihasilkan dalam ekspansi *plaintext*.

3. Penghilangan satu atau lebih blok cipherteks akan mengakibatkan terjadinya penghilangan satu atau lebih blok plainteks pada arsip hasil dekripsi yang letaknya berkoresponden dengan sebuah blok cipherteks yang dihilangkan. Hal ini juga dapat merusak data dengan baik karena merusak susunan struktur urutan data yang dihasilkan dalam ekspansi *plaintext*.

Hasil pengujian ini menunjukkan kelemahan serta keuntungan ZSC baik dalam mode byte maupun karakter, karena dengan mengubah satu hal di ciphertext, selama tidak menambah atau mengurangi, hanya blok tersebut yang terpengaruh.

Tetapi, apabila operasinya adalah menambah atau mengurangi sesuatu, maka mungkin hal itu dapat mengubah urutan *ciphertext* nya dan mengubah isi arsip secara keseluruhan, kecuali bila menambah satu blok saja. Hal itu hanya akan menambah satu blok plaintext baru, dan tak berpengaruh. Sehingga, apabila menambah satu karakter atau byte saja, itu akan merusak keseluruhan sistem *ciphertext*. Tetapi ZSC ini resisten dengan penambahan/pengurangan satu blok atau pengubahan isi.

5. Kesimpulan

Kesimpulan yang dapat diambil dari perancangan, implementasi dan pengujian *Zenarc Super Cipher* ini adalah sebagai berikut.

1. *Zenarc Super Cipher (ZSC)* merupakan salah satu solusi yang baik untuk mengatasi masalah keamanan dan kerahasiaan data yang pada umumnya diterapkan dalam pengiriman dan penyimpanan data melalui media elektronik.

2. *ZSC* dapat diimplementasikan secara efisien sebagai perangkat lunak dengan implementasi sederhana karena fungsi-fungsi yang digunakan cukup sederhana, hanya permutasi dan substitusi, tetapi ekstensif untuk digunakan karena pengolahan kuncinya yang unik.

.

3. Urutan lama waktu yang digunakan untuk proses enkripsi dan dekripsi ZSC adalah mode karakter dahulu, baru mode byte.

4. Apabila terdapat satu perubahan bit/byte/karakter atau penambahan/pengurangan astu blok penuh, hal itu tak terlalu berpengaruh pada keseluruhan blok *ciphertext* dan keamanan data tetap terjaga.

5. Bila terjadi penambahan astu bit/byte/karakter pada satu blok *ciphertext* maka struktur data akan terganggu secara keseluruhan dan mengubah isi arsipnya.

6. Mode karakter hanya cocok digunakan untuk mengenkripsikan file teks sederhana, yang mengandung karakter di dalamnya. Sedangkan, mode byte dapat digunakan untuk mengenkripsikan jensi file apapun yang ada.

7. Tingkat keamanan ZSC ini lumayan tinggi, karena sulit untuk dipecahkan selama tidak mengetahui kedua kuncinya secara bersamaan. Selain itu, karena besar cipherteks lebih besar dari plainteks, maka sulit untuk mengurutkan yang mana yang data asli dan yang mana yang merupakan data sampah.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF5054 Kriptografi. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.