Proteksi Short Message Service (SMS) Pada Smart Device Menggunakan Algoritma Rijndael
sms protection in smart device using rijndael algorithm

I Gusti Agung Gede Arya Kadyanan

Jurusan Ilmu Komputer, Udayana University
Bukit Jimbaran Bali

agung.arya@cs.unud.ac.id

Naskah diterima: 6 Januari 2012; Naskah disetujui: 1 Maret 2012

***Abstract*— SMS (Short Message Service) is the world's most widely used information technology in this century. In addition to cost, delivery speed also led to the development of text-based communication applications. Thus the text-based communication becomes very important, especially with the rapid development of mobile devices such as smart devices and mobile phones. Exchange of information by SMS can also be likened to buying and selling, in which has a large number of confidential information between the sender and the recipient. Must be very dangerous if it fell into a crackers. Encryption is necessary to maintain the confidentiality of messages sent via mobile devices. Talking about encryption would not be separated from the standard used in data encryption. Today many emerging encryption algorithms are quite popular. Call it a very well-known DES algorithm to the new *Rijndael* (AES standard) are discussed in this paper. The algorithm determines the strength in protecting the confidentiality of the data. To affirm the strength of encryption is then combined with a method. The method is to combine the key to the algorithm *Rijndael* with XML serialization and Device ID as key to getting a very strong encryption.**

Keywords*— *Rijndael, Smart device, Device ID,* XML *Serializer

***Abstrak*— Pesan singkat atau SMS adalah aplikasi data yang paling banyak digunakan di dunia teknologi informasi abad ini. Selain murah, kecepatan pengiriman juga menjadi alasan dikembangkannya berbagai aplikasi komunikasi berbasis sms. Dengan demikian komunikasi berbasis sms menjadi hal yang sangat penting terlebih lagi dengan pesatnya perkembangan perangkat *mobile* atau *mobile device* seperti *Smart device* dan *mobile phone*. Pertukaran informasi via sms dapat juga disamakan dengan transaksi jual beli, yang di dalamnya mempunyai banyak sekali informasi yang bersifat rahasia antara pengirim dengan penerima. Tentunya sangat berbahaya apabila sms tersebut jatuh ke tangan orang yang salah atau tidak bertanggung jawab seperti misalnya *cracker*. Maka diperlukan *enkripsi* untuk menjaga kerahasiaan pesan yang dikirim lewat perangkat *mobile* tersebut. Berbicara mengenai *enkripsi* tentunya tidak akan terlepas dari standard yang digunakan dalam *enkripsi* data. Saat ini banyak bermunculan algoritma-algoritma *enkripsi* yang cukup populer. Sebut saja DES yang sangat terkenal sampai algoritma terbaru *Rijndael* (standar AES) yang dibahas pada makalah ini. Algoritma tersebut menentukan kekuatan dalam melindungi kerahasiaan suatu data. Untuk menegaskan kekuatan *enkripsi* ini maka digabungkan dengan sebuah metode. Metode tersebut adalah menggabungkan key pada algoritma *Rijndael* dengan serialisasi XML dan *Device ID* sebagai kuncinya untuk mendapatkan *enkripsi* yang sangat kuat dan aman sehingga sangat sulit untuk dibongkar.**

Keywords*— *Rijndael, Smart device, Device ID,* XML *Serializer

I. PENDAHULUAN

Untuk keamanan pesan, salah satu solusinya adalah menggunakan program khusus proteksi atau *enkripsi*. Saat ini telah banyak beredar program khusus proteksi baik *freeware, shareware*, maupun komersial yang beredar di pasaran yang secara khusus ditujukan untuk mengamankan pesan. Pada umumnya program tersebut tidak hanya menggunakan satu metode saja, tetapi beberapa macam sehingga kita dapat memilih yang menurut kita paling aman. Namun untuk lebih menjamin tingkat keamanan yang lebih tinggi maka sebaiknya kita membuat program *enkripsi* sendiri dengan menggabungkan teknik dan metoda yang telah ada untuk dapat diaplikasikan pada perangkat *smart device*. Salah satunya adalah kombinasi serialisasi XML dan kunci *device ID* dengan menggunakan algoritma *Rijndael*.

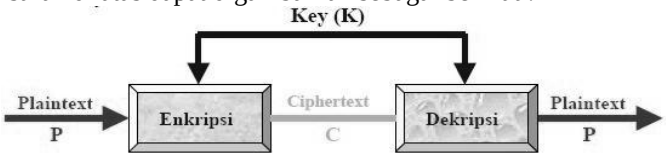
Tujuan dari makalah ini adalah merancang dan membuat program utilitas yang mempunyai kemampuan untuk memproteksi pesan pada *smart device* menggunakan algoritma *Rijndael* dengan kombinasi serialisasi XML dan kunci *device ID*.

II. KAJIAN LITERATUR

A. Kriptosistem

Menurut (Kurniawan, 2004) kriptosistem adalah algoritma kriptografi, ditambah seluruh kemungkinan *plaintext*, *chipertext*, dan kunci-kuncinya. Kriptografi merupakan seni dan ilmu untuk mengamankan pesan. Kriptoanalis adalah orang yang berusaha merusak suatu metode enkripsi untuk mendapatkan teks yang asli secara ilegal.

Suatu pesan yang tidak disandikan disebut sebagai plaintext ataupun dapat disebut juga sebagai cleartext. Proses yang dilakukan untuk mengubah plaintext ke dalam ciphertext disebut enkripsi atau encipherment. Sedangkan proses untuk mengubah ciphertext kembali ke plaintext disebut dekripsi atau *decipherment.* Secara sederhana istilah-istilah di atas dapat digambarkan sebagai berikut :



Gambar 1. Proses Enkripsi/Dekripsi Sederhana

Suatu kriptosistem terdiri dari sebuah algoritma, seluruh kemungkinan *plaintext*, *ciphertext* dan kunci-kunci. Secara umum kriptosistem dapat digolongkan menjadi 2 buah, yaitu :

1. Algoritma Simetri
   Dalam algoritma simetri ini, kunci yang digunakan untuk proses *enkripsi* dan *dekripsi* pada prinsipnya identik, tetapi satu buah kunci dapat pula diturunkan dari kunci yang lainnya. Kunci-kunci ini harus dirahasiakan. Oleh karena itulah sistem ini sering disebut sebagai *secret-key ciphersystem*. Jumlah kunci yang dibutuhkan umumnya adalah :
   $$_nC_2 = \frac{n(n-1)}{2}$$
   dengan n menyatakan banyaknya pengguna.

2. Algoritma Asimetri
   Dalam algoritma *asimetri* ini digunakan dua buah kunci. Satu kunci yang disebut kunci publik (*public key*) dapat dipublikasikan, sedang kunci yang lain yang disebut kunci privat (*private key*) harus dirahasiakan.

Menurut (Mao, 2004) ada beberapa mode operasi yang digunakan dalam meng*enkripsi* atau men*dekripsi* data. Mode operasi tersebut diantaranya :

1. Electronic Code Book (ECB)

Pada mode ini setiap blok *plaintext* di*enkripsi* secara independen menjadi blok *cipher*. Secara matematis dapat dinyatakan :

$$C_i = E_k(P_i)$$
$$P_i = D_k(C_i)$$

2. Cipher Block Chaining (CBC)

Pada proses *enkripsi* mode CBC, blok *plaintext* terlebih dahulu di-XOR-kan dengan blok *ciphertext* hasil *enkripsi* blok sebelumnya. Blok pertama *plaintext* di-XOR-kan dengan suatu *Initialization Vector* (Vektor Awal) yang besarnya sama dengan blok *plaintext*. Secara matematis dapat dinyatakan:

$$C_i = P(C_i - 1)$$
$$P_i = (C_i - 1).(D_k.C_i)$$

Setiap kriptosistem yang baik harus memiliki karakteristik antara lain :

1. Keamanan sistem terletak pada kerahasiaan kunci dan bukan pada kerahasiaan algoritma yang digunakan.
2. Kriptosistem yang baik memiliki ruang kunci (*keyspace*) yang besar.
3. Kriptosistem yang baik akan menghasilkan *ciphertext* yang terlihat acak dalam seluruh tes statistik yang dilakukan terhadapnya.
4. Kriptosistem yang baik mampu menahan seluruh serangan yang telah dikenal sebelumnya.

Namun demikian perlu diperhatikan bahwa bila suatu kriptosistem berhasil memenuhi seluruh karateristik di atas belum tentu ia merupakan sistem yang baik. Banyak kriptosistem lemah yang terlihat baik pada awalnya. Kadang kala untuk menunjukkan bahwa suatu kriptosistem kuat atau baik dapat dilakukan dengan menggunakan pembuktian matematika.

Menurut (Azani, 2006), keamanan dari sebuah teknik penyandian tergantung dari dua hal: algoritma penyandian dan panjang kunci (*key*). Algoritma sangat menentukan kekuatan dari sebuah teknik penyandian, tetapi panjang kunci juga tidak kalah penting dalam menentukan kekuatan sebuah teknik penyandian. Sebagai contoh, apabila seorang *kriptanalis* mengetahui algoritma yang dipakai untuk melakukan teknik penyandian terhadap suatu pesan, maka *kriptanalis* tersebut harus mendapatkan kunci yang dipakai terlebih dahulu sebelum dapat melakukan *dekripsi* terhadap semua *ciphertext* yang dia punya. Satu-satunya cara untuk mendapatkan kunci yang dipakai adalah dengan cara mencoba semua variasi kunci yang ada. Teknik serangan ini sering di kenal dengan nama *brute force*.

Adalah mudah untuk menghitung banyaknya variasi kunci yang ada. Apabila panjang kunci adalah 8 bit, maka ada 2^8 atau 256 kemungkinan kunci yang dapat dicoba. Dari 256 percobaan ini, peluang untuk mendapatkan kunci yang benar adalah 50 persen setelah melalui setengah usaha percobaan. Bila panjang kunci 56 bit, maka ada 2^{56} kemungkinan variasi kunci. Dengan menganggap sebuah superkomputer dapat mencoba satu juta kunci per detik, maka diperkirakan sekitar 2285 tahun untuk menemukan kunci yang benar. Bila menggunakan panjang kunci 64 bit, maka dengan superkomputer yang sama akan membutuhkan 585 ribu tahun. Dengan jangka waktu yang lama ini, maka dapat dipastikan bahwa pesan yang disandikan tersebut tidak mempunyai arti lagi apabila telah berhasil dilakukan *dekripsi*.

Dengan melihat situasi ini, maka kriptografi yang baik akan memilih untuk menggunakan sepanjang mungkin kunci yang akan digunakan, namun hal ini tidak dapat diterapkan begitu saja. Semakin panjang kunci, semakin lama pula waktu yang digunakan oleh komputer untuk melakukan proses *enkripsi*. Oleh sebab itu, panjang kunci yang akan digunakan hendaknya memperhatikan 3 hal, yaitu seberapa penting data yang akan dirahasiakan, berapa lama waktu yang dibutuhkan agar data tersebut tetap aman, dan seberapa kuat kemampuan *kriptanalis* dalam memecahkan teknik penyandian . Saat ini yang paling banyak dipakai adalah kunci dengan panjang 128 bit karena panjang kunci ini dianggap paling optimal untuk

saat ini. Dalam *kriptografi*, terdapat dua macam *key* atau kunci yang digunakan dalam proses *enkripsi* dan *dekripsi*. Dua macam *key* tersebut adalah :

1. Public Key
   Diungkapkan pertama kali oleh Diffie dan Hellman pada tahun 1976, kunci jenis ini banyak digunakan dalam *enkripsi* data yang berhubungan dengan jaringan komputer. Kunci yang ada dengan sengaja disebarluaskan kepada publik secara bebas. Sehingga pada saat proses pengiriman data yang telah terenkrip, pihak pengirim dan penerima menggunakan public *key* yang sama, tetapi di lain pihak masing-masing telah mempunyai private key sendiri yang nantinya dikombinasikan dengan public key yang telah ada.
2. Single Key
   Seringkali disebut sebagai algoritma konvensional dalam proses enkripsi. Pada jenis ini kunci yang dibutuhkan harus tetap terjaga kerahasiannya, dalam arti bahwa hanya pemilik kunci yang bisa menjalankan proses enkripsi dan dekripsi. Dalam implemetasinya, kunci yang digunakan adalah kunci dari jenis ini. Hal ini disebabkan bahwa proteksi dokumen yang dimaksud adalah proteksi dokumen dalam suatu personal komputer, bukan pada suatu sistem jaringan yang luas, sehingga memerlukan suatu transfer data yang kompleks. Selain itu, sebagian besar algoritma enkripsi sampai saat ini masih banyak menggunakan kunci jenis single sebagai pedoman dalam proses enkripsi. Single Key juga biasa disebut teknik kriptosistem simetrik, dimana intinya adalah pemilik yang mengenkripsi dokumen harus memberitahu kunci enkripsinya kepada pemilik yang akan menerima pesan dan melakukan dekripsi dokumen yang terenkripsi.

A. Encryption Standard

Pada tanggal 2 Januari 1997, *National Institute* of *Standard and Technology* (NIST) mengumumkan algoritma baru dalam kriptografi modern sebagai pengganti algoritma DES yang disebut sebagai AES atau kepanjangan dari Advanced Encryption Standard. AES sendiri dilombakan secara terbuka dan pada tanggal 2 Oktober 2000, NIST secara resmi menetapkan algoritma *Rijndael* sebagai pemenangnya.

Menurut (Kurniawan, 2004) nama algoritma *Rijndael* diambil dari nama kedua penciptanya, Joan **Dae**men dan Vincent **Rij**men. *Rijndael* memiliki ukuran blok *cipher* dan panjang kunci yang bervariasi antara 128, 192 dan 256 bit.

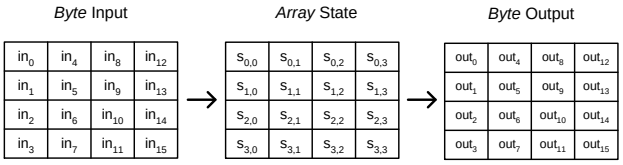
TABEL 1. KOMBINASI KUNCI BLOK-RONDE

	Panjang Kunci (N_k words)	**Ukuran Block (N_b words)**	**Jumlah Round (N_r)**
***Rijndael* 128-bit**	4	4	10
***Rijndael* 192-bit**	6	4	12
***Rijndael* 256-bit**	8	4	14

Menurut (Kurniawan, 2004), nama algoritma *Rijndael* diambil dari nama kedua penciptanya, Joan **Dae**men dan Vincent **Rij**men. *Rijndael* memiliki ukuran blok *cipher* dan panjang kunci yang bervariasi antara 128, 192 dan 256 bit.

Dari tabel 1. terlihat bahwa *Rijndael* 128 bit menggunakan panjang kunci $N_k = 4$ *word* yang setiap *word*nya berisi 32 bit sehingga total kuncinya 128 bit. Ukuran blok masukan 4 *word* (128 bit), sedangkan jumlah *round* (N_r) sebanyak 10 *round*. *Rijndael* 256 bit memiliki panjang kunci 256 bit, blok masukan *plaintext* 128 bit dan jumlah *round* 14.

Pada dasarnya, operasi *Rijndael* dilakukan terhadap *array byte* 2 dimensi yang disebut dengan *state*. *State* terdiri dari 4 baris *byte*. Setiap baris mengandung N_b *byte*, dimana N_b adalah panjang blok dibagi dengan 32. Jadi jika *plaintext* terdiri dari 128 bit, maka $N_b = \frac{128}{32} = 4$ *byte* (32 bit). Di dalam *state*, array dinyatakan dengan simbol S, dan setiap *byte* memiliki 2 indeks. Indeks pertama bernilai antara $0 \le r < 4$ yang merujuk kepada nomor baris. Indeks ke-2 merujuk pada nomor kolom yang bernilai $0 \le c < N_b$. Merujuk pada Tabel 1, N_b bernilai 4. Kemudian, setiap nilai dalam *state* dapat dianggap berbentuk $S_{r,c}$ atau $S_{[r,c]}$.



Gambar 2. Masukan dan Keluaran Array State

Dari gambar 2. di atas terlihat masukan berupa $in_0 ... in_{15}$. Setiap in berisi 8 bit, sehingga total masukan adalah 8×16 bit = 128 bit. Dalam setiap *round* (tahapan), masukan akan diolah di dalam state dan akhirnya akan menghasilkan keluaran $out_0 ... out_{15}$ sebanyak 128 bit.

Pada awalnya, masukan $in_0 ... in_{15}$ akan disalin ke dalam *state* dengan mengikuti rumus $S[r,c] = in[r + 4_c]$. Dan pada keluaran, *state* akan dikopikan ke keluaran $out_0 ... out_{15}$ dengan mengikuti rumus $Out[r + 4_c] = S[r,c]$. Misalkan $in_0 = a_{0,1}, in_1 = a_{1,0}$ dan seterusnya, maka *array state* yang dimasuki masukan $a_{0,1} ... a_{33}$ akan menjadi :

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Gambar 3. Dari masukan ke state

Dari gambar 3. dapat dianalisis dari sisi lain. Bila dibaca matrik tersebut dari atas ke bawah, maka akan dihasilkan sebuah array 1 dimensi 32-bit sebagai berikut :

1. $w_0 = a_{0,0}a_{1,0}a_{2,0}a_{23,0}$
2. $w_1 = a_{0,1}a_{1,1}a_{2,1}a_{23,1}$
3. $w_0 = a_{0,2}a_{1,2}a_{2,2}a_{23,2}$
4. $w_1 = a_{0,3}a_{1,3}a_{2,3}a_{23,3}$

Setiap a berisi 8-bit, sehingga setiap w akan berisi 32 bit. Operasi *enkripsi Rijndael* dapat dinyatakan dengan *pseudocode* sebagai berikut :

$in[4\times N_b]$, byte $out[4\times N_b]$, word $w[N_b\times(N_r+1)]$

Chiper (byte $in[4\times N_b]$, byte $out[4\times N_b]$, word $w[N_b\times(N_r+1)]$)

```
BEGIN
    byte state 4, Nb
    state = in
    XorRoundKey(state, w)
    FOR round = 1 step 1 to Nr − 1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        XorRoundKey(state, w + round × Nb)
    end for
    SubBytes(state)
    ShiftRows(state)
    XorRoundKey(state, w + Nr × Nb)
    out = state
END
```

Dari *pseudocode* di atas, dapat diketahui bahwa *enkripsi* dilakukan dengan fungsi *Cipher* yang memiliki parameter masukan $in = 16$, keluaran $out = 16$ *byte* dan array 1-dimensi w 44 *byte* untuk *Rijndael* 128-bit. Proses yang dilakukan pada setiap *round* adalah identik (dari *round* ke-0 sampai dengan *round* ke $N_r - 1)$, kecuali untuk *round* terakhir N_r .

B. Enkripsi

Dalam tiap putaran *enkripsi*, algoritma ini memerlukan empat langkah yaitu :

1. SubBytes
   Operasi SubBytes merupakan operasi subtitusi tidak-linear yang beroperasi secara independen pada setiap byte state menggunakan tabel subtitusi (kotak-S).
2. Transformasi ShiftRows
   Pada transformasi ShifRows, byte-byte pada 3 baris terakhir (baris 1, 2 dan 3) dari state, digeser secara memutar dengan jumlah pergeseran yang berbeda-beda. Baris pertama atau baris 0 tidak dilakukan pergeseran.
3. Transformasi MixColoumn
   Transformasi MixColumn beroperasi pada state kolom demi kolom, memerlukan setiap kolom sebagai polinomial 4-suku.
4. Transformasi AddRoundKey
   Pada operasi ini, kunci round ditambahkan ke state dengan operasi XOR. Setiap kunci round terdiri atas N_b word.

Dalam tiap putaran *enkripsi*, algoritma ini memerlukan empat langkah yaitu :

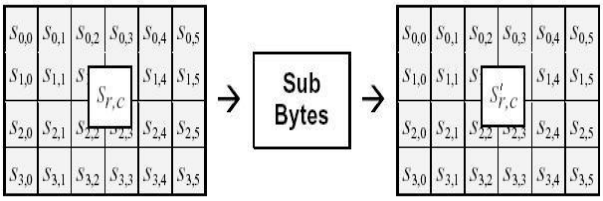
1. SubBytes
   Operasi SubBytes merupakan operasi subtitusi tidak-linear yang beroperasi secara independen pada setiap *byte state* menggunakan tabel subtitusi (kotak-S).

hex	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

x

Gambar 4. Kotak Subtitusi (Kotak-S) dalam hexadecimal

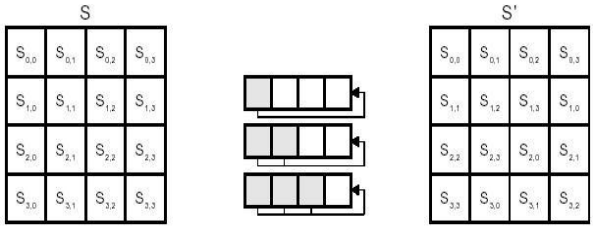
Pengaruh subtitusi kotak-S diperlihatkan pada gambar 5. berikut ini :



Gambar 5. Operasi SubBytes

Dari gambar 5. di atas terlihat jelas bahwa transformasi SubBytes terhadap $S[r,c]$ akan menghasilkan $S'[r,c]$. Dan dari gambar 5, bila $S[r,c] = 53_{hex}$, maka operasi SubBytes akan memberikan nilai $S'[r,c] = [ed]$. Dengan menarik garis lurus horisontal dengan $x = 5$ dan garis vertikal $y = 3$ dari kotak-S maka akan ditemukan nilai ed . Jadi nilai ed_{hex} akan mensubtitusi 53_{hex}.

2. Transformasi ShiftRows
   Pada transformasi ShifRows, *byte-byte* pada 3 baris terakhir (baris 1, 2 dan 3) dari *state*, digeser secara memutar dengan jumlah pergeseran yang berbeda-beda. Baris pertama atau baris 0 tidak dilakukan pergeseran.

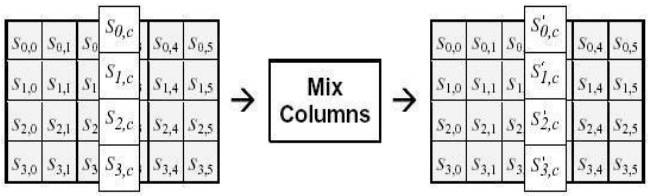


Gambar 6. ShiftRows memutar tiga baris terakhir

3. Transformasi MixColoumn
Transformasi MixColumn beroperasi pada *state* kolom demi kolom, memerlukan setiap kolom sebagai *polinomial* 4-suku. Kolom dianggap sebagai polinomial pada $GF(2)$ dan dikalikan dengan *polinomial* $a(x) \bmod (x^4+1)$, dimana ditetapkan $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. Ini dapat ditulis sebagai perkalian matrik. Dengan $s'(x) = a(x) \otimes s(x)$ maka :

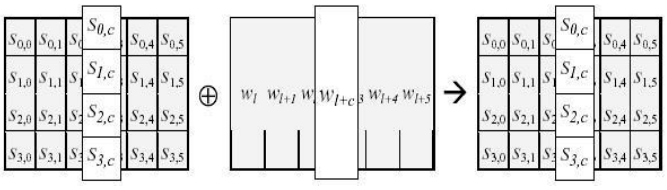
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Untuk $0 \le c \le N_b$



Gambar 7. Operasi MixColoumn pada state tiap kolom

4. Transformasi AddRoundKey
Pada operasi ini, kunci *round* ditambahkan ke *state* dengan operasi XOR. Setiap kunci *round* terdiri atas N_b word.

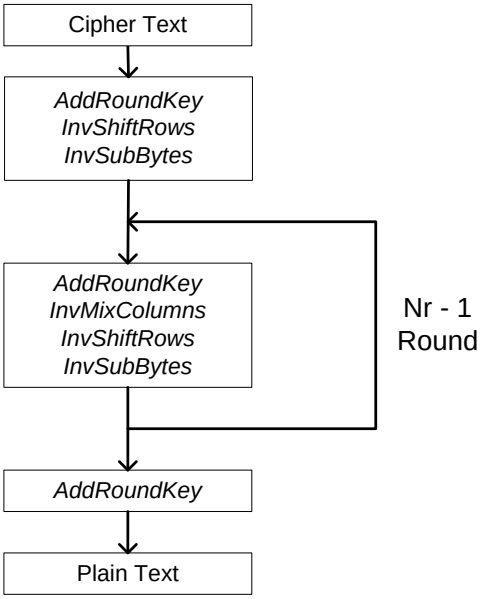


Gambar 8. Operasi AddRoundKey meng-XOR kolom state dengan subkey

Dimana $[W_t]$ adalah *subkey* yang diturunkan dari kunci utama.

C. Dekripsi

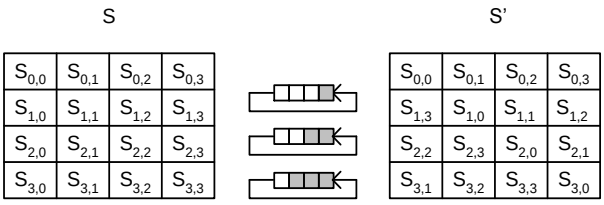
Setelah melalui proses *enkripsi* maka untuk dapat kembali ke bentuk semula, pesan harus melalui proses *dekripsi*. Transformasi *cipher* dapat dibalikkan dan diimplementasikan dalam arah yang berlawanan untuk menghasilkan *inverse cipher* yang mudah dipahami untuk algoritma AES. Transformasi byte yang digunakan pada invers *cipher* adalah InvShiftRows, InvSubBytes, InvMixColumns, dan AddRoundKey. Algoritma *dekripsi* dapat dilihat pada skema berikut ini:



Gambar 9. Diagram Alir Proses Dekripsi

Berikut penjelasan secara detail proses *dekripsi*:

1. InvShiftRows
InvShiftRows adalah transformasi byte yang berkebalikan dengan transformasi ShiftRows. Pada transformasi InvShiftRows, dilakukan pergeseran bit ke kanan sedangkan pada ShiftRows dilakukan pergeseran bit ke kiri. Pada baris kedua, pergeseran bit dilakukan sebanyak 3 kali, sedangkan pada baris ketiga dan baris keempat, dilakukan pergeseran bit sebanyak dua kali dan satu kali.



Gambar 10. Transformasi InvShiftRows

2. InvSubBytes
InvSubBytes juga merupakan transformasi bytes yang berkebalikan dengan transformasi SubBytes. Pada InvSubBytes, tiap elemen pada *state* dipetakan dengan menggunakan tabel *inverse S-Box*. Tabel ini berbeda dengan tabel *S-Box* dimana hasil yang didapat dari tabel ini adalah hasil dari dua proses yang berbeda urutannya, yaitu transformasi *affine* terlebih dahulu, baru kemudian perkalian *invers* dalam GF(2^8).

$$\begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Gambar 11. Matriks Invers Affine

Perkalian *invers* yang dilakukan pada transformasi InvSubBytes ini sama dengan perkalian *invers* yang dilakukan pada transformasi SubBytes.

3. InvMixColumns

Pada InvMixColumns, kolom-kolom pada tiap *state* (*word*) akan dipandang sebagai polinom atas GF(2^8) dan mengalikan modulo $x^4 + 1$ dengan polinom tetap $a^{-1}(x)$ yang diperoleh dari:

$$a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}.$$

Atau dalam matriks :

$$s'(x) = a(x) \otimes s(x)$$

$$\begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Gambar 12. Matriks InvMixColumns

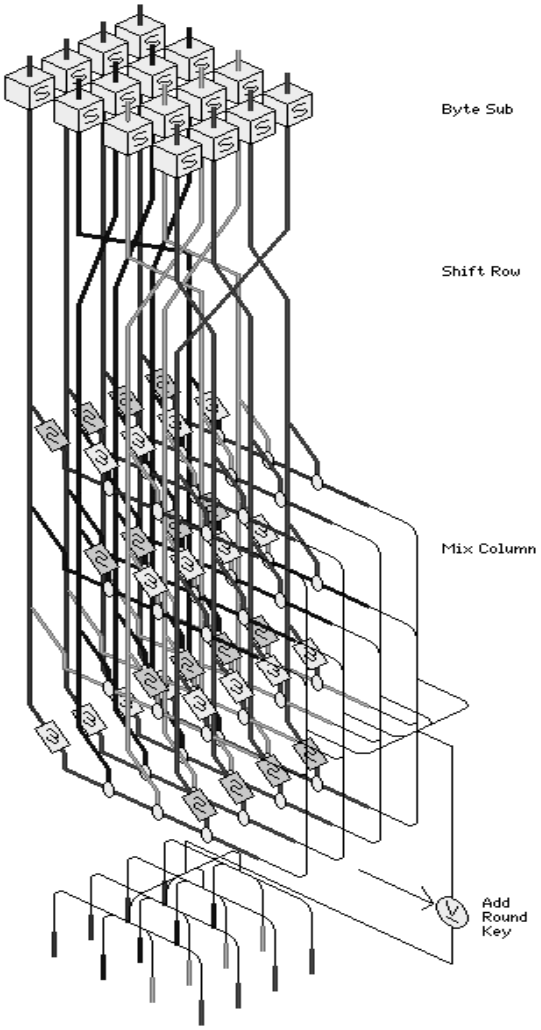
Hasil dari perkalian di atas adalah :

$$s_{0,c}' = (\{0E\} \bullet s_{o,c}) \oplus (\{0B\} \bullet s_{1,c}) \oplus (\{0D\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$
$$s_{1,c}' = (\{09\} \bullet s_{o,c}) \oplus (\{0E\} \bullet s_{1,c}) \oplus (\{0B\} \bullet s_{2,c}) \oplus (\{0D\} \bullet s_{3,c})$$
$$s_{2,c}' = (\{0D\} \bullet s_{o,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0E\} \bullet s_{2,c}) \oplus (\{0B\} \bullet s_{3,c})$$
$$s_{3,c}' = (\{0B\} \bullet s_{o,c}) \oplus (\{0D\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0E\} \bullet s_{3,c})$$

4. Inverse AddRoundKey

Transformasi Inverse AddRoundKey tidak mempunyai perbedaan dengan transformasi AddRoundKey karena pada transformasi ini hanya dilakukan operasi penambahan sederhana dengan menggunakan operasi bitwise XOR.

Selain 4 transformasi yang telah disebutkan di atas, masih diperlukan satu operasi lagi yaitu *ekspansi* kunci untuk memenuhi kebutuhan *subkey* yang dapat mencapai ribuan bit untuk melakukan *enkripsi*, sementara kunci *enkripsi* yang disediakan hanya 128-bit hingga 256 bit. Total *subkey* yang diperlukan adalah $N_b(N_r + 1)$ word. Jadi bila digunakan *enkripsi* 128 bit, maka akan dibutuhkan $4(10+1) = 44$ word $= 44 \times 32$ bit = 1408 bit kunci (*subkey*). Dengan kata lain, *enkripsi* 128 bit, diekspansi menjadi 1408 bit, melalui proses yang disebut dengan *key schedule*. *Subkey* sebanyak ini diperlukan karena setiap *round* membutuhkan N_b word ditambah satu word subkey untuk di awal. *Key schedule* menghasilkan *array linear* word $[w_1]$ sebesar 4 *byte*, sedangkan I memiliki nilai $\leq I < N_b(N_r + 1)$.



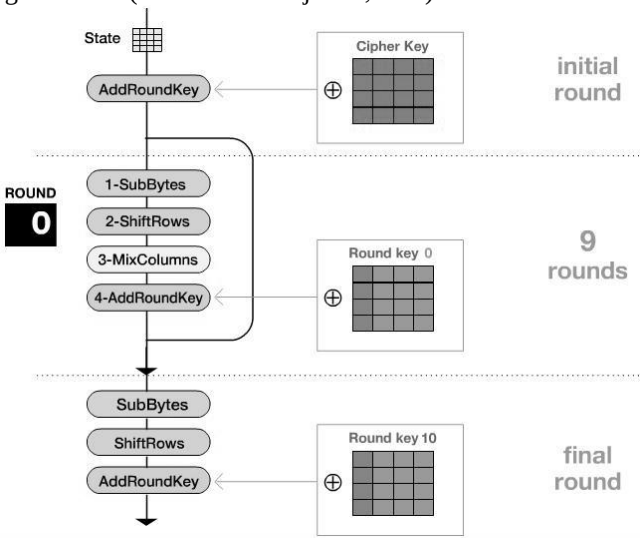
Gambar 13. Transformasi proses enkripsi algoritma Rijndael (Daemen dan Rijmen, 1999)

Untuk proses *enkripsi Rijndael* sendiri, ditunjukkan pada gambar 14. Gambar 14. menunjukkan proses *enkripsi Rijndael* 128 bit, proses akan dilakukan sebanyak 10 *round* hingga menghasilkan data terenkrip, dalam hal ini adalah file serialisasi dari *device ID* dengan blok *cipher* dan panjang kunci 128 bit.

Setelah proses *enkripsi* berhasil dilakukan maka program akan menghapus file asli. Ini dilakukan agar keamanan yang diperoleh lebih maksimal. Untuk itu sebelum melakukan *enkripsi* data, program akan menganjurkan agar data di*back up* terlebih dahulu.

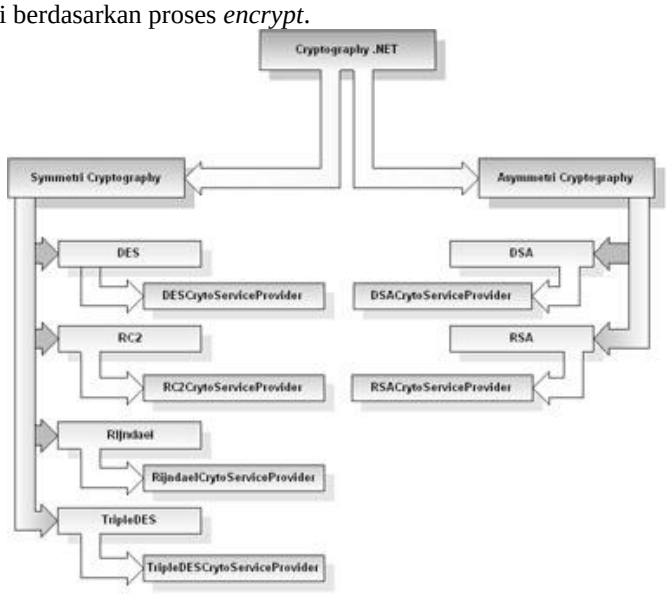
Salah satu keuntungan dari penggunaan algoritma tersebut adalah terciptanya sebuah kunci rahasia yang sebelumnya

telah disepakati oleh pihak yang akan melakukan *dekripsi* dengan pihak yang melakukan *enkripsi* data. Selain itu kecepatan *enkripsi* juga menjadi pertimbangan yang lebih dari algoritma ini (Daemen dan Rijment,1999).



Gambar 14. Proses enkripsi Rijndael untuk 128 bit (Daemen dan Rijmen, 1999)

Visual Basic.NET telah menyediakan *class* khusus untuk melakukan proses *enkripsi* dengan algoritma *Rijndael*. Class tersebut diturunkan dari namespaces Cryptography dan ditempatkan sejajar dengan algoritma yang lain seperti SHA dan DES. Class *Rijndael* ini memiliki properti dan method yang mampu *generate key* serta implementasi *managed class* untuk tipe data *string*. Dengan demikian, implementasi algoritma *Rijndael* menjadi lebih mudah. Pada aplikasi ini terdapat beberapa method yang dikembangkan berdasarkan apa yang telah disediakan oleh Visual Basic.NET seperti method untuk browse file, method untuk *enkripsi*, *dekripsi*, method untuk serialisasi XML, menghapus file XML dan mengambil device ID (Deitel dan Nieto, 2002). Pembahasan mengenai Cryptography dalam framework.NET cukup menempati banyak tempat. Secara umum sistem Cryptography digambarkan seperti pada gambar 15. di bawah ini berdasarkan proses *encrypt*.



Gambar 15. Sistem Cryptography berdasarkan proses encrypt dalam .NET (MSDN .NET library)

Baik sistem *Cryptography* menggunakan algoritma Simetri ataupun algoritma Asimetri masing-masing mempunyai kelebihan dan kekurangan. Sistem algoritma *Crytography* dalam framework .NET ditangani oleh *CryptoServiceProvider*. Semuanya dengan memanfaatkan *namespace System.Security.Cryptography.*

D. Serialisasi XML

Menurut Ziegeler dan Huber (1999), serialisasi adalah proses yang berjalan *runtime* untuk mengkonversi objek ke dalam bentuk sekuensi byte secara linear. Manfaat utama dari sebuah proses serialisasi adalah pemrosesan lebih lanjut hasil serialisasi ke sebuah bentuk blok *memori* untuk ditransfer lebih lanjut melalui jaringan dengan protokol yang umum.

Serialisasi XML atau dalam bahasa umumnya XML Serializer merupakan sebuah *class* dalam Visual Basic.NET dari namespace *XML.Runtime. Serialization* yang berfungsi untuk melakukan serialisasi sebuah obyek ke dalam sebuah file XML dan sebaliknya. Serialisasi obyek secara umum akan memanfaatkan kemampuan Visual Basic.NET untuk membaca memory stream dalam format tipe byte. Sehingga obyek yang dapat diserialisasi lebih bervariasi, misal dokumen image, dokumen PDF hingga runtime library.

Saat proses serialisasi yang menghasilkan dokumen XML berlangsung nantinya akan melalui dua tahap yaitu :

1. Sebuah dokumen XML yang telah valid (dalam bentuk dokumen XSD schema) yang telah menyertakan semua properti dari objek yang telah terserialisasi.
2. Dari XSD schema tersebut akan di*encoding* menjadi sebuah dokumen XML yang mampu dibaca dan class *reader* ataupun *writer* dalam lingkup Visual Studio.NET.

III. ANALISIS PERMASALAHAN

Proteksi data pada *Smart device* yang telah ada saat ini, hampir seluruhnya hanya berkonsentrasi ke perlindungan data dengan hanya mengandalkan algoritma tanpa menggabungkan dengan metode tertentu. Kebanyakan pula justru hanya mengandalkan penggunaan *password* sebagai *key*. Hal ini menjadikan algoritma yang sebenarnya ampuh tersebut menjadi lemah karena sebenarnya suatu algoritma apabila diketahui cara kerjanya maka dengan mudah orang dapat membongkarnya.

Sebagaimana telah dibahas pada bab sebelumnya, dalam makalah ini metode yang digunakan adalah menggabungkan algoritma *Rijndael* dengan serialisasi XML dan kunci *Device ID* pada *smart device*. Metode ini diharapkan dapat mengamankan pesan dengan kuat karena untuk membongkarnya maka harus pula membongkar serialisasi dan menggandakan *Smart device*, dimana cara terakhir ini sulit dilakukan. Untuk mencegah aksi pembongkaran *enkripsi*, maka pesan akan dilindungi kerahasiaannya menggunakan *Device ID* sebagai *key*. Setelah itu sebelum proses *enkripsi* dijalankan maka didahului dengan proses serialisasi terhadap pesan ke dalam bentuk XML. Hal ini dimaksudkan agar jika seandainya sampai terjadi pencurian *key*, pesan tetap aman karena harus melalui proses deserialisasi sebelum kembali ke bentuk semula.

Menurut (Philippop, 2001) setiap *Smart device* pasti memiliki *Device ID* yang bersifat unik, artinya *Device ID* tersebut pasti berbeda antara yang satu dengan yang lain bahkan untuk produsen yang sama atau merek yang sama.

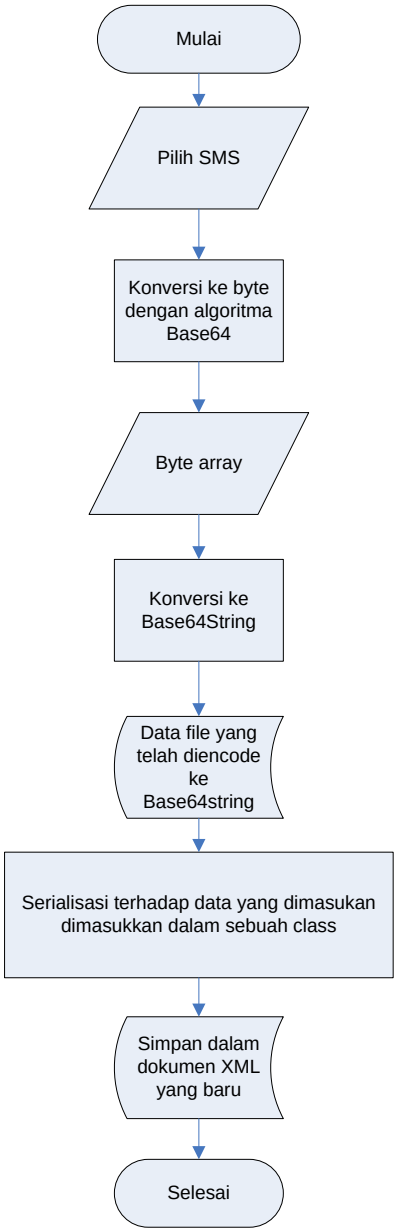
Berikut ini *pseudo-code* untuk mengambil *device ID Smart device* :

1. Tambahkan *class device ID* dengan library *"CoreDll.dll"*
2. Namespace System.Security.Cryptography
3. Lakukan proses ambil *Device ID*

IV. METODE PENELITIAN

A. *Kompresi File*

Dalam dunia komputer termasuk perangkat mobile seperti *Smart device*, pemampatan file digunakan dalam berbagai keperluan, jika anda ingin membackup data, anda tidak perlu menyalin semua file aslinya. Dengan memampatkan (mengecilkan ukurannya) file tersebut terlebih dahulu maka kapasitas tempat penyimpanan yang diperlukan akan menjadi lebih kecil. Jika sewaktu-waktu data tersebut diperlukan, baru dikembalikan lagi ke file aslinya. Dalam hal ini data yang dimaksud adalah pesan singkat atau sms yang akan dikirim melalui *smart device* tersebut.



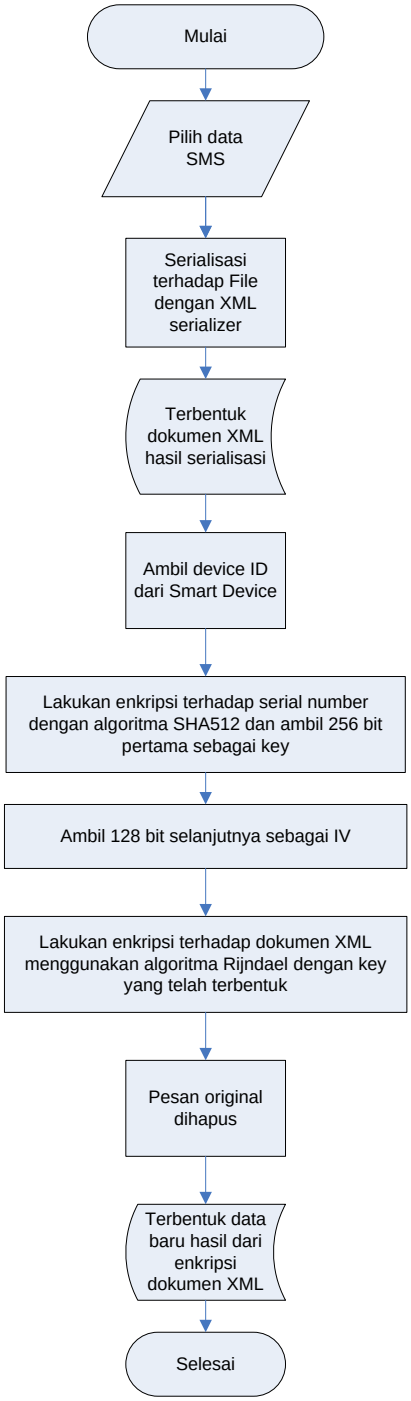
Gambar 16. Alur serialisasi data pada Smart device

B. *Proses serialisasi*

Proses serialisasi terhadap pesan dimulai dari pemanggilan pesan ke dalam blok memori. Serialisasi terhadap data tersebut outputnya akan dijadikan format XML, namun sebelumnya akan dimasukkan ke dalam sebuah *class* tersendiri yang memiliki properti pengaturan agar data saat dideserialisasi mampu kembali ke bentuk semula dengan sempurna (Gambar 16).

C. *Enkripsi dokumen XML*

Hasil dari proses serialisasi yang berupa dokumen XML selanjutnya akan di*enkripsi* dengan menggunakan algoritma *Rijndael* (Gambar 17).



Gambar 17. Alur enkripsi data dengan algoritma Rijndael

Dalam *enkripsi* yang dilakukan nantinya akan langsung menggunakan *class Rijndael* yang telah disediakan oleh Visual Basic.NET. *Class Rijndael* tersebut merupakan turunan dari namespace Cryptography dengan struktur sebagai berikut :

1. System.Object
2. System.Security.Cryptography.SymmetricAlgorithm
3. System.Security.Cryptography.Rijndael

E. Proses Hashing

Menurut (Kurniawan, 2004) *device ID* akan di*enkripsi* terlebih dahulu dengan menggunakan algoritma SHA-1 dan 256 bit pertama hasil dari proses tersebut akan menjadi *key*. SHA-1 merupakan algoritma hash yang dibuat oleh *National Security Agent* (NSA). SHA-1 berarti bahwa proses *enkripsi* yang dilakukan mencapai 512 bit. Dengan demikian *Device ID* yang akan dijadikan *key* akan selalu memiliki panjang kunci 256 bit meskipun panjang rata-rata *Device ID* adalah 33 karakter. Hal ini dilakukan untuk menghasilkan keamanan data lebih kuat.

F. Proses dekompresi

Proses dekompresi bertujuan mengembalikan ukuran file yang sebelumnya telah dimampatkan sebelum dilakukannya proses *enkripsi*. Adapun metode yang dipergunakan dalam dekompresi ini adalah sama dengan metode yang digunakan waktu mengkompres file yaitu *GZip*. Dalam memanfaatkan sebuah metode kompresi file, tidak semua jenis file dapat dilakukan kompresi dengan baik.

G. Proses dekripsi

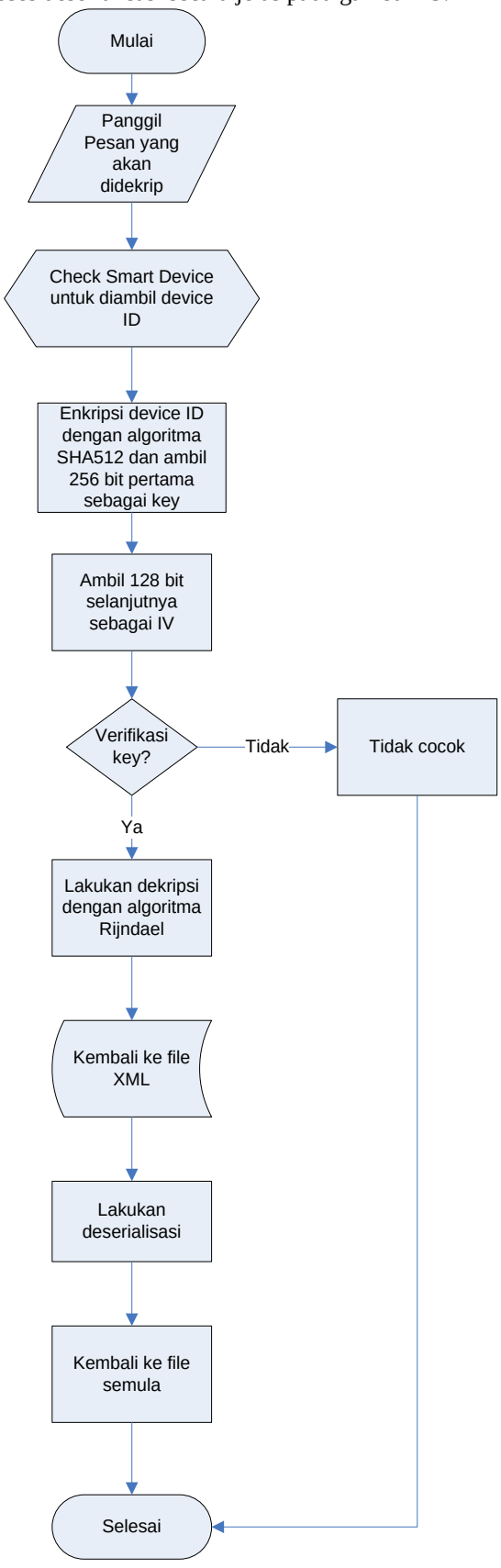
Untuk mengembalikan data yang telah ter*enkripsi* maka harus melalui proses yang dinamakan *dekripsi*. Proses *dekripsi* sendiri secara umum merupakan kebalikan dari proses *enkripsi*. Algoritma yang digunakan termasuk jenis algoritma simetris dan menggunakan *key* yang sama dengan saat proses *enkripsi*. Urutan proses *dekripsi* dari awal dapat dilihat pada gambar 18 berikut ini.

Sama seperti pada saat *enkripsi*, *device ID* yang diambil juga di*enkripsi* terlebih dahulu dengan menggunakan algoritma SHA-1 untuk dijadikan *key*. Langkah-langkahnya pun sama dengan saat proses *enkripsi* pesan. Hasil dari proses *enkripsi device ID* inilah yang akan dicocokkan dengan *key* saat melakukan *enkripsi* pesan. Apabila *key* yang dipakai pada saat proses *dekripsi* pesan tidak sama dengan saat proses *enkripsi* pesan maka dipastikan tidak akan berjalan. Proses *dekripsi* untuk algoritma *Rijndael* juga menggunakan *class* yang sama dengan saat proses *enkripsi* dilakukan.

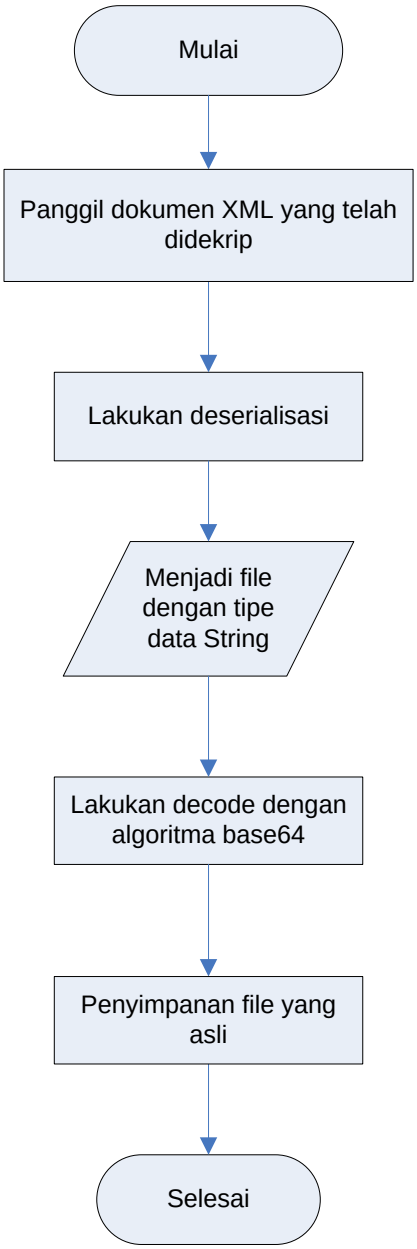
Setelah proses dekrip telah selesai dan pesan berhasil didekrip maka pesan kembali ke bentuk dokumen XML terlebih dahulu. Pesan dalam bentuk dokumen XML tersebut tetap belum bisa digunakan sebagaimana apabila pesan itu ada dalam bentuk aslinya. Sebelum diproses lebih lanjut menjadi pesan yang asli maka dokumen XML tersebut harus dibongkar terlebih dahulu atau di-*deserialisasi*.

H. Proses Deserialisasi

Apabila proses dekrip berjalan dengan sukses maka pesan akan kembali ke bentuk dokumen XML. Berikut dapat dilihat proses deserialisasi secara jelas pada gambar 19.



Gambar 18. Alur dekripsi dengan algoritma Rijndael



Gambar 19. Alur deserialisasi dokumen XML

Agar pesan dapat kembali ke bentuk semula diperlukan proses deserialisasi, yang juga kebalikan dari proses serialisasi.

V. HASIL PENELITIAN DAN PEMBAHASAN

Uji coba pertama dilakukan dengan melakukan proses *enkripsi* dan *dekripsi* pada pesan yang dikirim via *smart device* yang tentunya telah diinstal aplikasi *enkripsi* ini. Berikut beberapa skenario pengujian :

1. Skenario pengujian pertama yaitu dengan melakukan instalasi dan mengimplementasikan aplikasi enkripsi ini pada beberapa jenis smart device seperti 02 Xda II dan O2 Xda II mini untuk menguji kemampuan adaptasi dari aplikasi ini, sebagaimana tampak pada gambar 20 dan 21. dibawah ini.

Gambar 20. Tampilan Login

Gambar 21. Uji Coba Login

2. Melakukan penggantian *password* dengan tujuan agar tidak setiap orang dapat menggunakan aplikasi ini secara sembarangan.

Gambar 22. Tampilan Ganti Password

Dimana untuk ketarangan yaitu:

1. Field 1, inputkan *password* lama.
2. Field 2, inputkan *password* baru.
3. Field 3, menginputkan kembali *password* baru.
4. Field 4, tombol OK untuk melanjutkan proses penggantian *password*.

5. Field 5, tombol Cancel untuk membatalkan proses penggantian *password*.
6. *Virtual Keyboard*, fasilitas untuk mengetikkan karakter.

Gambar 23. Tampilan Ganti *Password*

7. Mengirimkan pesan yang telah terenkripsi dan melakukan dekripsi dengan tujuan melihat keutuhan pesan setelah proses enkripsi apakah masih dapat dibaca atau tidak. Detailnya dapat dilihat pada tabel 2. berikut ini.

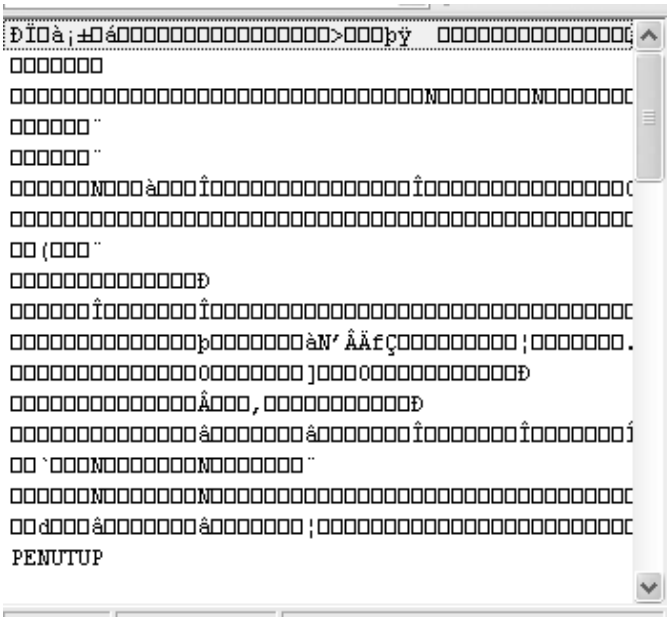
TABEL 2. HASIL PENGUJIAN PESAN

Jenis PDA	Panjang Pesan (karakter)	Waktu Enkripsi (sec)	Waktu Dekripsi (sec)	Keutuhan Pesan
O2 Xda IIs	150	2,8	2,5	Utuh
O2 Xda II mini	150	3,4	3,2	Utuh

8. Berikut tampilan form untuk pengiriman pesan.

Gambar 24. Tampilan Ganti Password

9. Mencoba membuka pesan hasil enkripsi dengan aplikasi pesan lain.



Gambar 25. Tampilan Pesan jika dibuka dengan aplikasi lain

VI. KESIMPULAN DAN REKOMENDASI

A. *Kesimpulan*

Setelah melakukan analisis, perancangan dan pembuatan aplikasi ini serta evaluasi hasil penelitiannya, maka dapat diambil kesimpulan sebagai berikut :

1. Aplikasi pengamanan dengan *Rijndael* ini dapat diterapkan jenis *smart device*.
2. Performa aplikasi ini sangat bergantung kepada spesifikasi *Smart device*.
3. Aplikasi ini memproteksi data dengan lebih maksimal karena data harus melalui proses serialisasi XML terlebih dahulu.
4. Penggunaan *key Rijndael* dengan Device ID sebagai *key* dapat meminimalisasi penggandaan *key* itu sendiri. Ini karena *device ID* bersifat unik.
5. Karena aplikasi ini sangat mudah untuk diimplementasikan maka sangat mungkin untuk dikembangkan sebagai sarana proteksi pada aplikasi sistem informasi pada *Smart device*.

B. *Rekomendasi*

Harapan dari penulis nantinya aplikasi ini suatu ketika dapat diterapkan di semua jenis *gadget* tidak terkecuali telepon selular biasa.

DAFTAR PUSTAKA

Azani, D. (2006). Tugas Akhir: *Proteksi Data Dengan Kombinasi Serialisasi Xml Dan Usb Dongle Menggunakan Algoritma Rijndael*. Surabaya: STIKOM.

Daemen, J., Rijmen, V. (1999). *The Rijndael Block Cipher AES, (Online),*(http://www.esat.kuleuven.ac.be/~rijmen/*Rijndael/*, diakses 15 Pebruari 2007).

Deitel, H.M., Deitel, P.J., and Nieto, T.T. (2002). *Visual Basic.NET How To Program Second Edition*. New Jersey: Prentice Hall.

Kurniawan, Y. (2004). *Kriptografi Keamanan Internet dan Jaringan Komunikasi*. Bandung: Informatika Bandung.

Mao, W. (2004). *Modern Cryptography*. New York: Prentice Hall Inc.

Philippop, V. (2001). *How Can I Get A Serial Number Of Smart device 2002 Device*, (Online),(http://www.pocketpcdn.com/sections/deviceinfo.html, diakses 15 Pebruari 2007).

Stiawan, D. (2004). *Talk Show Computer Easy di Radio Sonora FM Kerjasama Radio Sonora FM, dan PT. Elex Media Komputindo.* Jakarta: *PT. Elex Media Komputindo.*

Satria, W. R. (2003). *Melihat dari dekat Windows Mobile 2003 for Smart device.* Jakarta: Majalah InfoKomputer.

Wicaksono, S. R. (2000). Tugas Akhir: *Proteksi Dokumen Word Dengan Kombinasi Enkripsi Veernam Cipher dan Shift Transposition*. Surabaya: STIKOM.

Ziegeler, C., Huber, B. (1999). *XML Serializer*. United States: Cocoon.