APLIKASI KRITPOGRAFI MODERN UNTUK PENGIRIMAN DATA TERAMANKAN

Husni Fahmi, Haret Faidah

PTIK-BPPT Gedung II lantai 21, MH Thamrin 8, Jakarta
{fahmi, haret}@inn.bppt.go.id

20 Mei 2010

Abstraksi

Kemajuan teknologi telah melahirkan sistem komputer yang terdistribusi yang menghubungkan banyak pengguna dan komputer pada wilayah yang berjauhan untuk dapat saling berinteraksi melalui jaringan komunikasi data. Seiring dengan kemajuan teknologi ini, pengamanan terhadap data baik yang berada di dalam komputer maupun yang melintas melalui jaringan menjadi suatu tantangan teknologi yang perlu disediakan solusinya. Pada makalah ini, kami memberikan pengantar tentang kriptografi yang merupakan dasar sistem pengamanan informasi dengan tujuan untuk menyediakan layanan dasar keamanan informasi yaitu *confidentiality*, *data integrity, authentication* dan *non-repudiation*. Untuk mencapai layanan dasar keamanan informasi, kami menjelaskan beberapa teknik kriptografi yaitu kriptografi simetris yaitu Advanced Encryption Standard (AES) dan kriptografi asimetris yaitu RSA serta message digest SHA-256. Ketiga teknik ini dan didukung dengan pengelolaan kunci dapat diterapkan pada bahasa pemrograman .NET untuk membangun aplikasi pengiriman data teramankan.

Kata kunci: computer security, cryptography, encryption, integrity, authentication, .NET application

1. PENDAHULUAN

Dengan munculnya komputer pada tahun 1960-an, maka data tidak lagi disimpan di lemari dalam bentuk tumpukan kertas tetapi disimpan di dalam komputer dalam bentuk pulsa-pulsa listrik. Kemampuan untuk mengamankan dan melindungi informasi yang ada di komputer menjadi sangat penting. Di era awal ini, di mana komputer masih berdiri sendiri dan belum saling berhubungan, pengamanan bisa dilakukan cukup dengan mengunci komputer atau menggunakan password. Tetapi kemudian komputer pun mengalami kemajuan teknologi di mana satu komputer tidak lagi berdiri sendiri tetapi bisa dihubungkan dengan beberapa buah komputer lain dan digunakan oleh beberapa orang bersama-sama. Pengamanan data menjadi lebih penting lagi karena kemungkinan penggunaan oleh orang yang tidak berwenang menjadi lebih besar.

Bersamaan dengan kemajuan teknologi komputer maka terjadi juga kemajuan yang besar dalam bidang teknologi jaringan. Kemajuan teknologi ini melahirkan suatu sistem komputer terdistribusi di mana ratusan pemakai dan komputer dalam suatu wilayah yang luas akan saling berinteraksi melalui falisitas jaringan. Koneksitas antar komputer dalam berkembang menjadi *Internet* (berasal dari kata *internetworking*) [COME06]. Dalam Internet,

ribuan orang dan komputer di seluruh dunia terhubung dalam satu dunia maya yang dikenal sebagai *cyber space*. Kemajuan teknologi di bidang komputer dan jaringan memberi kemudahan kepada pemakai komputer di seluruh dunia, tetapi kemajuan teknologi itu akan selalu diikuti dengan sisi buruk dari teknologi itu sendiri. Salah satunya adalah rawannya keamanan data sehingga menimbulkan tantangan dan tuntutan akan tersedianya suatu sistem pengamanan data yang sama canggihnya dengan kemajuan teknologi komputer itu sendiri. Bagaimana kita bisa melindungi informasi yang melintas ke belahan dunia yang lain melalui saluran komunikasi menjadi suatu tantangan yang harus dijawab oleh para teknokrat di bidang teknologi jaringan.

Ada beberapa cara untuk melakukan pengamanan data yang melalui suatu saluran seperti kriptografi, otentikasi, anti virus, firewall, dan intrusion detection system. Dalam naskah ini, kami akan menitikberatkan pada kriptografi untuk mengamankan data karena ia merupakan dasar dari sistem pengamanan informasi secara luas. Dalam kriptografi, data yang dikirimkan melalui jaringan akan disamarkan sedemikian rupa sehingga kalaupun data itu bisa dibaca maka tidak bisa dimengerti oleh pihak yang tidak berhak. Istilah kriptografi atau *Cryptography* berasal dari bahasa Yunani, yaitu *kryptós* atau "*hidden*" yang berarti tersembunyi, dan *gráphein* atau *"to write*" yang berarti menulis [Konh81]. Jadi kriptografi berarti *secret writing* atau penulisan rahasia.

Dalam pengiriman informasi lewat jaringan, seseorang bisa membuat ribuan duplikasi data tanpa bisa diketahui apakah itu asli atau tidak dan seseorang bisa melihat isi suatu berita yang dikirimkan melalui jaringan tanpa diketahui bahwa informasi sudah dibaca oleh yang tidak berhak. Di samping merahasiakan tulisan, kriptografi juga berperan dalam melindungi data dari pemalsuan (*authentication*) dan menjaga integritas data (*data integrity*) dari perubahan oleh pihak yang tidak berwenang. Dengan demikian, kriptografi adalah semua hal mengenai cara menghindari dan menemukan semua penipuan dan semua ke-tidakjujuran yang terjadi pada suatu pengiriman informasi.

Pada dasarnya, semua cara yang dianggap tradisional dan manual bisa diterapkan ke dalam bentuk pengamanan elektronik. Salah satu contoh pengamanan cara tradisional yang kemudian juga sangat berperanan sebagai cara pengamanan elektronik adalah tanda tangan. Dengan berubahnya informasi menjadi informasi elektronik, maka muncul juga tanda tangan di dalam bentuk elektronik. Tanda tangan di dalam bentuk elektronik ini dikenal sebagai *digital signature.* Ia menjadi dasar untuk banyak tujuan dari kriptografi modern misalnya untuk mencegah penyangkalan (*non-repudiation*), menjaga keaslian sumber data (*data origin authentication*), dan identifikasi pengirim (*identification*).

Kriptografi dikelompokkan dalam dua kelompok besar yaitu kriptografi klasik dan kriptografi modern. Kriptografi klasik adalah cara menyamarkan berita yang dilakukan dengan cara-cara manual sebelum adanya komputer, sedangkan kriptografi modern berkembang setelah ada komputer. Kriptografi modern adalah suatu metoda kriptografi yang menggunakan algoritma matematika dan suatu kunci.

Ada banyak tujuan dari kriptografi modern seperti *privacy* atau *confidentiality, data integrity, entity authentication* atau *identification, message authentication, signature, authorization, validation, access control, certification, timestamping, witnessing, receipt, confirmation, ownership, anonymity, non-repudiation* dan *revocation* [MOV96].

Dari semua tujuan pengamanan informasi yang disebutkan itu, empat hal bisa dianggap sebagai dasar dari yang lainnya yaitu *confidentiality*, *data integrity, authentication* dan *non-repudiation* [MOV96].

1. *Confidentiality* adalah suatu layanan yang digunakan untuk menjaga agar informasi hanya diketahui oleh pihak yang berwenang. *Secrecy* atau kerahasiaan adalah istilah yang sama dengan *confidentiality* dan *privacy*. Ada banyak cara untuk menjaga *confidentiality*, mulai dari cara pengamanan fisik sampai dengan penggunaan algoritma matematika.
2. *Data integrity* adalah layanan yang menjaga perubahan data oleh pihak yang tidak berhak. Untuk bisa menjamin integritas data maka seseorang harus bisa menemukan manipulasi data oleh pihak yang tidak berwenang. Manipulasi data ini termasuk penyisipan, penghapusan dan perubahan data.
3. *Authentication* adalah layanan yang berkaitan dengan peng-identifikasian. Identifikasi ini berlaku terhadap dua hal, yaitu kepada kedua pihak yang saling berkomunikasi dan juga terhadap berita yang dikirimkan. Karena itu, *authentication* biasanya dibagi dua yaitu *entity authentication* dan *data origin authentication. Data origin authentication* termasuk ke dalam *data integrity.*
4. *Non-repudiation* adalah suatu layanan yang menghindari satu pihak tidak mengakui suatu perjanjian atau suatu tindakan yang sudah dilakukan sebelumnya. Jika terjadi ketidaksesuaian karena salah satu pihak menyangkal suatu tindakan sebelumnya, maka ada suatu cara untuk membuktikan dan mengatasi keadaan itu.

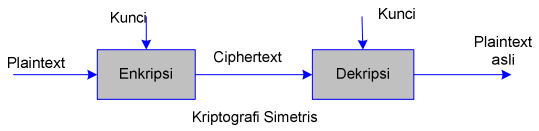
Tujuan dasar dari kriptografi modern adalah mencakup keempat hal yang disebutkan di atas. Keempat tujuan dasar ini bisa membentuk suatu kerangka kerja yang bisa digunakan untuk memenuhi tujuan yang lain.

Pada awalnya, kriptografi dinilai sebagai suatu seni yang dilakukan oleh orang-orang yang berusaha memenuhi kebutuhan pengamanan informasi di zamannya. Tetapi lambat laun, mulai sekitar tahun 1970-an, terjadi perubahan dari kriptografi sebagai suatu seni menjadi kriptografi sebagai suatu ilmu [MOV96]. Sekarang telah banyak dilakukan konferensi ilmiah berskala internasional tentang kriptografi. Ada juga suatu organisasi internasional yaitu *International Association for Cryptologic Research (IARC)* yang bertujuan untuk melakukan penelitian di bidang kriptografi.

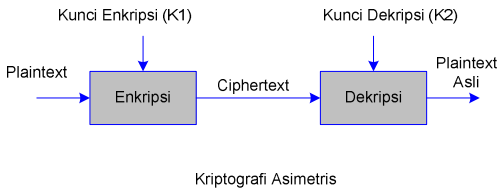
Seperti sudah disebutkan sebelumnya, bahwa kriptografi dikelompokkan dalam dua kelompok besar yaitu kriptografi klasik dan kriptografi modern. Kriptografi klasik adalah cara menyamarkan berita yang dilakukan dengan cara-cara manual sebelum adanya komputer, yaitu dengan cara-cara transposisi dan substitusi huruf. Ini dilakukan sejak jaman Romawi sampai dengan mulai digunakannya komputer dalam proses-proses kriptografi [HEBE]. Sedangkan kriptografi

modern, yang berkembang setelah adanya komputer, merupakan suatu metoda kriptografi yang menggunakan algoritma matematika dan suatu kunci. Informasi atau berita asli yang belum mengalami perubahan oleh proses kriptografi dikenal sebagai *plaintext*, dan berita hasil dari perubahan atau proses kriptografi dikenal sebagai *ciphertext*.

Ada dua jenis kriptografi modern yaitu kriptografi simetris dan kriptografi asimetris [JJ89]. Kriptografi simetris menggunakan kunci yang sama untuk enkripsi dan dekripsi seperti digambarkan pada Gambar 1. Kriptografi simetris disebut juga dengan kriptografi kunci rahasia (*secret key cryptography*). Contoh kriptografi simetris adalah *Data Encryption Standard* (DES) [FIPS99] dan *Advanced Encryption Standard* (AES) [FIPS01]. Sedangkan kriptografi asimetris menggunakan kunci yang berbeda untuk enkripsi dan dekripsi seperti digambarkan pada Gambar 2. Kriptografi asimetris disebut juga kriptografi kunci publik (*public key cryptography*). Contohnya adalah Rivest-Shamir-Adleman disingkat RSA [RFC 2313, RFC 2437].



Gambar 1. Kriptografi Simetris



Gambar 2. Kriptografi Asimetris

Sebagai ilustrasi penggunaan kriptografi modern di kehidupan nyata, maka tulisan ini akan dilengkapi dengan aplikasi yang mencakup penggunaan berbagai macam kriptografi modern yang sering digunakan. Ada penggunaan kriptografi simetris yaitu *AES*, pembuatan kunci privat dan kunci publik menggunakan *RSA*, kriptografi asimetris menggunakan *RSA*, dan pembuatan *hash code* menggunakan *SHA-256*.

2. Ulasan Teori Yang Digunakan

Sebelum melihat bagaimana aplikasi ini bekerja, maka akan diterangkan secara ringkas bagaimana algoritma-algoritma kriptografi modern bekerja.

2.1. Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) adalah kriptografi simetris berbasis block *cipher* dengan input *plaintext* sepanjang 128 bit dan kunci bervariasi antara 128, 192 dan 256 bit. Keluarannya berupa *ciphertext* dengan panjang block 128 bit [FIPS01].

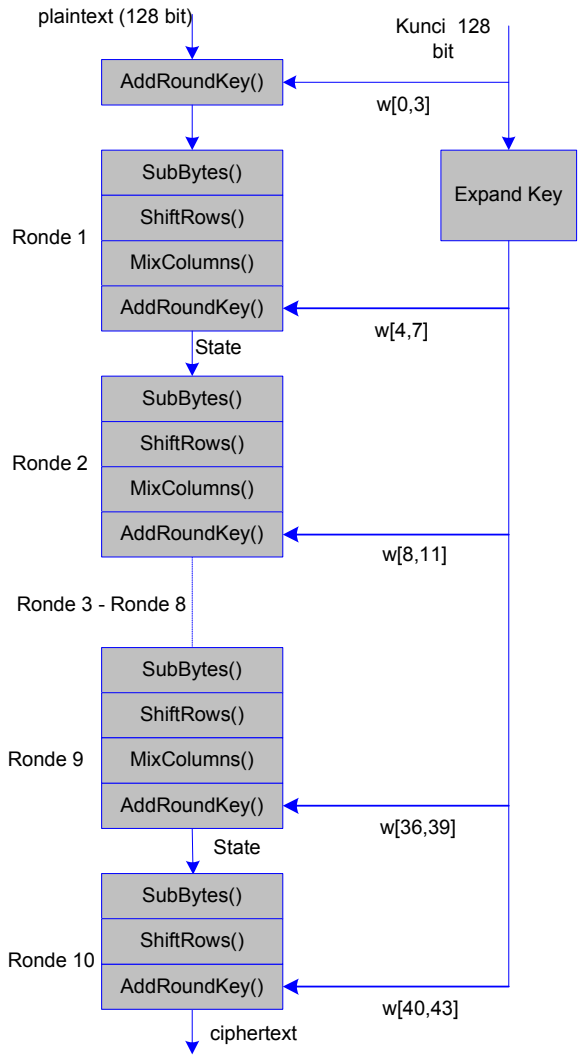
2.1.1. Enkripsi AES

Proses Enkripsi AES yang digambarkan pada Gambar 3 adalah sebagai berikut:

1. Pada awal enkripsi, input plaintext 128 bit akan disalinkan ke suatu array yang diberi nama *state array*.
2. Proses enkripsi dimulai dengan suatu proses yang disebut *AddRoundKey().*
3. Untuk kunci *cipher* 128 bit, jumlah ronde adalah 10.
4. Proses selanjutnya adalah 9 ronde yang masing-masing ronde terdiri dari urut-urutan empat macam proses yaitu *SubBytes(),*

ShiftRows(), MixColumns() dan *AddRoundKey().*

5. Diakhiri dengan ronde ke 10 yang hanya terdiri dari 3 proses yaitu *SubBytes(), ShiftRows()* dan *AddRoundKey().*
6. Dengan proses *Expand Key* atau perpanjangan kunci, kunci yang terdiri dari 4 word (1 word terdiri dari 32 bit) akan diperpanjang menjadi suatu array yang terdiri dari 44 word yang masing-masing word terdiri dari 32 bit atau 4 byte. Kunci yang asli digunakan pada awal enkripsi, kemudian setiap 4 buah word hasil perpanjangan kunci akan digunakan pada ronde 1 sampai dengan ronde 10.
7. Setiap Ronde terdiri dari empat proses, yang terdiri dari satu operasi permutasi dan tiga operasi substitusi yaitu:
   a. *SubBytes(),* yang menggunakan suatu *AES S-Box* untuk melakukan substitusi *byte-per-byte*.
   b. *ShiftRows(),* yang melakukan suatu permutasi sederhana dengan cara *left shift* atau geser kiri.
   c. *MixColumns()* melakukan suatu operasi aritmatika dalam GF (2^8).
   d. *AddRoundKey()* melakukan operasi XOR antara hasil dari tiga proses sebelumnya dengan 4 word kunci Ronde.

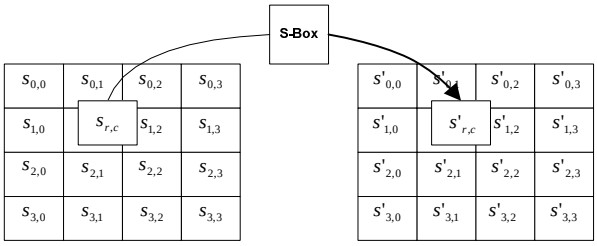


Gambar 3. Enkripsi AES

2.1.1.1. *SubBytes ()*

SubBytes() ini menggunakan matriks 16x16 yang disebut *AES S-Box* yang berisi permutasi semua 256 kemungkinan dari kombinasi 8-bit. Setiap sel pada matriks berisi 1 byte (2 digit hexadecimal atau *nibble*). Setiap byte *State* akan ditransformasikan ke nilai lain di dalam matriks.

Gambar berikut memperlihatkan konversi oleh suatu *S-Box*



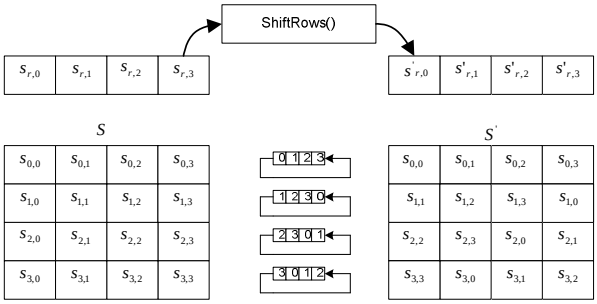
Gambar 4. Konversi *S-Box*

2.1.1.2. *ShiftRows()*

Proses *ShiftRows()* ini adalah proses yang sangat sederhana. Baris ke-1 sampai baris ke-3 dari *State* akan digeser ke kiri. Baris pertama tidak mengalami geser kiri. *ShiftRows()* adalah sebagai berikut: [note: please check the formula]

$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb}$ untuk $0 \langle r \langle 4$ dan $0 \le c \langle Nb,$

di mana *shift(r,Nb)* tergantung ke pada nomor baris *r*. *Nb* adalah panjang blok (4). Jadi *shift(1,4)=1; shift(2,4)=2; shift(3,4)=3* .



Gambar 5. ShiftRow()

2.1.1.3. *MixColumns()*

Transformasi menggunakan *MixColumns()* adalah proses ketiga dalam satu Ronde enkripsi *AES*. Di sini kolom-kolom pada array *State* akan diperlakukan sebagai suatu polinomial yang berada dalam GF$\left(2^8\right)$ dan akan dikalikan dengan modulo $x^4 + 1$, dengan suatu polinomial tertentu:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Kalau kita anggap bahwa $s'(x) = a(x) \otimes s(x)$, maka

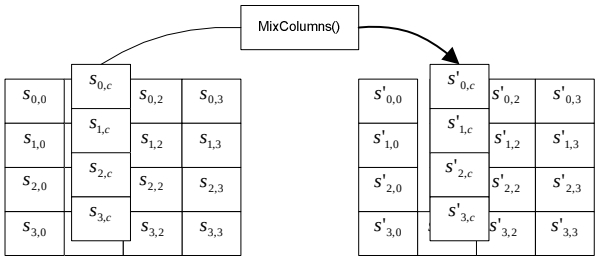
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Untuk $0 \le c \langle Nb$.

Sebagai hasil perkalian, maka 4 byte dalam setiap kolom pada *State* array akan berubah sebagai berikut:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

Gambar berikut memperlihatkan transformasi *MixColumns():*



Gambar 6. Transformasi *MixColumns()*

2.1.1.4. *AddRoundKey()*

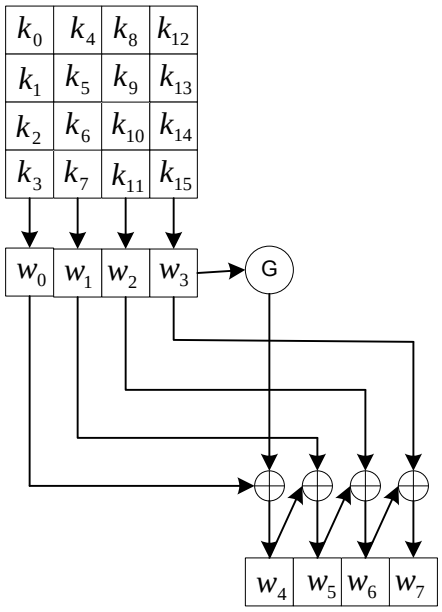
Pada proses *AddRoundKey()* maka 128 bit hasil *State* akan di-XOR-kan dengan Kunci Ronde, yaitu kunci hasil dari proses *Expand Key*. Pada awal enkripsi, 128 bit plaintext akan di-XOR-kan dengan 128 bit kunci yang asli. Kemudian 128 bit plaintext akan mengalami proses-proses:

SubBytes(), ShiftRows() dan *MixColumns().* Pada proses *AddRoundKey(),* 128 bit yang sudah melalui ketiga proses tersebut akan di-XOR-kan dengan Kunci Ronde hasil *Expand Key* yang pertama. Hasil *AddRoundKey()* ini adalah *state* pada Ronde 1. *State* 1 ini akan mengalami ketiga proses tersebut kembali. Pada *AddRoundKey()* yang berikut, maka 128 bit yang sudah mengalami perubahan pada ketiga proses tersebut kembali akan di-XOR-kan dengan Kunci Ronde hasil *Expand Key* kedua dan seterusnya sampai Ronde 9.

Pada Ronde terakhir atau Ronde 10, maka hanya tiga proses yang dilakukan yaitu *SubBytes(), ShiftRows()* dan *AddRoundKey().* Hasilnya dari Ronde 10 adalah *State* 10. Ini akan disalinkan ke array *output* sebagai *ciphertext*.

2.1.1.5. *Expand Key*

Expand Key adalah proses untuk membuat kunci yang panjangnya 128 bit atau 4 word menjadi kunci sebanyak 44 word. 4 word pertama, yaitu bit-bit kunci yang asli sebanyak 128 bit digunakan pada awal enkripsi. Kemudian 40 word berikutnya, hasil dari *Expand Key* akan digunakan pada setiap Ronde. Ada 10 Ronde yang masing-masing menggunakan 4 word [Sta03].



Gambar 7. ExpandKey

1. Perpanjangan kunci (*key expansion*) dimulai dengan memasukkan 4 buah word, yaitu 128 bit kunci yang asli ke 4 posisi pertama dari array 44 word yaitu w_0, w_1, w_2 dan w_3.
2. Sisa array 40 word, dari w_4 sampai dengan w_{43}, akan diisi setiap kali dengan 4 buah word baru.
3. Setiap word ke 4, yaitu w_3, w_7, w_{11} dan seterusnya sampai dengan w_{39}, akan diproses oleh suatu fungsi.
4. Setiap word berikutnya akan diperoleh dengan cara:

   - jika $i \equiv 0 \bmod 4$, maka

   $$w[i] = w[i-4] \oplus \text{ output fungsi G,}$$

   - jika $i \neq 0 \bmod 4$, maka

   $$w[i] = w[i-4] \oplus w[i-1]$$

5. Proses ini akan terus berulang sampai diperoleh w_{43}.

Cara kerja Fungsi G:

Ada 3 proses pada fungsi G, yaitu rotasi kiri (*RotWord*), substitusi Word (*SubWord*) dan XOR.

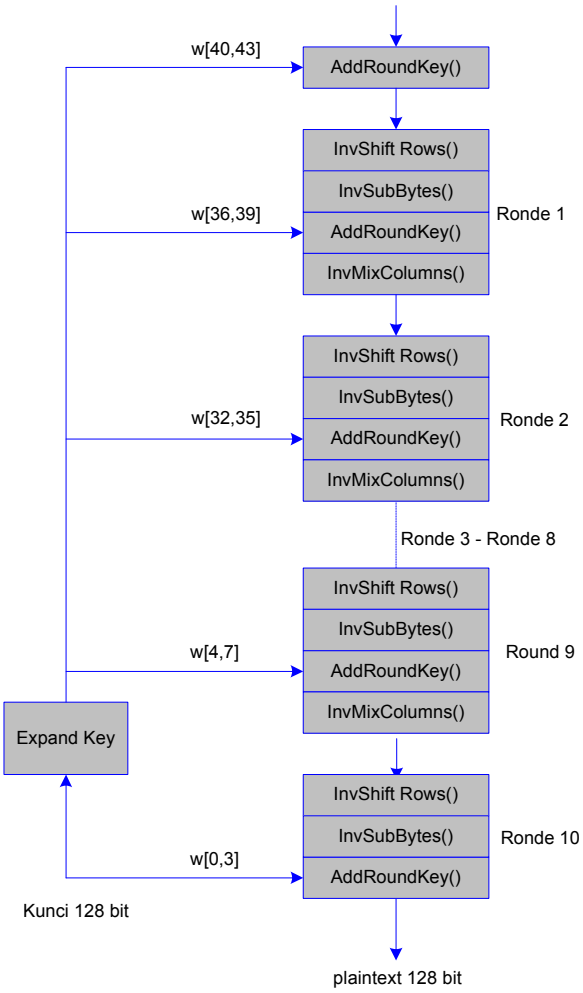
1. *RotWord*: Word ke-4 (w_3) akan mengalami rotasi kiri sebanyak 1 byte, jadi word yang terdiri dari: $[k_{12}, k_{13}, k_{14}, k_{15}]$ akan menjadi $w_3^{'} = [k_{13}, k_{14}, k_{15}, k_{12}]$.
2. *SubWord*: Dilakukan byte substitution terhadap setiap byte pada setiap word sesuai AES S-Box, $w_3^{'}$ akan menjadi $w_3^{''}$
3. Dilakukan operasi XOR antara $w_3^{''}$ dengan suatu nilai yaitu $Rcon[j]$.
4. $Rcon[j]$ adalah suatu konstanta yang 3 byte kanannya adalah 0, dan byte terkiri adalah selalu tetap sesuai tabel berikut:

Table 1. Tabel Konstanta $Rcon[j]$

Ronde ke	1	2	3	4	5	6	7	8	9	10
Rcon[j]	01	02	04	08	10	20	40	80	1B	36

2.1.2. Dekripsi *AES*

Urutan proses dekripsi AES tidak merupakan kebalikan dari enkripsi. Proses Dekripsi *AES* ini digambarkan pada Gambar 4. Ada proses yang dipertukarkan urutannya, walaupun penggunaan kuncinya sama. Jika urut-urutan proses pada enkripsi adalah *SubBytes(), Shift Rows() , Mix Columns()* dan *AddRoundKey()* , maka urut-urutan proses pada dekripsi adalah *InvShiftRows(), InvSubBytes(), InvAddRoundKey()* dan *InvMixColumns().* Keempat proses tersebut adalah masing-masing invers dari empat proses pada enkripsi tetapi urutan *InvShiftRows()* dan *InvSubBytes()* dipertukarkan dan *InvAddRoundKey()* dan *InvMixColumns()* juga dipertukarkan.



Gambar 8. Dekrispi AES

2.1.2.1. *InvShiftRows()*

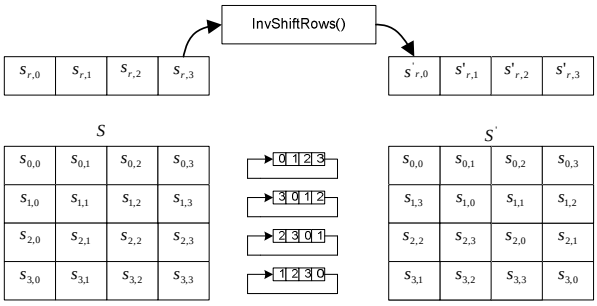
Proses ini adalah proses invers dari *ShiftRows()* pada enkripsi. Jika pada enkripsi dilakukan geser kiri maka pada dekripsi atau *InvShitfRows()* maka yang dilakukan adalah geser kanan. Baris pertama tidak mengalami geser kanan. Baris selanjutnya mengalami geser kanan dengan transformasi sebagai berikut:

[note: periksa formula]

$s_{r,(c+shift(r,Nb))\bmod Nb} = s_{r,c}$ untuk $0 \langle r \langle 4$ dan $0 \le c \langle Nb$

r adalah baris (dari 0 sampai 3)

c adalah kolom (dari 0 sampai N_b, $N_b = 4$)



Gambar 9. *InvShiftRows()*

2.1.2.2. *InvSubBytes()*

Ini adalah invers dari *SubBytes()* yang dilakukan pada enkripsi. Matriks yang digunakan adalah invers dari *S-Box* yang digunakan pada enkripsi.

2.1.2.3. *InvMixColumns()*

InvMixColumns() adalah invers dari proses *MixColumns()* pada enkripsi. *InvMixColumns()* bekerja pada kolom-per-kolom dari *State array*, dan memperlakukan setiap kolom sebagai suatu polinomial pangkat 3. Polinomial dianggap ada pada GF$\left(2^8\right)$ dikalikan dengan modulo $x^4 + 1$, dengan polinomial tertentu $a^{-1}(x)$ berikut:

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

Jika $s'(x) = a^{-1}(x) \otimes s(x)$, 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 $0 \le c \langle Nb$.

Dari hasil perkalian ini, 4 byte di dalam kolom akan diubah sebagai berikut :

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

2.1.2.4. *InvAddRondeKey()*

InvAddRoundKey() adalah sama dengan *AddRoundKey()*

2.1.3. *Equivalent Invers Cipher*

Seperti sudah dijelaskan sebelumnya bahwa urut-urutan proses pada dekripsi *AES* tidak tepat sama dengan enkripsi. Ada proses yang dipertukarkan urutannya. Urut-urutan proses pada dekripsi adalah *InvShiftRows(), InvSubBytes(), InvAddRoundKey()* dan *InvMixColumns()* sedangkan urut-urutan pada enkripsi adalah *SubBytes(), ShiftRows(), MixColumns()* dan *AddRoundKey().* Ini menimbulkan konsekuensi bahwa suatu aplikasi yang membutuhkan enkripsi dan dekripsi harus menggunakan dua perangkat lunak atau peranti tegar (*firmware*) yang berbeda pada satu aplikasi. Akhirnya dikembangkan suatu metode dekripsi *AES* yang urut-urutannya sama dengan enkripsi yaitu: *InvSubBytes(), InvShift Rows(), InvMixColumns()* dan *AddRoundKey().*

Equivalent Inverse Cipher atau Ekuivalen Dekripsi AES bisa dilakukan karena :

1. Proses *SubBytes()* dan *ShiftRows()* bisa saling dipertukarkan. Jadi proses *SubBytes()* yang diikuti dengan *ShiftRows()* sama dengan proses *ShiftRows()* yang diikuti dengan proses *SubBytes().* Begitu juga dengan inversnya, proses *InvShiftRows(),* yang diikuti dengan proses *InvSubBytes()* sama dengan

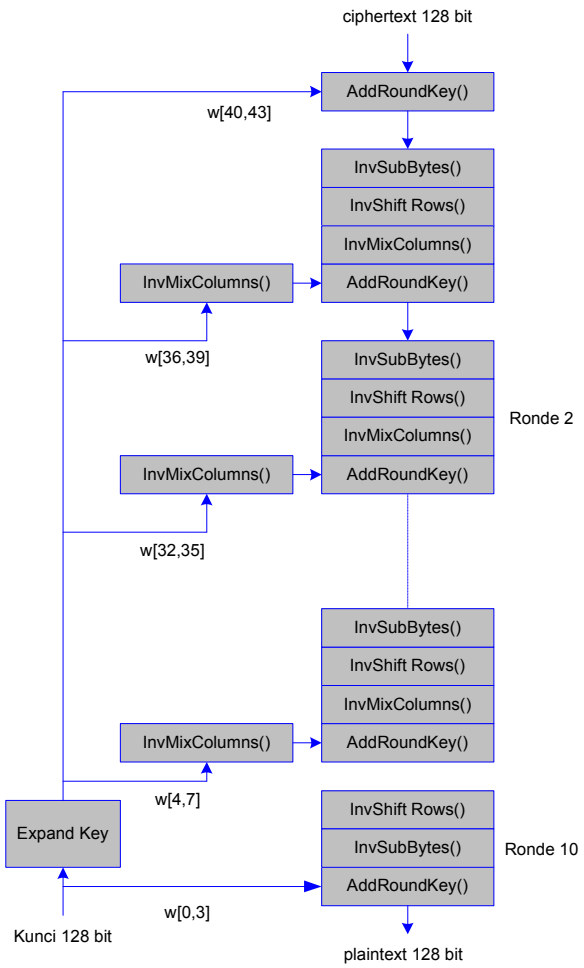
proses *InvSubBytes()* yang diikuti dengan proses *InvShiftRows(),*

2. Proses *MixColumns()* dan *InvMixColumns()* linier terhadap kolom input, yang berarti: *InvMixColumns(state XOR RondeKey) = InvMixColumns(state) XOR InvMixColumns(RoundKey)*

Kedua sifat ini memungkinkan urutan proses *InvSubBytes()* dan *InvShiftRows()* untuk dipertukarkan. Juga urutan *AddRoundKey()* dan *InvMixColumns()* bisa dipertukarkan, jika w_4 sampai dengan w_{39} dari hasil *Expand Key* dimodifikasi menggunakan *InvMixColumns().*

Equivalen Dekripsi AES dilakukan dengan menukarkan urutan *InvSubBytes()* dan *InvShiftRows()* dan menukarkan urutan *AddRoundKey()* dan *InvMixColumns()* setelah memodifikasi kunci untuk Ronde 1 sampai Ronde N_r -1 menggunakan *InvMixColumns.*

Equivalen dekripsi *AES* memberi struktur yang lebih efisien dari *Inverse Cipher* yang sudah diterangkan sebelumnya.



Gambar 10. Skema Ekuivalen Dekripsi AES

2.2. Pembuatan kunci dan kriptografi asimetris menggunakan RSA

RSA adalah singkatan dari Rivest-Shamir-Adleman yaitu nama tiga profesor MIT yang mengembangkannya. RSA adalah kriptografi asimetris yang paling banyak digunakan [note: please add a reference]. Kekuatan RSA dalam melakukan pengamanan data didapatkan dari sepasang kunci yang diperoleh dari fungsi dua buah bilangan prima. Sepasang kunci ini adalah hasil dari fungsi dua buah bilangan prima yang sangat besar, yang panjangnya 100, 200 digit atau mungkin lebih besar. Dari perhitungan matematika menggunakan kedua buah bilangan

prima ini, akan didapatkan kunci publik dan kunci privat. Sesudah digunakan untuk menghitung kunci, kedua bilangan prima boleh dihapus, tetapi kerahasiaannya harus tetap dijaga. Dua buah teori bilangan yang mempunyai peranan penting dalam kriptografi asimetris termasuk RSA adalah Teorema Fermat dan Teorema Euler [Sta03].

Cara membuat kunci RSA:

1. Pilih dua buah bilangan prima yang besar secara acak (random) , misalnya p dan q.
2. Kalikan ke dua bilangan tersebut, n = pq.
3. Hitung totient $\phi(n)$=(p-1)(q-1).
4. Pilih suatu integer e, di mana 1<e< $\phi(n)$, dan e relatif prima terhadap m.
5. Hitung d, di mana de $\equiv$ 1(mod $\phi(n)$).
6. Bisa juga dikatakan d= e^{-1} (mod $\phi(n)$).
7. Kunci publik KU{n,e}.
8. Kunci privat KR{n,d}.

Contoh Pembuatan Kunci

Untuk kemudahan, contoh ini dibuat dengan bilangan-bilangan kecil

1. Pilih dua buah bilangan prima, misalnya p=61 dan q=53.
2. Hitung n = pq = 61 x 53 = 3233.
3. Hitung totient $\phi(n)$= (p-1)(q-1) = 60 x 52 = 3120.
4. Pilih suatu integer e, di mana 1<e< $\phi(n)$, dan e relatif prima terhadap $\phi(n)$

   Kita pilih e = 17.
5. Hitung d, dimana de $\equiv$ 1(mod $\phi(n)$). Setelah dihitung dengan algoritma extended Euclid maka akan didapatkan d = 2753.
6. Didapatkan kunci publik KU{3233,17} dan kunci privat KR{3233,2753}.

2.2. Enkripsi dan Dekripsi RSA

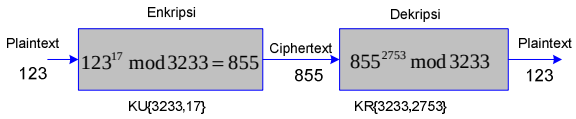
- Jika *C* adalah ciphertext dan *P* adalah plaintext (*P<n),* maka enkripsi dilakukan sebagai berikut:

$$C = P^e \bmod n,$$

- dan dekripsi dilakukan sebagai berikut :

$$P = C^d \bmod n$$

Jika kita melakukan enkripsi terhadap 123, maka



Gambar 11. RSA

2.3. Pembuatan message digest menggunakan SHA-256

Tanda tangan elektronik atau *digital signature* menjadi dasar untuk banyak tujuan dari kriptografi modern misalnya *non-repudiation, data origin authentication,* dan *identification.* Salah satu cara untuk melakukan *digital signature* adalah dengan menggunakan algoritma *hash*. Pesan yang akan dikirimkan akan menjadi *input* dari algoritma *hash* dan akan menghasilkan suatu nilai yang dikenal sebagai *message digest* atau *hash code*. *Hash code* ini akan dienkripsi menggunakan kunci privat, dan hasil enkripsi ini yang akan menjadi *digital signature* seseorang. Algoritma *hash* ini menggunakan pesan yang panjangnya bervariasi (M) sebagai *input* dan akan menghasilkan *output* yaitu *hash code* h, di mana h adalah fungsi dari H(M). Algoritma *hash* tidak menggunakan kunci seperti pada enkripsi/dekripsi, dan hanya menggunakan pesan sebagai *input*. Algoritma *hash* tidak mengenal algoritma *invers*, jadi tidak

ada cara untuk mengembalikan *hash code* ke dalam bentuk *input*nya kembali [FIPS02].

Hash code h dibuat dengan menggunakan suatu fungsi H yang mengubah *input* pesan M sebagai berikut :

$$h = H(M)$$

Fungsi *hash* yang berkembang dan dianggap cukup handal adalah dari kelompok *SHA (Secure Hash Algorithm)* yang dipublikasikan melalui *FIPS PUB 180.* Sekarang, algoritma ini dikenal sebagai *SHA-0*. Kemudian muncul *SHA-1* yang dipublikasikan pada tahun1995 melalui *FIPS PUB 180-1.* Pada tahun 2002, NIST melengkapi dengan mempublikasikan *FIPS 180-2.* Publikasi ini menambahkan tiga buah algoritma hash dalam kelompok *SHA* yaitu *SHA-256, SHA-384* dan *SHA-512,* yang mempunyai *hash code* yang lebih panjang dari *SHA-1. SHA-256, SHA-384* dan *SHA-512* berturut-turut memiliki panjang *hash code* 256, 384 dan 512 bit. Aplikasi dalam paper ini menggunakan *SHA-256*.

2.3.1. Operasi SHA-256

Setiap algoritma di dalam *SHA* menggunakan 2 tahap yaitu tahap *preprocessing* dan tahap penghitungan *hash code*. Tahap *preprocessing* melakukan penambahan bit-bit pada pesan sehingga membentuk suatu *input text*, pengelompokkan *input* dalam blok-blok yang terdiri dari m bit, dan pemberian nilai awal (*initial value*) untuk digunakan dalam penghitungan *hash*. Penghitungan *hash* akan membuat suatu skedul pesan (*message schedule*) dari *input* dan menggunakan skedul pesan tersebut bersama-sama dengan konstanta, dan word di dalam suatu fungsi, untuk secara berulang menghasilkan suatu nilai *hash*. Nilai *hash* terakhir yang dihasilkan dari perulangan fungsi adalah *hash code* yang dihasilkan.

2.3.2. Fungsi yang Digunakan

Algoritma *SHA-256* menggunakan fungsi $Ch(x, y, z)$ dan fungsi $Maj(x, y, z)$. *SHA-256* menggunakan enam buah fungsi logik di mana setiap fungsi bekerja pada suatu word yang terdiri dari 32 bit dan direpresentasikan sebagai x, y dan z. Hasil dari setiap fungsi adalah juga suatu word 32 bit.

Keenam fungsi tersebut adalah:

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$

$$\sigma_0^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

2.3.3. Konstanta yang digunakan

SHA-256 menggunakan 64 buah konstanta yang adalah word yang terdiri dari 32 bit. Di dalam hexadesimal, word-word ini adalah:

428a2f98	71374491	b5c0fbcf	e9b5dba5
3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3
72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc
2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7
c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13
650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3

d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5
391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208
90befffa	a4506ceb	bef9a3f7	c67178f2

2.3.4. Tahap *preprocessing*

Panjang dari pesan yang akan jadi *input* fungsi *hash* harus kelipatan 512. Jika panjang pesan adalah l bit, maka setelah bit ke l, tambahkan dengan sebuah bit '1' , kemudian sisanya tambahkan bit '0' sebanyak k bit sehingga memenuhi $l+1+k \equiv 448 \bmod 512$. Kemudian tambahkan lagi 64 bit yang merepresentaskan panjang pesan l direpresentasikan dalam 64 bit. Jadi panjang dari input akan menjadi tepat kelipatan 512. Input akan dikelompokan dalam N buah blok yang terdiri dari 512 bit. Blok-blok tersebut adalah $M^{(1)}, M^{(2)}, M^{(3)}....., M^{(N)}$. Blok 512 bit bisa juga dinyatakan sebagai 16 buah word dan tiap word terdiri dari 32 bit, maka 32 bit pertama dalam blok ke-i bisa dituliskan sebagai $M_0^{(i)}$, 32 bit kedua bisa dituliskan $M_1^{(i)}$ sampai dengan 32 bit terakhir sebagai $M_{15}^{(i)}$. Sebelum penghitungan *hash* dilakukan, maka harus diberikan *Initial Hash Value,* $H^{(0)}$.

Pada SHA-256, digunakan delapan word sebagai berikut:

$$H_0^{(0)} = 6a09e667$$

$$H_1^{(0)} = bb67ae85$$

$$H_2^{(0)} = 3c6ef372$$

$$H_3^{(0)} = a54ff53a$$

$$H_4^{(0)} = 510e527f$$

$$H_5^{(0)} = 9b05688c$$

$$H_6^{(0)} = 1f83d9ab$$

$$H_7^{(0)} = 5be0cd19$$

SHA-256 digunakan untuk membuat suatu *hash code* dari suatu pesan M, yang mempunyai panjang pesan l bit, di mana $0 \le l \langle 2^{64}$. Algoritma *SHA-256* menggunakan:

1. *Message schedule* yang terdiri dari 64 buah word. Satu word terdiri dari 32 bit. *Message schedule* ini diberi label $W_0, W_1, W_2.....W_{62}, W_{63}$.
2. Delapan buah variabel yang didefinisikan sebagai variabel a, b, c, d, e, g dan h.
3. Suatu nilai *hash* yang terdiri dari delapan word. Setiap word diberi label $H_0^{(i)}, H_1^{(i)}, H_2^{(i)}, H_3^{(i)}.....H_7^{(i)}$. *Hash value* ini akan berisi *initial hash value* $H^{(0)}$. Setelah satu proses akan digantikan dengan *hash value* $H^{(i)}$ dan seterusnya diakhiri dengan *hash value* yang final yaitu $H^{(N)}$.
4. *SHA-256* juga menggunakan dua buah word yaitu T_1 dan T_2.

Hasil akhir adalah *hash code* yang mempunyai panjang 256 bit.

2.3.5. Penghitungan *hash value S-256*

SHA-256 menggunakan fungsi dan konstanta yang sudah disebutkan sebelumnya. Penjumlahan + adalah penjumlahan modulo 2^{32}. Setelah tahap *preprocessing* selesai, maka setiap blok pesan $M^{(1)}, M^{(2)}, M^{(3)}....., M^{(N)}$ akan diproses secara berurutan menggunakan langkah berikut:

Untuk $i = 1$ sampai N :

{

1. siapkan *message schedule* $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

2. variabel a, b, c, d, e, g dan h diberi nilai awal, dengan *hash value* ke-i berikut:

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

3. untuk $t = 0$ sampai 63:

{

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

}

4. Hitung *hash value* ke-i $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

}

Setelah menjalankan penghitungan pada butir 1 sampai dengan butir 4 sebanyak N kali sampai dengan blok input yang terakhir, maka akan dihasilkan *hash value* atau *message digest* yang panjangnya 256 bit dari pesan M.

Hash code tersebut adalah:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}.$$

2.4. Public Key Authority

Baik pada kriptografi simetris maupun kriptografi asimetris ada penggunaan kunci yang harus diketahui oleh kedua belah pihak yang saling berkomunikasi. Pada kriptografi simetris, ada sebuah kunci rahasia yang digunakan untuk enkripsi dan dekripsi, dan harus diketahui oleh kedua pihak. Pada kriptografi asimetris, ada kunci privat yang dipegang hanya oleh pemiliknya dan kunci publik yang harus diketahui oleh pasangannya. *Key Management* atau manajemen kunci dan *Key Exchange* atau Pertukaran Kunci adalah cara bagaimana mengatur agar kunci itu bisa sampai ke pihak penerima dengan aman.

Key exchange atau pertukaran kunci diperlukan untuk mempublikasikan atau mendistribusikan kunci rahasia pada kriptografi simetris dan kunci publik pada kriptografi asimetris ke pada pihak yang memang membutuhkan menggunakan kunci tersebut.

Ada beberapa macam cara mempublikasikan kunci publik :

- mempublikasikan kunci publik secara terbuka
- mempublikasikan kunci publik melalui directory tertentu
- menggunakan *public key authority*
- menggunakan *public key certificate*

Aplikasi ini menggunakan *Public Key Authority Server* sebagai cara pertukaran kunci. Semua kunci publik disimpan di suatu file atau directory dan dikelola oleh seorang penanggungjawab atau seorang pengelola. Permintaan seseorang untuk mengetahui kunci publik seseorang dilakukan dengan menggunakan kunci publik dari pengelola directory. Peminta mengirimkan permohonan yang dienkripsi dengan menggunakan kunci publik pengelola kepada pengelola. Pengelola akan melakukan dekripsi permohohon dengan menggunakan kunci privatnya sendiri. Informasi yang dibutuhkan yaitu kunci publik yang diminta beserta informasi lain yang dibutuhkan akan dikirimkan oleh pengelola dengan menggunakan kunci privat. Pemohon akan bisa membaca dengan menggunakan kunci publik pengelola. Jelas ini lebih aman karena sudah ada peranan kriptografi yaitu adanya enkripsi dan dekripsi. (www.microsoft.com/technet,Sta03)

Pada aplikasi ini, sebuah komputer dijadikan *Public Key Authority Server* yaitu tempat komputer lain menyimpan kunci publiknya. Lihat Gambar 12.

2.5. Pengkodean Basis 64

Pengkodean basis 64 adalah suatu cara pengkodean untuk bisa mengirimkan suatu file melalui Internet mail. Mail hanya didesain untuk mengirimkan format *ASCII* sehingga semua macam data harus bisa dikonversi menjadi format *ASCII. Base64* adalah suatu cara untuk mengkonversi segala macam format data menjadi *ASCII*. Dasarnya adalah mengubah pengkodean 8 bit menjadi pengkodean 6 bit. 3 buah karakter yang tersusun dari 8 bit diubah menjadi 4 buah karakter yang tersusun dari 6 bit, dan dikirimkan melalui Internet dengan pengkodean 6 bit.

3. Kriptografi di dalam Aplikasi .NET

Perkembangan dan penggunaan yang luas dari aplikasi *e-commerce* saat ini menuntut suatu algoritma pengamanan untuk melindungi keamanan dan kerahasiaan data yang digunakan. Microsoft telah memperkenalkan sistem pengamanan data pertama kali pada tahun 1996 dengan nama *Cryptography API (Crypto API).* Kebutuhan pengamanan data yang terus berkembang mendorong Microsoft memperkenalkan sistem layanan kriptografi seperti *System.Security.Cryptography namespace* dari *Microsoft .NET common langauge runtime (CLR).* Cryptography Namespace ini memungkinkan suatu program mengakses suatu layanan kriptografi sehingga bisa melengkapi suatu aplikasi dengan kemampuan enkripsi dekripsi data, menjamin integritas data, melakukan *digital signature* dan *certificates*.

3.1. Cryptography Namespace

Fungsi utama dari Cryptography namespace ini adalah untuk menyediakan *classes* yang menerapkan algoritma pengamanan data seperti enkripsi dan pembuatan *hash*. Pada hirarkinya yang tertinggi, *Cryptography Namespace* bisa dikelompokkan dalam empat bagian. Lihat Gambar 12.

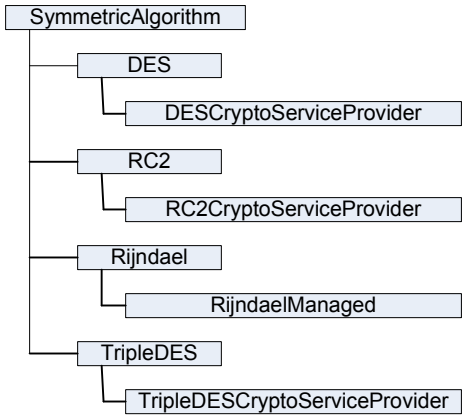
Division	Description
Encryption Algorithm	A set of classes used to implement both symmetric and asymmetric encryption and hash algorithms
Helper Classes	Classes used to generate random numbers, perform conversations, interact with Crypto API store, and actually perform the encryption using a stream-based model
X.509 Certificates	Classes defined in the System.Security.Cryptography.X509Certificates namespace used to represent digital certifictes
XML Digital Sugnatures	Classes defined in the System.Cryptography.Xml namespace used to represent digital signatures in XML documents

Gambar 12. Komponen Kriptografi (dari //msdn.microsoft.com/msdnmag)

Algoritma ini diterapkan dengan menggunakan pola yang bisa diperluas yang terdiri dari dua tingkatan (*inheritance*). Tingkatan pertama adalah suatu *abstract base class* seperti *AsymmetricAlgorithm* atau *HashAlgorithm* yang menentukan tipe algoritma. Tingkatan kedua diambil dari tingkatan di atasnya dan melengkapi dengan *public interface* untuk algoritma tersebut.

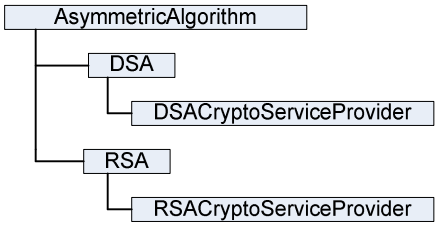
3.2. Algoritma Enkripsi

Seperti terlihat pada Gambar 12, salah satu algoritma enkripsi yang ada pada *Cryptography Namespace* adalah *symmetric cryptographic algorithm* atau algoritma kriptografi simetris. Dinamakan demikian karena digunakan satu kunci yang sama untuk enkripsi dan dekripsi. S*ymmetric cryptographic algorithm* ini mencakup *Data Encryption Standard (DES), RC2, Rijndael* dan *Triple Data Encryption Standard (TripleDES).*



Gambar 13. *Symmetric Algorithm* (dari //msdn.microsoft.com/msdnmag)

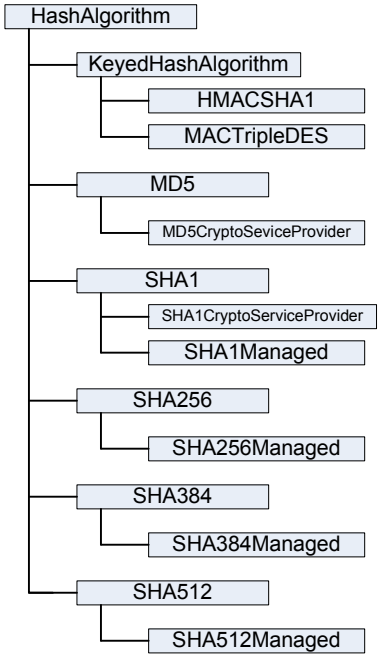
Tipe algoritma pengamanan data kedua dikenal sebagai *asymmetric algorithm* atau algoritma asimetris. Algoritma ini mencakup *Digital Signature Algorithm (DSA*) dan *RSA* yang mendapatkan nama dari pengembangnya yaitu Ron Rivest, Adi Shamir dan Len Adleman [JJ89]. A*symmetric algorithm* menggunakan sepasang kunci, yaitu sebuah kunci privat dan sebuah kunci publik.



Gambar 14. *Asymmetric Algorithm* (dari //msdn.microsoft.com/msdnmag)

Tipe algoritma pengamanan data ketiga adalah *hash algorithm*. Algoritma *hash* ini menghitung suatu nilai yang dikenal sebagai *hash code* atau *digest* dari data yang akan dikirimkan. *Hash code* ini akan dikirimkan bersamaan dengan data. Pada penerima, *hash code* ini bisa dihitung lagi. Jika kedua *hash code* ini sama berarti tidak ada perubahan data, jika berbeda berarti ada perubahan data. Algoritma hash ini juga

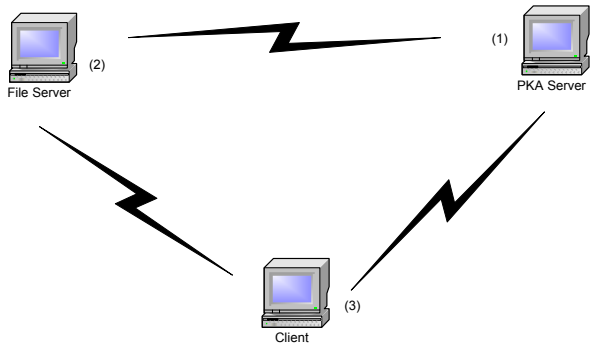
digunakan sebagai dasar untuk pembuatan *digital signature. Cryptography namespace* berisi class yang diberi nama *HashAlgorithm* yang berisi algoritma MD5, SHA1, SHA256, SHA384 dan SHA512. Algoritma MD5 menghasilkan *hash code* sepanjang 128 bit dan SHA1 menghasilkan *hash code* sepanjang 160 bit. Algoritma yang lain menghasilkan *hash code* yang mempunyai panjang sesuai namanya. Bertambah panjang suatu *hash code* maka bertambah aman algoritma tersebut dan akan lebih kebal terhadap *bruce-force attack*.



Gambar 15. *Hash Algorithm* (dari //msdn.microsoft.com/msdnmag)

4. Rancangan dan Implementasi

Sebagai ilustrasi penggunaan kriptografi modern di kehidupan nyata, maka tulisan ini akan dilengkapi dengan aplikasi untuk mengirimkan suatu file melalui jaringan yang mencakup penggunaan berbagai macam kriptografi. Ada penggunaan kriptografi simetris yaitu *AES*, pembuatan kunci privat dan kunci publik menggunakan *RSA*, kriptografi asimetris yaitu *RSA*, penggunaan *digital signature* dan pembuatan *hash code* menggunakan *SHA-256*. Pertukaran kunci dilakukan dengan membuat salah satu komputer menjadi *Public Key Authority Server.*



Gambar 16. Konfigurasi *PKA Server – File Server - Client*

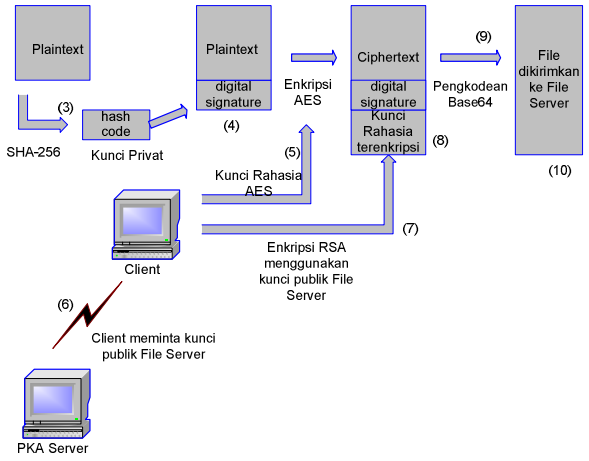
Pada gambar 16, ada 3 buah komputer yang akan saling berkomunikasi. Komputer 1 adalah sebuah *Public Key Authority Server* atau disingkat sebagai *PKA Server*. Seperti sudah diterangkan sebelumnya, PKA Server menyimpan semua kunci publik agar bisa didapatkan oleh pihak yang ingin saling berkomunikasi. *Client* akan mengirimkan suatu file ke *File Server*. File ini akan dienkripsi dengan enkripsi *AES*, ditandatangani dengan *SHA-256* dan *RSA*. Kunci publik yang dibutuhkan akan diperoleh dari *PKA Server.* Di *File server*, identitas pengirim dan keutuhan file akan diperiksa. Kemudian akan ada konfirmasi dari *File Server* ke *Client*.

Urut-urutan aplikasi:

1. Masing-masing komputer yaitu File Server, Client dan PKA Server akan membuat

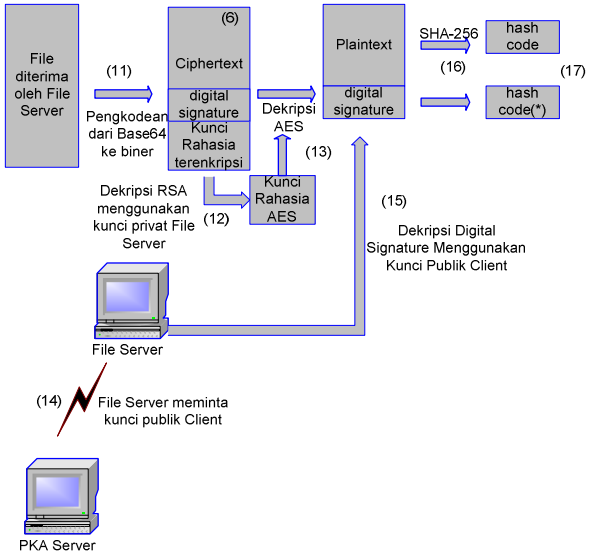
sepasang kunci publik dan privat menggunakan algoritma RSA.

2. Agar bisa terdistribusi ke pada pihak yang memerlukan, maka File Server dan Client akan menyimpan kunci publik masing-masing di *PKA Server*.



Gambar 17. Proses di *Client*

3. *Client* akan mengirimkan suatu *file* ke *File Server.* Dengan menggunakan algoritma *SHA-256*, *client* akan membuat *message digest* atau *hash code* dari *plaintext.*
4. *Client* akan mengenkripsi *hash code* menggunakan kunci privat miliknya untuk menghasilkan *digital signature. Digital signature* ini akan disambungkan pada *file* yang akan dikirimkan.
5. *Client* akan membuat kunci rahasia untuk melakukan enkripsi file menggunakan algoritma AES.
6. *Client* meminta kunci publik *File Server* dari PKA server dan kemudian PKA server akan mengirimkan kunci publik yang dienkripsi *RSA* menggunakan kunci privat *PKA Server*. *Client* akan membuka menggunakan kunci publik *PKA Server*.
7. Kunci rahasia *AES* pada butir 5 akan dienkripsi menggunakan algoritma *RSA*. Ini berarti kunci rahasia ini dienkripsi menggunakan kunci publik milik *File Server* yang didapatkan dari *PKA Server*.
8. Kunci rahasia yang sudah dienkripsi pada butir 6 akan disambungkan dengan file yang sudah dienkripsi pada butir 5.
9. File pada butir 7 akan diubah ke dalam pengkodean *base64*.
10. File dikirimkan ke *File server*.



Gambar 18. Proses di *File Server*

11. *File server* akan melakukan pengkodean kembali dari *base64* ke biner.
12. *File Server* akan melakukan dekripsi kunci rahasia *AES* menggunakan kunci privat.
13. Kunci rahasia hasil dekripsi butir 11 digunakan untuk dekripsi *file* dengan algoritma *AES*.
14. *File Server* meminta kunci publik *Client* dari *PKA server. PKA server* akan mengirimkan kunci publik yang dienkripsi *RSA* menggunakan kunci privat *PKA Server*. *File*

Server akan membuka menggunakan kunci publik *PKA Server*.

15. Hasil dari butir 13 adalah *plaintext* disambung dengan *digital signature client*. *File Server* akan melakukan dekripsi *digital signature* menggunakan kunci publik milik *Client*. Hasilnya adalah *hash code(*).*
16. Dari *plaintext* yang diterima dibuatkan *hash code* menggunakan *SHA-256*.
17. *Hash code* yang dihitung oleh file server dan *hash code(*)* yang diterima dari client diperbandingkan. Jika keduanya sama, maka *data integrity* dan *authentication* terpenuhi. Artinya *file* tidak mengalami perubahan dan identitas pengirim terpenuhi. *File Server* akan mengirimkan pernyataan kepada *client* bahwa *file* telah diterima dengan baik, sebaliknya jika *hash code* dan *hash code(*)* tidak sama, maka *file* ini akan ditolak. Ada 2 kemungkinannya, terjadi perubahan pada isi *file* atau pengirim tidak teridentifikasi.

5. Kesimpulan

Dalam makalah ini, kami telah menjelaskan teknik kriptografi modern untuk menyediakan layanan dasar keamanan informasi yaitu kriptografi simetris yaitu Advanced Encryption Standard (AES) dan kriptografi asimetris yaitu RSA serta message digest SHA-256. Ketiga teknik ini dan didukung dengan pengelolaan kunci dapat diterapkan pada bahasa pemrograman .NET untuk membangun aplikasi pengiriman data teramankan.

6. Daftar Kepustakaan

FEBR04 Jack Febrian, Pengetahuan Komputer dan Teknologi Informasi

COME06 Douglas E. Comer, Internetworking with TCP/IP Principles, Protocols and Architecture Volume1, Prentice Hall, 2006

JJ89 Jeniffer Seberry, Josef Pieprzyk, Cryptography: An Intrduction to Computer Security

KAHN96 Kahn, D. The Codebreakers: The Story of secret writing. New York:Scribner, 1996

KONH81 Alan G. Konheim, Cryptography, A Primer, John Wiley & Sons, Inc 1981

MOV96 A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

MSW05 Mohammad A. Musa, Edward F. Schaefer, Stephen Wedig, A Simplified AES Algorithm And Its Linear And Differential Cryptanalyses, 2005

Scha96 Edward F. Schaefer, A simplified Data Encryption Standard algorithm, 1996

Schn96 Bruce Schneier, Applied Cryptography, Protocols, Algorithms, and Source Code n C, 1996

Sha49 Shannon Claude, Communication Theory of Secrecy Systems. Bell Systems Technical Journal No. 4, 1949

Sta03 Willam Stallings, Cryptography and Network Security, Principles and Practices, 2003

Sta04 William Stalling, Data and Computer Communications, 2004

Tan03 Andrew S. Tanenbaum, Computer Networks, 2003

CCITT Data Communication Networks: Open System Interconnection (OSI); Security, Structure and Applications. Security Architecture For Open Systems Interconnection For CCITT Applications. Recommendation X.800. Geneva, 1991.

Adam Berent AES (Advanced Encryption Standard) Simplified

FIPS99 FIPS PUB 46-3 Specifications For The Data Encryption Standard (DES), October 25, 1999

FIPS02 FIPS PUB 180-2 Announcing the SECURE HASH STANDARD, August 1 2002

FIPS94 FIPS PUB 186 Announcing the Standard for Digital Signature Standard (DSS), May 19, 1994

FIPS01 FIPS PUB 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES), November 26, 2001

Diffie-Hellman Key Exchange and ElGamal

Stefan Lucks, Attacking Triple Encryption. Fast Software Encryption 1998

RFC 2313 - PKCS #1: RSA Encryption Version 1.5

RFC 2437 - PKCS #1: RSA Cryptography Specifications Version 2.0

RFC 2828 : Internet Security Gloassary

Mihir Bellare, Tadayoshi Kohno: Hash Function Balance and Its Impact on Birthday Attacks. EUROCRYPT 2004

Gary C. Kessler, An Overview of Cryptography 1998

HEBE Shireen J. Hebert **A Brief History of Cryptography** http://www.cybercrimes.net/Cryptography/Articles/Hebert.html

Guide, Architecture for Public-Key Infrastructure (APKI) Draft 1 The Open Group, May 1997

Rob95 M.J.B. Robshaw , Stream Ciphers, RSA Laboratories Technical Report TR-701, Version 2.0 July 25, 1995

JJ89 Jeniffer Seberry, Josef Pieprzyk, Cryptography: An Intrduction to Computer Security