Algoritma *Twofish* Sebagai Finalis AES dan Metode Kriptanalisisnya

Dani – NIM : 13504060

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if14060@students.if.itb.ac.id

Abstrak

Twofish merupakan 128-bit cipher block yang diajukan oleh Bruce Schneier, dapat menerima panjang kunci hingga mencapai 256 bits. Cipher nya sendiri merupakan jaringan *Feistel* 16-round, dengan fungsi F bijektif dan terdiri dari 8x8 bit S-boxes yang bergantung pada kunci, 4x4 MDS matrix pada GF(2^8), transformasi pseudo-Hadamard, perputaran bitwise, dan penjadwalan kunci yang didesign dengan baik. Implementasi algoritma Twofish yang optimal mengenkripsi pada kecepatan 17.8 *clock cycles* per *byte* pada processor *Pentium Pro*. Twofish dapat diimplementasikan pada perangkat keras dalam 14000 gerbang. Design dari fungsi *round* dan penjadwalan kunci memungkinkan variasi *tradeoffs* antara kecepatan, ukuran perangkat lunak, waktu pembentukan kunci, jumlah gerbang, dan memory. Algoritma ini tidak mengandung kunci yang lemah, sangat efisien, serta memiliki design yang fleksible dan simple.

Kata kunci: Twofish, kriptografi, kriptanalisis, *block cipher*, Advanced Encryption Standard, enkripsi, dekripsi

1. Pendahuluan

Algoritma DES (*Data Encryption Standard*) yang banyak digunakan sebagai standard enkripsi kriptografi kunci simetri mendekati akhir penggunaannya karena mengandung banyak kontroversi dan dianggap sudah tidak aman lagi. Beberapa kriptografer keberatan karena proses pembuatannya yang bersifat tertutup. Selain itu panjang kuncinya juga dianggap terlalu pendek untuk dapat digunakan secara luas dalam berbagai bidang, dengan perangkat keras khusus kuncinya bisa ditemukan dalam beberapa hari. *National Institute of Standard and Technology* (NIST) sebagai agensi Departemen Perdagangan Amerika Serikat mengusulkan kepada Pemerintah Federal Amerika Serikat untuk sebuah standard kriptografi yang baru.

NIST mengadakan sayembara terbuka, meminta publik untuk mengajukan standardnya dan membuat algoritma untuk memenuhi standard tersebut. NIST mengajukan syarat bahwa algoritma baru tersebut termasuk ke dalam algoritma kriptografi simetri berbasis *cipher* blok. *Cipher* blok dapat digunakan untuk mendesain *cipher* aliran dengan variasi sinkronisasi dan properti ekstensi kesalahan, fungsi hash satu arah, kode autentifikasi pesan, dan generator bilangan *pseudo-random*. Karena fleksibilitasnya, *cipher* blok digunakan dalam kriptografi modern. Selain itu terdapat beberapa kriteria lain : seluruh rancangan algoritma baru harus publik (tidak dirahasiakan), panjang kunci fleksibel, ukuran blok yang dienkripsi adalah 128 bit, dan algoritma dapat diimplementasikan baik sebagai software maupun hardware.

NIST menerima 15 proposal yang masuk, dan memilih 5 finalis yang didasarkan pada aspek keamanan algoritma, kemangkusan (*efficiency*), fleksibilitas, dan kebutuhan memori yang penting untuk *embedded system*. Twofish, dibuat oleh Bruce Schneier dan timnya, merupakan salah satu dari lima finalis tersebut. Algoritma ini memenuhi semua criteria NIST - 128 bit blok, 128, 192, 256 bit kunci, dapat bekerja secara efisien pada berbagai platform, tidak memiliki kunci lemah (*weak keys*), dan memiliki design yang fleksibel serta sederhana.

Twofish dapat :

- mengenkripsi data pada 285 *clock cycles* per blok pada Pentium Pro, setelah 12700 *clock-cycle* pembentukan kunci.
- mengenkripsi data pada 860 *clock cycles* per blok pada Pentium Pro, setelah 1250 *clock-cycle* pembentukan kunci.
- mengenkripsi data pada 26500 *clock cycles* per blok pada 6805 smart card, setelah 1750 *clock- cycle* pembentukan kunci.

2. Blok Pembangunan Twofish

2.1 Jaringan Feistel

Hampir semua algoritma *cipher* blok bekerja dalam model jaringan Feistel. Jaringan Feistel ditemukan oleh Horst Feistel dalam designnya tentang Lucifer, dan dipopulerkan oleh DES. Jaringan Feistel adalah metode umum untuk mentransformasi fungsi apapun (biasa disebut fungsi F) ke dalam permutasi. Beberapa algoritma kriptografi lain yang menggunakan jaringan Feistel misalnya LOKI, GOST, FEAL, Blowfish, Khufu Khafre, dan RC-5. Model jaringan Feistel bersifat reversible, untuk proses enkripsi dan dekripsi, sehingga kita tidak perlu membuat algoritma baru untuk mendekripsi cipherteks menjadi plainteks.
Bagian yang penting dari jaringan Feistel adalah fungsi F, pemetaan string input menjadi string output berdasarkan kunci yang digunakan. Fungsi F selalu tidak linear dan mungkin tidak surjektif:

$$F : \{0,1\}^{n/2} \times \{0,1\}^{N} \rightarrow \{0,1\}^{n/2}$$

di mana n adalah ukuran blok dari jaringan Feistel dan F adalah fungsi yang menerima n/2 bits dari blok dan N bits kunci sebagai input dan menghasilkan n/2 bits output. Dalam setiap *round*, blok sumber adalah input fungsi F dan output dari fungsi F diXORkan dengan blok target, lalu dua blok ini dipertukarkan sebelum masuk ke *round* berikutnya. Ide yang digunakan adalah untuk mengambil fungsi F, yang mungkin merupakan algoritma enkripsi yang lemah jika berdiri sendiri, dan melakukan perulangan untuk membuat algoritma enkripsi yang kuat. Dua *round* pada jaringan Feistel disebut satu *cycle*, dan dalam satu cycle setiap bit dari blok teks telah dimodifikasi sekali.
Twofish merupakan jaringan Feistel 16-round, dengan fungsi F bijektif.

2.2 Kotak-S (S-boxes)

Kotak-S adalah matriks yang berisi substitusi non-linear yang memetakan satu atau lebih bit dengan satu atau lebih bit yang lain dan digunakan di banyak *cipher* blok. Kotak-S memiliki ukuran input dan ukuran output yang bervariasi. Ada empat pendekatan yang digunakan dalam mengisi Kotak-S : dipilih secara acak, dipilih secara acak lalu diuji, dibut oleh orang, dihitung secara matematis. Kotak-S pertama digunakan di Lucifer, lalu DES dan diikuti banyak algoritma enkripsi yang lain.
Twofish menggunakan empat buah 8x8 bit Kotak-S yang berbeda, bijektif, dan bergantung pada kunci. Kotak-S ini dibuat menggunakan 8x8 bit permutasi dan material kunci.

2.3 MDS Matrices

Kode MDS (Maximum Distance Separable) pada sebuah field adalah pemetaan liner dari x elemen field ke y elemen field, dan menghasilkan vektor komposit $x + y$ elemen, dengan ketentuan bahwa jumlah minimum dari elemen bukan nol pada setiap vektor bukan nol paling sedikit $y + 1$. Dengan kata lain, jumlah elemen yang berbeda diantara dua vektor berbeda yang dihasilkan oleh pemetaan MDS paling sedikit $y + 1$. Dapat dibuktikan dengan mudah bahwa tidak ada pemetaan yang dapat memiliki jarak pisah yang lebih besar diantara dua vektor yang berbeda, maka disebut jarak pisah maksimum(maximum distance separable). Pemeteaan MDS dapat direpresentasikan dengan sebuah MDS matriks yang teridiri dari $x \times y$ elemen. Kode perbaikan-kesalahan Reed-Solomon (RS) adalah MDS. Kondisi yang diperlukan untuk sebuah $x \times y$ matriks untuk menjadi MDS adalah semua kemungkinan submatriks kotak, yang diperoleh dengan membuang kolom atau baris, adalah tidak singular.
Serge Vaudenay pertama kali mengajukan MDS matrices sebagai elemen desain *cipher*. Shark dan Square menggunakan MDS matrices, meskipun konstruksinya pertama kali ditemukan di *cipher* Manta yang tidak dipublikasikan. Twofish menggunakan sebuah 4x4 matriks MDS pada GF(2^8).

2.4 Transformasi Pseudo-Hadamard

Transformasi Pseudo-Hadamard (PHT) adalah sebuah operasi pencampuran sederhana yang berjalan secara cepat dalam perangkat lunak. 32-bit PHT dengan dua masukkan didefinisikan esbagai :

$$a' = a + b \bmod 2^{32}$$
$$b' = a + 2b \bmod 2^{32}$$

SAFER menggunakan 8-bit PHT untuk difusinya. Twofish menggunakan 32-bit PHT untuk mengubah keluaran dari fungsi *g* nya. PHT ini dapat dieksekusi dalam dua opcodes di mikroprosesor modern seperti keluarga Pentium.

2.5 Whitening

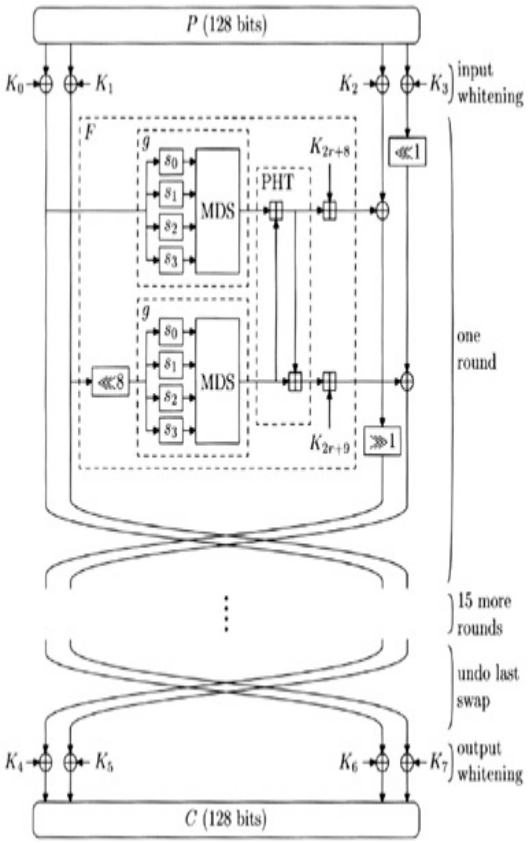
Whitening, sebuah teknik mengXORkan material kunci sebelum round pertama dan setelah round terakhir, digunakan oleh Merkle dalam Khufu/Khafre, dan ditemukan oleh Rivest untuk DES-X. Whitening menambah tingkat kesulitan serangan pencarian kunci terhadap *ciphertext*, dengan menyembunyikan masukkan spesifik terhadap round pertama dan round terakhir dari fungsi F.
Twofish mengXORkan 128-bit sub-kunci sebelum round Feistel yang pertama, dan 128-bit lagi setelah round Feistel terakhir. Sub-kunci ini diperhitungkan dengan cara yang sama seperti sub-kunci round, tetapi tidak digunakan di tempat lain dalam *cipher*.

2.6 Penjadwalan Kunci

Penjadwalan kunci adalah proses pengubahan bit-bit kunci menjadi sub-kunci tiiap round yang dapat digunakan oleh *cipher*. Twofish memerlukan banyak material kunci dan memiliki penjadwalan kunci yang rumit. Untuk memfasilitasi analisis, penjadwalan kunci menggunakan primitif yang sama seperti fungsi round.

3. Twofish

Gambar 1 menunjukkan garis besar algoritma Twofish. Twofish menggunakan 16-round struktur seperti jaringan Feistel dengan penambahan whitening pada input dan output. Satu-satunya elemen bukan Feistel adalah pergeseran satu bit. Pergeseran ini dapat dipindahkan ke dalam fungsi F untuk membuat sebuah jaringan Feistel yang sesungguhnya, namun hal ini memunculkan pergeseran tambahan dari *words* sebelum langkah whitening keluaran.
Plaintext dibagi menjadi empat buah 32-bit words. Pada langkah whitening masukkan, plaintext ini diXORkan dengan 4 words kunci. Langkah ini dilanjutkan oleh enam belas round, yang pada tiap roundnya 2 words di kiri digunakan sebagai masukkan untuk fungsi g (salah satunya terlebih dahulu digeser sebanyak 8 bit ke kiri). Fungsi g teridiri dari empat kotak-S yang bergantung pada kunci, dan diikuti langkah pencampuran linear berbasis matriks MDS. Hasil dari dua fungsi g ini dikombinasikan dengan menggunakan transformasi Pseudo-Hadamard (PHT), dan 2 keywords ditambahkan. Dua hasil ini diXORkan dengan words di sebelah kanan



Gambar 1

(salah satunya terlebih dahulu digeser sejauh 1 bit ke kiri, dan satu lagi digeser 1 bit ke kanan setelah diXORkan). Bagian kiri dan kanan lalu dipertukarkan untuk round berikutnya. Setelah semua round selesai dilakukan, pertukaran terakhir dikembalikan dan empat words tersebut diXORkan dengan empat keywords menghasilkan *ciphertext*.
Untuk lebih formalnya, 16 bytes dari plaintext p0,…,p15 dibagi menjadi 4 words P0,…,P3 dengan setiap wordsnya menggunakan konvensi little-endian.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)} \cdot 2^{8j} \qquad i = 0, \ldots, 3$$

Pada langkah whitening masukkan, words ini diXORkan dengan 4 words kunci yang telah diekspansi.

$$R_{0,i} = P_i \oplus K_i \; i = 0,...,3$$

Di tiap 16 rounds, dua words yang pertama digunakan sebagai masukkan untuk fungsi F,

yang juga menerima nomor round sebagai masukkan. Word yang ketiga diXORkan dengan keluaran pertama dari F, lalu digeser ke kanan sejauh 1 bit. Word yang keempat digeser ke kiri sejauh 1 bit, lalu diXORkan dengan keluaran kedua dari fungsi F. Setelah semua langkah selesai dilakukan, kedua sisi tersebut dipertukarkan. Maka,

$$R_{r+1,0} = \mathrm{ROR}(R_{r,2} \oplus F_{r,0}, 1)$$
$$R_{r+1,1} = \mathrm{ROL}(R_{r,3}, 1) \oplus F_{r,1})$$
$$R_{r+1,2} = R_{r,0}$$
$$R_{r+1,3} = R_{r,1}$$

dengan r = 0,…,15 dan ROR dan ROL adalah fungsi yang menggeser argumen pertamanya (sebuah word 32-bit) ke kiri atau ke kanan sesuai dengan jumlah bit yang diindikasikan oleh argument ke-duanya.
Langkah whitening keluaran membatalkan pertukaran round terakhir dan mengXORkan data words dngan 4 words kunci yang telah diekspansi.

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4} \; i = 0,...,3$$

Empat words dari *ciphertext* kemudian ditulis sebagai 16 bytes c0,…,c15 menggunakan konvensi little-endian yang sama dengan yang digunakan untuk *plaintext*.

$$c_i = \left\lfloor \frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8 \qquad i = 0, \ldots, 15$$

3.1 Fungsi F

Fungsi F adalah permutasi 64-bit yang bergantung pada kunci. Fungsi ini menerima tiga argumen, dua masukkan words R_0 dan R_1, dan nomor round r yang digunakan untuk memilih sub-kunci yang diperlukan. R_0 dimasukkan ke dalam fungsi g, sehingga menghasilkan T_0. R_1 digeser sejauh 8 bit ke kiri lalu dimasukkan ke dalam fungsi g dan menghasilkan T_1. T_0 dan T_1 kemudian dikombinasikan dalam sebuah PHT dan ditambahkan dua words kunci yang telah diekspansi.

$$T_0 = g(R_0)$$
$$T_1 = g(\mathrm{ROL}(R_1, 8))$$
$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$
$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$

di mana (F_0, F_1) adalah hasil dari F.

3.2 Fungsi *g*

Fungsi g merupakan inti dari Twofish. Word masukkan X dibagi menjadi empat byte. Setiap byte diproses dengan Kotak-S nya masing-masing yang bergantung pada kunci. Setiap Kotak-S adalah bijektif, menerima 8 bit masukkan dan menghasilkan 8 bit keluaran. Keempat hasil yang diperoleh diinterpretasikan sebagai vektor dengan panjang 4 dalam GF(2^8), dan dikalikan dengan matriks MDS 4x4 (menggunakan field GF(2^8) untuk komputasi). Vektor yang dihasilkan diinterpretasikan sebagai word 32-bit, dan menjadi hasil dari fungsi g.

$$x_i = \lfloor X/2^{8i} \rfloor \bmod 2^8 \qquad i = 0, \ldots, 3$$
$$y_i = s_i[x_i] \qquad i = 0, \ldots, 3$$
$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} . & \cdots & . \\ \vdots & \text{MDS} & \vdots \\ . & \cdots & . \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

dengan s_i adalah kotak-S yang bergantung pada kunci dan Z adalah hasil dari *g*. Untuk mendefinisikannya dengan baik, diperlukan spesifikasi korespondensi antara nilai byte dan elemen field pada GF(2^8). GF(2^8) direpresentasikan sebagai GF(2)[x]/(x) di mana $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ adalah polinom primitif berderajat 8 pada GF(2^8). Elemen field $a = \sum_{i=0}^{7} a_i x^i$ dengan $a_i \in$ GF(2) diidentifikasi dengan nilai byte $\sum_{i=0}^{7} a_i 2^i$.
Matriks MDS diberikan sebagai :

$$\mathrm{MDS} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix}$$

di mana elemennya telah dituliskan sebagai nilai byte heksadesimal menggunakan korespondensi yang didefinisikan di atas.

3.3 Penjadwalan Kunci

Penjadwalan kunci harus menyediakan 40 words kunci yang telah diekspansi K_0,…,K_{39}, dan 4 Kotak-S yang bergantung pada kunci dan digunakan pada fungsi g. Twofish didefinisikan untuk kunci dengan panjang N = 128, N = 192,

dan N = 256. Kunci yang lebih pendek dari 256 bit dapat digunakan dengan melakukan padding bit 0 sampai panjang kunci terdekat yang didefinisikan.
Didefinisikan k = N/64. Kunci M terdiri dari 8k byte $m_0,\ldots,m_{k-1}$. Byte-byte tersebut dikonversikan menjadi 2k words masing-masing 32 bit.

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \cdot 2^{8j} \qquad i = 0,\ldots,2k-1$$

dan lalu menjadi dua word vektor dengan panjang k.

$$M_e = (M_o, M_2, \ldots, M_{2k-2})$$
$$M_0 = (M_1, M_3, \ldots, M_{2k-1})$$

Word vektor ketiga dengan panjang k juga diturunkan dari kunci. Hal ini dilakukan dengan mengambil byte kunci dalam grup 8, menginterpretasikannya sebagai vektor pada GF(2^8), dan mengalikannya dengan matriks 4x8 yang diturunkan dari kode RS. Setiap 4 byte hasil kemudian diinterpretasikan sebagai word 32-bit. Word-word ini membangun vektor yang ketiga.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} \cdot & \cdots & \cdot \\ \vdots & \text{RS} & \vdots \\ \cdot & \cdots & \cdot \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}$$

$$S_i = \sum_{j=0}^{3} s_{i,j} \cdot 2^{8j}$$

untuk i = 0,…,k-1 dan

$$S = (S_{k-1}, S_{k-2}, \ldots, S_0)$$

Perlu dicatat bahwa S melist words-words tersebut dalam urutan yang terbalik. Untuk perkalian matriks RS, GF(2^8) direpresentasikan sebagai GF(2)[x]/$w(x)$, di mana $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ adalah polinom primitif lain berderajat 8 pada GF(2^8). Pemetaan antara nilai byte dan elemen dari GF(2^8) menggunakan definisi yang sama dengan yang digunakan untuk perkalian matriks MDS. Menggunakan pemetaan ini, matriks RS diberikan sebagai :

$$\text{RS} = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix}$$

Tiga vektor M_e, M_o, dan S membentuk basis dari penjadwalan kunci.

3.3.1 Penambahan Panjang Kunci

Twofish dapat menerima semua kunci dengan panjang lebih kecil sama dengan 256 bit. Untuk ukuran kunci yang tidak didefinisikan di atas, kunci tersebut dipadding pada ujungnya dengan bit-bit 0 hingga mencapai panjang kunci terdekat yang sudah didefinisikan. Sebagai contoh, sebuah kunci 80 bit $m_0,\ldots,m_9$ diperpanjang dengan membuat $m_i = 0$ untuk i = 10,…,15 dan diperlakukan sebagai kunci 128 bit.

3.3.2 Fungsi h

Gambar 2 menunjukkan fungsi h secara garis besar. Fungsi ini menerima dua masukkan – sebuah word 32-bit X dan sebuah list $L = (L_0, \ldots, L_{k-1})$ dari word 32-bit dengan panjang k – dan menghasilkan sebuah word output. Fungsi ini bekerja dalam k tahapan. Pada setiap tahap, empat byte tersebut masing-masing dilewatkan pada Kotak-S yang tetap, dan diXORkan dengan byte yang diturunkan dari list. Setelah itu, byte-byte tersebut sekali lagi dilewatkan pada Kotak-S dan dikalikan dengan matriks MDS seperti pada fungsi g. Untuk lebih formalnya, kita bagi words menjadi byte.

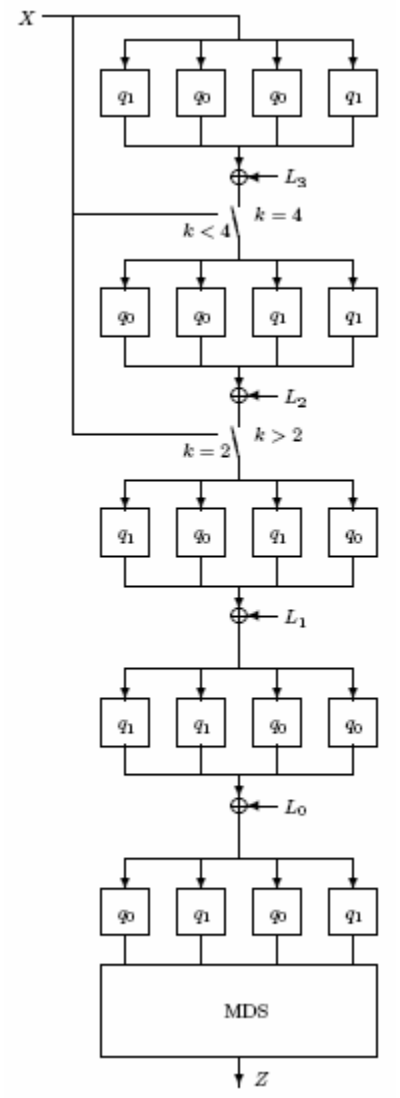
$$l_{i,j} = [L_i/2^{8j}] \bmod 2^8$$
$$x_j = [X/2^{8j}] \bmod 2^8$$

untuk i = 0,…,k – 1 dan j = 0,…,3. Lalu sekuensi substitusi dan XOR diterapkan.

$$y_{k,j} = x_j \; j = 0,\ldots,3$$

Jika k = 4 kita mendapat

$$y_{3,0} = q_1[y_{4,0}] \oplus l_{3,0}$$
$$y_{3,1} = q_0[y_{4,1}] \oplus l_{3,1}$$
$$y_{3,2} = q_0[y_{4,2}] \oplus l_{3,2}$$
$$y_{3,3} = q_1[y_{4,3}] \oplus l_{3,3}$$



Gambar 2

Jika k ≥ 3 kita mendapat

$y_{3,0} = q_1[y_{3,0}] \oplus l_{2,0}$
$y_{3,1} = q_0[y_{3,1}] \oplus l_{2,1}$
$y_{3,2} = q_0[y_{3,2}] \oplus l_{2,2}$
$y_{3,3} = q_1[y_{3,3}] \oplus l_{2,3}$

Dalam semua kasus kita mendapat

$y_0 = q_1[q_0[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}]$
$y_1 = q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1}]$
$y_2 = q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}]$
$y_3 = q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}]$

q_0 dan q_1 adalah permutasi pada nilai 8 bit yang akan didefinisikan di bagian lain. Vektor hasil y_i dikalikan dengan matriks MDS seperti pada fungsi g.

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} . & \cdots & . \\ \vdots & \text{MDS} & \vdots \\ . & \cdots & . \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

dengan Z adalah hasil dari fungsi h.

3.3.3 Kotak-S yang Bergantung pada Kunci

Kotak-S yang digunakan dalam fungsi g didefinisikan sebagai

$$g(X) = h(X, S)$$

untuk i = 0,…,3, Kotak-S s dibentuk dengan pemetaan dari x ke y dalam fungsi h, di mana list L sama dengan vektor S yang diturunkan dari kunci.

3.3.4 Word Kunci *Kj* yang Diekspansi

Word dari kunci yang diekspansi didefiniskan dengan menggunakan fungsi h.

$\rho = 2^{24} + 2^{16} + 2^8 + 2^0$
$A_i = h(2_{i\rho}, M_e)$
$B_i = \text{ROL}(h((2i + 1)_\rho, M_o), 8)$
$K_{2i} = (A_i + B_i) \bmod 2^{32}$
$K_{2i+1} = \text{ROL}((A_i + 2B_i) \bmod 2^{32}, 9)$

Konstanta ρ digunakan untuk menduplikasi byte; konstanta ini memiliki property bahwa untuk i = 0,…,255, word $i\rho$ terdiri dari empat byte yang sama, masing-masing dengan nilai i. Fungsi h diterapkan untuk word dengan tipe ini. Untuk *Ai* nilai byte adalah 2*i*, dan argument kedua dari h adalah *Me*. *Bi* dikomputasikan dengan cara yang sama menggunakan 2*i* + 1 sebagai nilai byte dan *Mo* sebagai argument kedua, dengan pergeseran tambahan sejauh 8 bit. Nilai dari *Ai* dan *Bi* dikombinasikan dalam sebuah PHT. Salah satu hasilnya digeser sejauh 9 bit. Kedua hasil ini membentuk dua word dari kunci yang diekspansi.

3.3.5 Permutasi q_0 dan q_1

Permutasi q_0 dan q_1 adalah permutasi pada nilai 8 bit. q_0 dan q_1 masing-masing dibangun dari empat permutasi 4-bit yang berbeda. Untuk nilai masukkan x, didefinisikan nilai keluaran y sebagai berikut:

$a_0, b_0 = [x/16], x \bmod 16$
$a_1 = a_0 \oplus b_0$
$b_1 = a_0 \oplus \mathrm{ROR}_4(b_0,1) \oplus 8a_0 \bmod 16$
$a_2, b_2 = t_0[a_1], t_1[b_1]$
$a_3 = \mathrm{a}_2 \oplus b_2$
$b_3 = a_2 \oplus \mathrm{ROR}_4(b_2,1) \oplus 8a_2 \bmod 16$
$a_4, b_4 = t_2[a_3], t_3[b_3]$
$y = 16\ \mathrm{b}_4 + a_4$

di mana ROR_4 adalah sebuah fungsi yang sama dengan ROR yang melakukan pergeseran nilai 4 bit. Pada awalnya, byte dipisahkan menjadi dua nibble. Nibble ini dikombinasikan dalam langkah pencampuran bijektif. Setelah itu, setiap nibble dilewatkan pada Kotak-S 4-bit nya masing-masing, dan diikuti langkah pencampuran dan pencarian Kotak-S yang lain. Akhirnya, dua nibble tersebut direkombinasikan menjadi sebuah byte. Untuk permutasi q_0, Kotak-S 4-bit nya diberikan sebagai berikut :

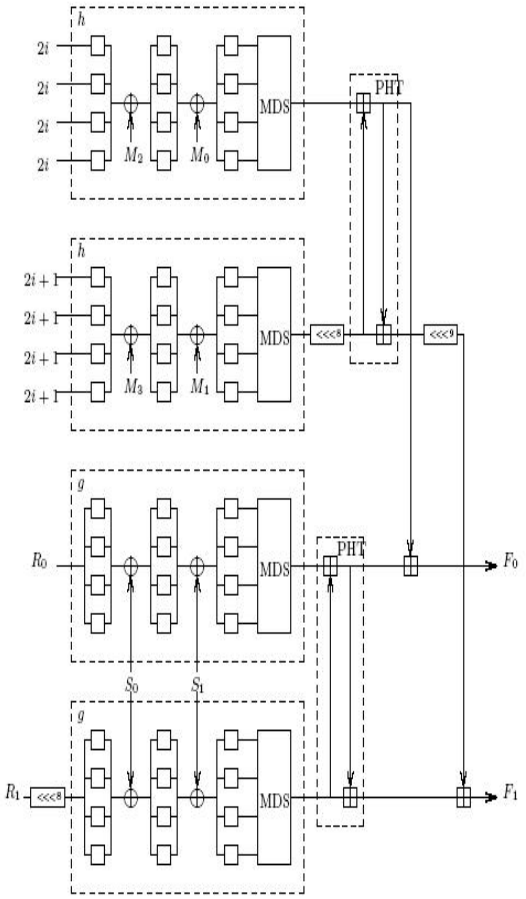
$t0$ = [8 1 7 D 6 F 3 2 0 B 5 9 E C A 4]
$t1$ = [E C B 8 1 2 3 5 F 4 A 6 7 0 9 D]
$t2$ = [B A 5 E 6 D 9 0 C 8 F 3 2 4 7 1]
$t3$ = [D 7 F 4 1 2 6 E 9 B 3 0 8 5 C A]

di mana setiap Kotak-S 4-bit direpresentasikan dengan list entry menggunakan notasi heksadesimal. (Entry untuk masukkan 0,1,…,15 dilist sesuai urutan.)
Dengan cara yang sama, untuk $q1$ Kotak-S 4-bit nya diberikan sebagai :

$t0$ = [2 8 B D F 7 6 E 3 1 9 4 0 A C 5]
$t1$ = [1 E 2 B 4 C 3 7 6 D A 5 F 9 0 8]
$t2$ = [4 C 7 5 1 6 9 A 0 E D 8 2 B 3 F]
$t3$ = [B 9 5 1 C 3 D E 6 4 7 F 2 0 8 A]

3.4 Overview Fungsi Round

Gambar 3 menunjukkan tampilan yang lebih detail mengenai bagaimana fungsi F dikomputasi setiap round jika panjang kunci yang digunakan adalah 128 bit. Pembuatan Kotak-S dan sub-kunci round membuat fungsi round Twofish terlihat lebih rumit, akan tetapi berguna untuk memvisualkan secara tepat bagaimana algoritma ini bekerja.



Gambar 3

4. Performansi Twofish

Dari awal, Twofish telah didesain dengan memperhitungkan faktor performansi. Algoritma ini efisien untuk berbagai platform : 32-bit CPU, 8-bit smart card, perangkat keras VLSI. Untuk performa Twofish pada 8-bit smart card dan perangkat keras VLSI tidak akan dibahas lebih lanjut di sini. Twofish didesain untuk memfasilitasi beberapa lapisan dari *tradeoffs* performa, tergantung pada kepentingan relatif dari kecepatan enkripsi, pembentukan kunci, penggunaan memori, jumlah gerbang perangkat keras, dan berbagai parameter implementasi lainnya. Twofish sangat fleksibel dan dapat diimplementasikan dengan efisien pada berbagai aplikasi kriptografi.

4.1 Performa pada *Large Microprocessor*

Tabel 1 memberikan performa Twofish, enkripsi atau dekripsi, untuk pilihan penjadwalan kunci

yang berbeda dan pada prosesor Pentium Pro/II dengan menggunakan berbagai bahasa dan *compiler*.

Language	Keying Option	Code Size	Clocks to Key			Clocks to Encrypt		
			128-bit	192-bit	256-bit	128-bit	192-bit	256-bit
Assembly	Compiled	8900	12700	15400	18100	285	285	285
Assembly	Full	8450	7800	10700	13500	315	315	315
Assembly	Partial	10700	4900	7600	10500	460	460	460
Assembly	Minimal	13600	2400	5300	8200	720	720	720
Assembly	Zero	9100	1250	1600	2000	860	1130	1420
MS C	Full	11200	8000	11200	15700	600	600	600
MS C	Partial	13200	7100	9700	14100	800	800	800
MS C	Minimal	16600	3000	7800	12200	1130	1130	1130
MS C	Zero	10500	2450	3200	4000	1310	1750	2200
Borland C	Full	14100	10300	13600	18800	640	640	640
Borland C	Partial	14300	9500	11200	16600	840	840	840
Borland C	Minimal	17300	4600	10300	15300	1160	1160	1160
Borland C	Zero	10100	3200	4200	4800	1910	2670	3470

Tabel 1

4.1.1 Pilihan *Keying*

Semua pilihan keying melakukan komputasi K_i untuk $i = 0,\ldots,39$ sebelumnya dan menggunakan 160 byte RAM untuk menyimpan konstanta-konstanta ini. Perbedaan muncul pada cara mengimplementasikan fungsi g. Pilihan-pilihan tersebut meliputi:

- Full Keying

Pilihan ini melakukan prekomputasi keseluruhan kunci. Dengan menggunakan 4 KB tempat pada tabel, setiap Kotak-S diekspansi menjadi 8x32 bit tabel yang mengkombinasikan pencarian Kotak-S dan perkalian oleh kolom matriks MDS. Komputasi fungsi g terdiri dari empat pencarian tabel dan tiga XOR. Enkripsi dan dekripsi konstan dan tidak bergantung pada panjang kunci.

- Partial Keying

Untuk aplikasi di mana beberapa blok dienkripsi dengan kunci tunggal, tidak realistis untuk membangun penjadwalan kunci yang lengkap. Pilihan partial keying melakukan prekomputasi empat Kotak-S di 8x8 bit tabel, dan menggunakan empat 8x32 bit tabel MDS untuk melakukan perkalian MDS. Hal ini mengurangi tempat tabel penjadwalan kunci sampai 1 KB. Kecepatan enkripsi dan dekripsi juga konstan dan tidak bergantung pada panjang kunci.

- Minimal Keying

Untuk aplikasi di mana beberapa blok dienkripsi dengan kunci tunggal, dan memiliki optimasi yang lebih baik. Dibandingkan dengan partial keying, lebih sedikit satu lapisan q-box yang diprekomputasi ke tabel S-box, dan q-box sisanya dilakukan selama proses enkripsi. Pilihan ini menggunakan tabel 1 KB untuk menyimpan Kotak-S hasil prekomputasi sebagian.

- Zero Keying

Tidak melakukan prekomputasi Kotak-S sehingga tidak memerlukan tabel tambahan. Setiap entry dikomputasi *on the fly*, waktu pembuatan kunci hanya terdiri dari komputasi nilai K_i dan S. Untuk aplikasi yang tidak memiliki waktu pembuatan kunci, waktu yang diperlukan untuk mengenkripsi satu blok adalah jumlah dari waktu pembuatan kunci dan waktu enkripsi untuk pilihan zero keying.

- Compiled

Pilihan ini tersedia hanyan untuk bahasa *assembly*, konstanta sub-kunci secara langsung dicocokan pada salinan kode yang spesifik terhadap kunci.

4.1.2 Ukuran Kode dan Data

Ukuran kode untuk implementasi Twofish yang optimal pada Pentium Pro berkisar antara 8450 byte pada assembler sampai 14100 byte pada Borland C. Selain kode juga ada 4600 byte tabel untuk matriks MDS, q_0 dan q_1 yang diperlukan untuk pembentukan kunci, dan setiap kunci memerlukan sekitar 4300 byte tabel data yang bergantung pada kunci untuk pilihan full keying. Pilhan keying lain menggunakan lebih sedikit tempat data dan tabel seperti telah dijelaskan sebelumnya.

4.2 Perbandingan dengan Finalis AES Lain

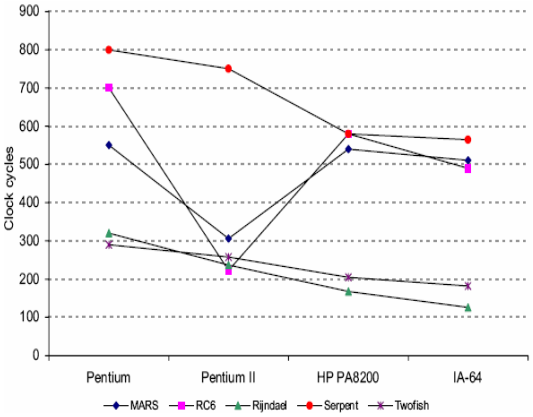
4.2.1 Performa sebagai Fungsi Panjang Kunci

Kecepatan dari MARS, RC6, dan Serpent untuk melakukan organisasi kunci, mengenkripsi dan mendekripsi tidak tergantung dari panjang kunci. Twofish mengenkripsi dan mendekripsi dengan kecepatan yang tidak dipengaruhi panjang kunci, namun memerlukan waktu yang lebih lama untuk organisasi kunci yang lebih panjang. Kecepatan Rjindael sangat dipengaruhi oleh panjang kunci, baik saat organisasi kunci maupun proses enkripsi dan dekripsi.

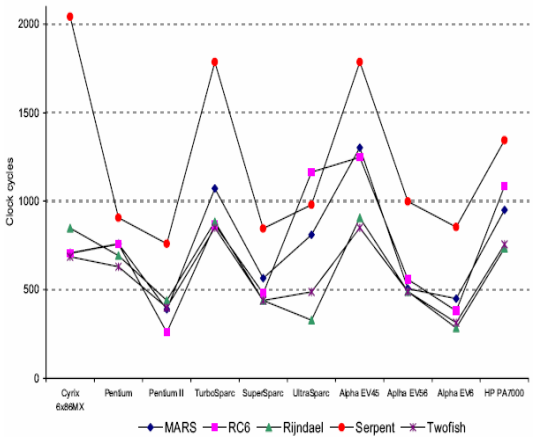
4.2.2 Performa Perangkat Lunak

Tabel 2 dan 3 memberikan perbandingan finalis AES pada CPU yang berbeda. Tabel 2

memberikan kecepatan enkripsi untuk kunci 128-bit dalam assembly, sedangkan tabel 3 dalam C.



Tabel 2



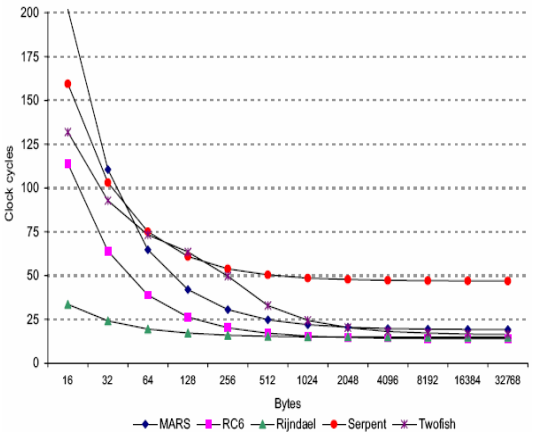
Tabel 3

4.2.3 Organisasi Kunci dan Enkripsi

Untuk enkripsi blok plainteks yang sedikit, performa adalah fungsi dari kecepatan enkripsi dan kecepatan organisasi kunci. Tabel 4 memberikan tingkat kecepatan organisasi kunci dan enkripsi per byte untuk kunci 128 bit pada Pentium II dalam bahasa assembly.

4.3 Pengujian

Telah dibuat sebuah perangkat lunak (bukan oleh penulis) yang dapat melakukan proses enkripsi dan dekripsi dengan menggunakan algoritma Twofish, Serpent, MARS, RC6, Rjindael, dan DES. Perangkat lunak tersebut dapat didownload di http://students.if.itb.ac.id/~if14060/kriptografi/ Pada URL yang sama juga terdapat modul proses enkripsi dan dekripsi dengan algoritma Twofish untuk bahasa Python. Perangkat lunak dan



Tabel 4

modul tersebut merupakan alat bantu yang digunakan penulis dalam melakukan pengujian. Bagian ini bertujuan untuk mengetahui kecepatan proses enkripsi dan dekripsi Twofish dibandingkan dengan finalis AES lain dalam berbagai mode operasi blok cipher yang diajarkan (*ECB, CBC, CFB,* dan *OFB*). Pengujian dilakukan pada sebuah komputer Pentium 4 3.00 GHZ dengan RAM 512 MB. Hasil pengujian tidak dapat dijadikan sebagai acuan karena pengujian hanya dilakukan pada dua buah berkas dengan ukuran 5.21 MB dan 10.1 MB.

4.3.1 Perancangan Kasus Uji Program

Kasus uji yang dirancang adalah sebagai berikut:

1. Kasus Uji 1
   Pengujian proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi menggunakan algoritma Twofish dan Serpent, dengan mode operasi *ECB, CBC, CFB,* dan *OFB* untuk panjang kunci 128-bit, 192-bit, dan 256-bit pada berbagai ukuran file.
2. Kasus Uji 2
   Pengujian proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi menggunakan algoritma Twofish dan MARS dengan mode operasi *ECB, CBC, CFB,* dan *OFB* untuk panjang kunci 128-bit, 192-bit, dan 256-bit pada berbagai ukuran file.
3. Kasus Uji 3
   Pengujian proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi menggunakan algoritma Twofish dan RC6, dengan mode operasi *ECB, CBC, CFB,* dan *OFB* untuk panjang kunci 128-bit, 192-bit, dan 256-bit pada berbagai ukuran file.

4. Kasus Uji 4
   Pengujian proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi menggunakan algoritma Twofish dan Rjindael dengan mode operasi *ECB, CBC, CFB,* dan *OFB* untuk panjang kunci 128-bit, 192-bit, dan 256-bit pada berbagai ukuran file.
5. Kasus Uji 5
   Pengujian proses enkripsi dan dekripsi beserta lama waktu proses enkripsi dan dekripsi menggunakan algoritma Twofish dan DES dengan mode operasi *ECB, CBC, CFB,* dan *OFB* pada berbagai ukuran file.

4.3.2 Evaluasi Hasil Pengujian

Perangkat lunak diasumsikan telah melakukan proses enkripsi dan dekripsi dengan benar. Proses enkripsi dengan menggunakan kunci tertentu akan menyandikan berkas plainteks dan proses dekripsi dengan kunci yang sama akan mengembalikan berkasi cipherteks menjadi berkas plainteks awal. Kesalahan penggunaan kunci mengakibatkan berkas hasil dekripsi tidak sama dengan berkas plainteks awal.

Kasus Uji 1 menunjukkan bahwa:
1. Untuk panjang kunci 128-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih cepat daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Serpent.
2. Untuk panjang kunci 192-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih cepat daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Serpent.
3. Untuk panjang kunci 256-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih cepat daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Serpent.
4. Pemilihan mode operasi blok cipher memang mempengaruhi waktu enkripsi dan dekripsi, namun tidak dapat dijadikan sebagai acuan perbandingan algoritma Twofish dan Serpent karena mode operasi CFB dan OFB memang memerlukan waktu lebih lama dari mode operasi ECB dan CBC. Untuk selanjutnya perbedaan mode operasi akan diabaikan.

Hasil ini menunjukkan bahwa untuk berkas berukuran kecil, algoritma Twofish memerlukan waktu yang lebih sedikit dalam melakukan proses enkripsi dan dekripsi pada berbagai panjang kunci dibandingkan dengan algoritma Serpent.

Kasus Uji 2 menunjukkan bahwa:
1. Untuk panjang kunci 128-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma MARS.
2. Untuk panjang kunci 192-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma MARS.
3. Untuk panjang kunci 256-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma MARS.

Hasil ini menunjukkan bahwa untuk berkas berukuran kecil, algoritma Twofish memerlukan waktu yang lebih lama dalam melakukan proses enkripsi dan dekripsi pada berbagai panjang kunci dibandingkan dengan algoritma MARS.

Kasus Uji 3 menunjukkan bahwa:
1. Untuk panjang kunci 128-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma RC6.
2. Untuk panjang kunci 192-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma RC6.
3. Untuk panjang kunci 256-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma RC6.

Hasil ini menunjukkan bahwa untuk berkas berukuran kecil, algoritma Twofish memerlukan waktu yang lebih lama dalam melakukan proses enkripsi dan dekripsi pada berbagai panjang kunci dibandingkan dengan algoritma RC6.

Kasus Uji 4 menunjukkan bahwa:
1. Untuk panjang kunci 128-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Rjindael.
2. Untuk panjang kunci 192-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Rjindael.

3. Untuk panjang kunci 256-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Rjindael.

Hasil ini menunjukkan bahwa untuk berkas berukuran kecil, algoritma Twofish memerlukan waktu yang lebih lama dalam melakukan proses enkripsi dan dekripsi pada berbagai panjang kunci dibandingkan dengan algoritma Rjindael.

Kasus Uji 5 menunjukkan bahwa:

1. Untuk panjang kunci 128-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma DES.
2. Untuk panjang kunci 192-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma DES.
3. Untuk panjang kunci 256-bit, waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma Twofish lebih lama daripada waktu yang diperlukan untuk proses enkripsi dan dekripsi algoritma DES.

Perlu dicatat bahwa lamanya waktu enkripsi dan dekripsi Twofish dibandingkan dengan DES tidak dapat dijadikan sebagai acuan performansi karena algoritma Twofish memiliki tingkat keamanan yang lebih baik dibandingkan DES. Pengujian ini dilakukan hanya sebagai perbandingan saja.

5. Kriptanalisis Twofish

Algoritma ini telah dicoba dikriptanalisis oleh pembuatnya, dan hasil dari serangan yang berhasil adalah sebagai berikut :

- Pada Twofish dengan Kotak-S, tanpa pergeseran 1 bit, tanpa whitening, dan dengan metode serangan *meet in the middle* pada sebelas round setelah memerlukan 2^{225} memory, 2^{56} plainteks yang diketahui (known plaintext), dan 2^{232} kerja, dan metode serangan *differential* memecahkan sembilan round memerlukan 2^{41} memory, 2^{41} plainteks yang dipilih, dan 2^{254} kerja.
- Pada Twofish standard, empat round serangan *meet in the middle* memerlukan 2^{225} memory, 2^{56} plainteks yang diketahui (known plaintext), dan 2^{232} kerja. Serangan *differential* memecahkan lima round dengan 2^{232} kerja dan 2^{41} query plainteks yang dipilih.
- Serangan *chosen-key* melibatkan pemilihan 160 bit pasangan kunci, K, K*, dengan bit sisanya untuk ditemukan. Serangan ini memerlukan 2^{34} kerja, 2^{32} query plainteks yang dipilih, dan 2^{12} query plainteks yang dipilih adaptif untuk memecahkan sepuluh round tanpa whitening.
- Serangan *related-key* terhadap Twofish sepuluh round tanpa whitening. Serangan ini memerlukan 2^{155} query *related-key*, 2^{187} kerja, dan untuk setiap 2^{155} kunci memerlukan 2^{32} plainteks yang dipilih dan 2^{12} plainteks yang dipilih adaptif.

Fakta bahwa Twofish mampu bertahan dengan baik terhadap serangan *related-key* merupakan hasil yang paling menarik, karena serangan jenis ini memberikan penyerang paling banyak kendali terhadap masukkan cipher. Kriptanalisis konvensional memungkinkan penyerang untuk mengendalikan masukkan plainteks dan cipherteks ke cipher. Kriptanalisis dengan related-key memberikan penyerang jalan tambahan menuju ke cipher : penjadwalan kunci. Sebuah cipher yang dapat bertahan terhadap serangan dengan related-key seharusnya dapat bertahan terhadap teknik yang lebih sederhana yang hanya melibatkan plainteks dan cipherteks.
Dapat disimpulkan bahwa tidak terdapat serangan yang lebih efisien lagi terhadap Twofish daripada brute force. Serangan paling efisien terhadap Twofish dengan panjang kunci 128 bit memiliki kompleksitas 2^{128}; serangan paling efisien terhadap Twofish dengan panjang kunci 192 bit memiliki kompleksitas 2^{192}; dan serangan paling efisien terhadap Twofish dengan panjang kunci 256 bit memiliki kompleksitas 2^{256}.

5.1 Serangan *Meet in the Middle*

5.1.1 Hasil Serangan

Varian cipher : 256-bit key, Full Twofish
Round : 4
Kerja : 2^{232}
Memory :2^{225}

5.1.2 Teknik Serangan

Pada serangan *meet in the middle*, diturunkan beberapa state intermediate di dalam cipher, menggunakan hanya sebagian material kunci dari plainteks dan cipherteks. Untuk setiap nilai yang mungkin, yang menurut terkaan dapat diterima bagian dari kunci, diturunkan nilai intermediate dari plainteks dan juga dari cipherteks. Setelah itu didapat list berukuran besar yang berisi nilai intermediate yang mungkin. List tersebut

diurutkan dan diperiksa untuk nilai yang sesuai (*match*). Nilai yang sesuai untuk jumlah bit yang lebih banyak dari yang diharapkan untuk muncul secara acak mengindikasikan bahwa tebakan telah benar. Serangan jenis ini penting untuk dipertimbangkan karena kandidat AES menggunakan panjang kunci yang besar, yang mengimplikasikan tingkat keamanan yang tinggi. Penyerangan terhadap Twofish lengkap masih sulit dilakukan. Dari sisi plainteksnya, harus ditebak 128 bit S, sub-kunci round yang menentukan nilai yang diXORkan pada C di round pertama, dan sub-kunci pre-whitening untuk C. Hal ini memungkinkan kita memprediksi low-order bit dari B setelah round kedua diXORkan dengan konstanta yang tidak diketahui (bit ini akan menerima sebuah urutan nilai atau negasi dari urutan tersebut). Pendekatan yang sama digunakan untuk sisi cipherteks, mengijinkan hanya empat round untuk diserang.

5.2 Kriptanalisis Diferensial

5.2.1 Hasil Serangan

Varian cipher : 256-bit key, Full Twofish
Round : 5
Teks yang dipilih : 2^{41}
Kerja :2^{232}

5.2.2 Teknik Serangan

Ide utama dari serangan ini adalah untuk memaksa karakteristik fungsi F ke bentuk (a,b) → (X,0) untuk muncul di round ke-dua. Jika hal ini terjadi, dan ada k bits pada perbedaan XOR tersebut, ada 2^{-k} kemungkinan bahwa keluaran dari PHT akan tidak berubah pada salah satu dari dua wordnya, yang akan mengakibatkan pola yang bisa dideteksi di perbedaan keluaran dari round ketiga, yang lebih jauh lagi akan mengakibatkan pola yang bisa dideteksi (meskipun diperlukan kerja yang sangat keras) di perbedaan keluaran dari round ke-lima cipher, yang pada akhirnya memungkinkan serangan terhadap Twofish dengan jumlah round yang dikurangi (*reduced-round*). Serangan ini mengharuskan adanya perbedaan urutan yang spesifik dan dapat diprediksi. Karena itu dipilih :

r	$\Delta_{R_r,0}$	$\Delta_{R_r,1}$	$\Delta_{R_r,2}$	$\Delta_{R_r,3}$
0	0	0	a'	b'
1	a	b	0	0
2	0	X	a	b
3	Y	Z	0	X
4	Q	R	Y	Z
5	S	T	Q	R

di mana tabel tersebut memberikan pola perbedaan untuk nilai round untuk setiap 5 round, dengan a' = ROL(a, 1), b' = ROR(b, 1). Z = ROL(a,1) $\oplus$ $F_{2,1}$, dan low-order bit dari $F_{2,1}$ dipastikan nol; juga dipilih a sehingga low-order bit dari ROL(a,1) juga nol. Lalu perbedaan Z dapat dideteksi, karena low-order bit dari Z selalu nol. Untuk mengetahui pasangan yang benar, harus ditebak sub-kunci round terakhir dan mendekripsi satu round di atasnya, memeriksa apakah low-order bit dari perbedaan seperti yang diharapkan. Ada beberapa cara lain untuk mengidentifikasi perbedaan Y, Z, akan tetapi hal tersebut tidak mengubah kesulitan serangan. Perlu dicatat bahwa X, Y, Z, S, T, Q, dan R adalah perbedaan yang nilainya tidak dipedulikan, selama low-order bit Z adalah nol.

Satu-satunya saat yang kritis untuk serangan ini muncul pada r = 1, di mana kita perlu karakteristik (a,b) → (X,0) untuk terjadi dengan probabilitas yang tinggi. Berdasarkan properti dari matriks MDS, diketahui bahwa terdapat tiga byte tunggal keluaran XOR untuk s_0 yang mengakibatkan keluaran XOR dari g dengan berat Hamming hanya 8. Jika perbedaan yang sama dengan ini masuk ke dalam MDS matriks dalam kedua komputasi g di round ke-dua,lalu, dengan probabilitas 2^{-8}, diperoleh pasangan offset dari nilai dalam keluaran dua g, keluaran yang satu memiliki beberapa nilai yang ditambahkan padanya dalam modulo 2^{32}, dan keluaran yang lain memiliki nilai yang sama dikurangi dari keluaran tersebut. Ketika keluaran tersebut dilewatkan pada PHT, mengakibatkan hanya satu word keluaran dari seluruh F diubah, yang memungkinkan serangan ini dilakukan. Oleh karena itu, dipilih a = (α,0,0,0) untuk menjadi perbedaan byte tunggal di s_0, dan b = ROR(a,8) sehingga perbedaan byte tunggal menuju komputasi g yang ke-dua pada round 2 bersesuaian dengan perbedaan byte tunggal di a.

Pada serangan ini, dipertimbangkan N merupakan *batch* 256 pasangan plainteks, di mana paling tidak satu diharapkan mengandung pasangan yang semuanya tepat. Harus dipilih masukkan ke cipher untuk membentuk *batch* yang seperti ini. Setelah itu diterka beberapa bagian dari kunci yang diperlukan untuk memeriksa satu bit yang diketahui dari perbedaan yang masuk round ke-lima. Kunci terkaan tersebut akan digunakan untuk menguji setiap *batch*. *Batch* yang tidak dibangun dari pasangan yang tepat akan dibedakan setelah sejumlah kecil pasangan diperiksa, karena perbedaan 1 bit akan salah dengan probabilitas ½ dalam setiap pasangan yang salah diperiksa

dengan kunci yang tepat, dan dalam setiap pasangan diperiksa dengan kunci yang tepat. Serangan tersebut terdiri dari langkah-langkah berikut :

- Meminta enkripsi dari blok plainteks yang diperlukan untuk membentuk setidaknya satu *batch* dari 256 pasangan plainteks yang memiliki probabilitas tinggi untuk dibangun hanya dari pasangan yang benar.
- Menerka bagian terkecil dari material kunci yang akan memperlihatkan satu bit perbedaan yang diketahui dalam pasangan yang benar.
- Untuk setiap terkaan, periksa setiap *batch* sampai ditemukan satu *batch* yang dibangun hanya dari pasangan yang tepat.

Jika terdapat N *batch* untuk diperiksa, dan setiap *batch* rata-rata memerlukan dua kali percobaan dekripsi yang layak dilakukan, maka akan terdapat 2N kerja yang harus dilakukan untuk setiap terkaan material kunci yang diperlukan untuk mengetahui apakah perbedaan yang diketahuinya adalah perbedaan yang diharapkan.

5.3 Kriptanalisis *Related-Key*

5.3.1 Hasil Serangan

Varian cipher : Twofish tanpa pre- dan post-whitening
Tipe Serangan : Related-*Key*
Round : 10
Faktor kerja : 2^{187}
Jumlah plainteks yang dipilih : 2^{32}
Jumlah plainteks yang dipilih adaptif: 2^{12}
Keperluan special : 2^{155} query related-key

5.3.2 Teknik Serangan

Kriptanalisis Related-Key menggunakan penjadwalan kunci cipher untuk memecahkan plainteks yang dienkripsi dengan kunci yang berelasi. Dalam bentuk yang paling *advance*, kriptanalisis diferensial related-key, plainteks dan kunci (dengan diferensial yang dipilih) digunakan untuk menemukan kunci. Analisis dengan tipe ini memiliki kemungkinan sukses terhadap cipher dengan penjadwalan kunci sederhana - seperti GOST dan 3 Way – dan merupakan serangan yang realistis pada keadaan tertentu. Serangan konvensional biasanya ditentukan parameter jumlah plainteks dan cipherteks yang diperlukan untuk serangan, dan tingkat akses terhadap cipher yang diperlukan untuk mendapatkan teks-teks itu (plainteks yang diketahui, plainteks yang dipilih, plainteks yang dipilih adaptif). Serangan related-key menambahkan parameter lain yaitu jumlah kunci yang berelasi yang akan digunakan untuk mengenkripsi plainteks.

Serangan diferensial related-key melakukan penyerangan diferensial terhadap blok cipher lewat kunci, dan dapat juga ditambahkan melalui port plainteks/cipherteks. Terhadap Twofish, serangan seperti ini harus mengontrol urutan perbedaan sub-kunci setidaknya pada round di tengah. Untuk lebih menyederhanakan serangan ini, penyerang dianggap ingin melakukan pengubahan sub-kunci yang dipilih pada sub-kunci 12 round tengah, yaitu mengubah M ke M* dan mengontrol D[i,M,M*] untuk i = 12,…35. Pada waktu yang sama, ia juga harus menjaga fungsi *g* dan kunci S dari perubahan. Diasumsikan kunci yang digunakan memiliki panjang 256 bit. Perlu dicatat bahwa serangan related-key yang berhasil terhadap kunci 128 atau 192 bit yang mendapat hanya nol perbedaan sub-kunci pada round yang harus mengontrol perbedaan sub-kuncinya dapat ditranslasikan secara langsung terhadap serangan related-key pada kunci 256 bit.

Penyerang yang mencoba melakukan serangan diferensial related-key dengan kunci S yang berbeda, harus menghasilkan keluaran *g* yang bebeda untuk setiap masukkan, karena tidak ada pasangan nilai S yang menghasilkan Kotak-S yang identik. Dengan menganggap pasangan nilai S tidak menghasilkan Kotak-S yang terhubung secara linear, tidak dimungkinkan kompensasi untuk perubahan di S dengan perubahan pada sub-kunci di sebuah round. Jika dianggap 12 round aktif pada serangan, kesulitan yang ditambahkan kira-kira seperti menambahkan 24 Kotak-S aktif pada serangan related-key yang sudah dilakukan. Untuk alasan ini, dianggap setiap serangan related-key memerlukan pasangan kunci yang membuat S tidak berubah.

Serangan related-key yang paling sederhana untuk dianalisis adalah yang menjaga S dan sub-kunci 12 round tengah tidak berubah. Serangan ini menghasilkan urutan A dan B yang identik untuk 12 round, dan menjaga urutan byte individu yang digunakan untuk menurunkan A dan B tetap identik.

Kode RS yang digunakan untuk menurunkan S dari M secara ketat membatasi jalan penyerang untuk mengubah M tanpa mempengaruhi S. Penyerang harus mencoba untuk menjaga jumlah sub-kunci aktif yang menghasilkan Kotak-S sesedikit mungkin, karena setiap Kotak-S aktif adalah penghalang lain untuk melancarkan serangannya. Penyerang dapat menjaga jumlah

Kotak-S aktif sampai lima tanpa mengubah S, jadi inilah yang harus dilakukan. Dengan hanya byte kunci yang mempengaruhi lima sub-kunci generasi Kotak-S aktif, penyerang dapat mengubah antara satu dan empat byte pada kelima Kotak-S; sifat dari matriks RS adalah jika ia perlu untuk merubah empat byte pada Kotak-S manapun, ia harus mengubah byte di kelimanya. Dalam prakteknya, untuk memaksimalkan kontrolnya terhadap urutan byte yang dihasilkan oleh Kotak-S ini, ia harus mengubah empat byte di kelima Kotak-S aktif.

Untuk memperoleh nol perbedaan sub-kunci, penyerang harus mendapat nol perbedaan pada urutan byte yang dihasilkan oleh kelima Kotak-S aktif. Misalnya pada urutan tunggal byte seperti itu : penyerang mencoba untuk menemukan pasangan masukkan kunci empat byte di mana hal itu membawa mereka ke urutan byte identik pada 12 round di tengah, yang berarti 12 byte tengah. Ada 2^{63} pasangan masukkan kunci untuk dipilih, dan sekitar 2^{95} urutan byte yang mungkin. Dari analisis urutan byte tersebut (yang tidak dibahas di sini), tidak diharapkan ada pasangan kunci untuk Kotak-S dengan lebih dari delapan byte berurutan yang tidak diubah, dan diharapkan delapan byte berurutan dari urutan byte yang tidak diubah untuk memerlukan control terhadap keempat byte kunci pada Kotak-S. Diharapkan ada pasangan spesifik dari byte kunci yang diperlukan untuk menghasilkan urutan byte yang sama. Untuk meluaskan hal ini ke lima Kotak-S aktif, diharapkan - pada kasus terbaik - ada satu pasangan nilai untuk 20 byte kunci aktif yang menyebabkan ke-delapan sub-kunci tengah tidak berubah.

Penyerang yang memiliki kontrol terhadap urutan perbedaan XOR di A_i, B_i tidak perlu memiliki kontrol penuh terhadap XOR atau urutan perbedaan 2^{32} yang tampak di sub-kunci. Pada konteks serangan diferensial related-key, penyerang biasanya tidak mengetahui semua byte kunci yang menghasilkan baik A_i maupun B_i, akan tetapi ia mengetahui urutan perbedaan XOR di A_i dan B_i.

Dianggap nilai A_i dengan perbedaan XOR adalah δ. Jika berat Hamming dari δ adalah k, maka perkiraan terbaik untuk perbedaan dua word sub-kunci dalam modulo 2^{32} untuk round yang diberikan memiliki kemungkinan sekitar 2^{-k}.

Hal ini menunjukkan kesulitan melakukan serangan related-key yang berhasil dengan perbedaan urutan A_i, B_i yang tidak nol. Jika seorang penyerang dapat menemukan urutan perbedaan untuk A_i, B_i yang menjaga k=3, dan perlu untuk mengontrol perbedaan sub-kunci untuk 12 round, dia memiliki peluang sekitar 2^{-72} untuk mendapatkan urutan perbedaan XOR sub-kunci yang paling mungkin, dan sekitar 2^{-36} untuk mendapatkan urutan perbedaan modulo 2^{32} yang paling mungkin. Setelah mendapatkan perbedaan mod 2^{32} sub-kunci yang paling mungkin, penyerang harus mengatasi keadaan bahwa nilai keluaran diXORkan ke setengah dari blok. Peluang perbedaan mod 2^{32} yang relatif cukup tinggi tersebut kembali berakhir pada 2^{-72} peluang untuk setiap urutan perbedaan XOR yang diberikan yang pada kenyataanya diXORkan ke blok tujuan suksesif round tersebut.

Serangan diferensial related-key tidak bekerja terhadap Twofish, akan tetapi jika Twofish digunakan secara langsung untuk mendefinisikan fungsi hash serangan *chosen-key* dapat digunakan.

5.4 Serangan *Chosen-Key*

5.4.1 Hasil Serangan

Varian cipher : Twofish tanpa pre- dan post-whitening
Tipe Serangan : Partial *Chosen-Key*
Round : 10
Faktor kerja : 2^{34}
Jumlah plainteks yang dipilih : 2^{32}
Jumlah plainteks yang dipilih adaptif: 2^{12}
Keperluan special : 160 bit kunci dari *K, K** dipilih oleh penyerang

5.1.2 Teknik Serangan

Serangan chosen-key merupakan varian dari serangan related-key. Tidak hanya ada beberapa kunci yang berelasi, penyerang dapat memilih beberapa bagian dari kunci ini. Bagian yang tidak dipilih tidak diketahui oleh penyerang, dan tujuan dari serangan ini adalah untuk menemukan bagian-bagian tersebut.

Serangan ini dilakukan sebagai berikut :

- Pilih sepasang kunci, K, K*, dengan ketentuan bahwa sub-kunci round delapan dan nilai kunci S sama untuk kedua kunci tersebut.
- Meminta sebuah enkripsi di bawah K, dan 2^{32} enkripsi di bawah K*, mencoba menemukan jalan untuk mengganti efek perubahan pada sub-kunci round pertama.
- Jika ditemukan pasangan yang tampak benar (dapat dideteksi karena terjadi *zero difference* pada setengah dari blok cipher), gunakan untuk menyelidiki isi dari setiap Kotak-S dalam giliran. Hal ini memungkinkan kita

untuk menemukan nilai kunci S. Nilai ini, dengan diberikannya matriks RS yang digunakan untuk menurunkannya dan pengetahuan awal tentang sebagian dari kunci, memungkinkan kita untuk menemukan sisa kunci secara aljabar.

6. Kesimpulan

Twofish adalah algoritma ideal yang efisien untuk diterapkan baik pada mikroprosesor besar, *smart cards*, maupun perangkat keras *dedicated*. Beberapa lapisan dari *tradeoffs* performa dalam penjadwalan kunci membuat algoritma ini cocok untuk berbagai implementasi. Perhatian khusus pada aspek kriptografi dalam pembuatannya – baik fungsi enkripsi maupun penjadwalan kunci – membuat Twofish cocok sebagai *codebook*, cipher aliran *output-feedback* dan *cipher-feedback*, fungsi hash satu arah (menggunakan teknik standard untuk mengubah blok cipher menjadi fungsi hash), dan pembangkit bilangan *pseudorandom*.

Twofish merupakan blok cipher simetri 128-bit, memiliki panjang kunci 128-bit, 192-bit, atau 256-bit, dan tidak memiliki kunci lemah. Penjadwalan kunci pada Twofish dapat dikomputasi sebelumnya untuk kecepatan maksimum, atau dikomputasi *on the fly* untuk ketangkasan maksimum dan keperluan memori minimum.

Twofish memiliki algoritma enkripsi dan penjadwalan kunci yang dibuat berpasangan, perubahaan pada satu bagian mempengaruhi bagian lainnya. Hal ini disebabkan tidak cukup jika hanya mendesain fungsi round yang kuat dan menerapkan penjadwalan kunci yang kuat pada fungsi tersebut, keduanya harus selalu dikerjakan bersama. Selain itu, Twofish membuat cipher dengan enkripsi lokal yang kuat dan memiliki fungsi round untuk menangani difusi global.

Beberapa pengembangan dapat dilakukan untuk meningkatkan performa Twofish seperti :

- Pengurangan jumlah round. Sampai saat ini, serangan bukan *related-key* (diferensial) dapat memecahkan lima round. Jika tidak ada serangan lebih baik lagi yang ditemukan setelah beberapa tahun, mungkin pengurangan round menjadi 12 atau 14 dapat dilakukan.
- Alternatif fixed tabel untuk meningkatkan keamanan. Twofish menggunakan matriks MDS dan permutasi q_0 dan q_1. Jika ada konstanta lain yang membuat Twofish lebih sulit dikriptanalisis algoritma ini dapat direvisi.
- Varian Twofish dengan dengan Kotak-S yang sudah ditentukan (pasti). Varian ini akan memiliki waktu pembentukan kunci yang lebih cepat dan kecepatan enkripsi dan dekripsi yang sama dengan varian standard.
- Pengembangan batas bawah pada kompleksitas serangan diferensial dan linear.

Referensi

[1] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *The Twofish Encryption Algorithm: A 128-Bit Block Cipher*, John Wiley & Sons, 1999.

[2] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *Twofish : A 128-Bit Block Cipher*, 15 Juni 1998.

[3] R. Munir, *Diktat Kuliah IF5054 Kriptografi*, Program Studi Teknik Informatika Institut Teknologi Bandung, 2006.

[4] B. Schneier, D. Whiting, *A Performance Comparison of the Five AES Finalists*, 7 April 2000.

[5] S. Moriai, Y. Lisa Yin, *Cryptanalysis of Twofish (II)*, 2000.

[6] B. Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, 1996.

[7] http://www.schneier.com/twofish.html, diakses selama Oktober 2006.