Implementasi Algoritma AES-128 pada Mikrokontroler 8051

Arif Kusbandono, *13299108*

***Abstrak*— AES-128 dapat di-*porting* ke prosesor 8 bit. Algoritma ini dapat langsung diterjemahkan ke bahasa pemrograman C. Akan tetapi, melakukan hal tersebut untuk sebuah program mikrokontroler 8051 mengharuskan kita bergantung pada *compiler* C. Dengan demikian, metoda optimasi diimplementasikan di sini dengan menggunakan bahasa C dan *assembly*. Penulis telah mengimplementasikan optimasi hingga mencapai ukuran program 3407 byte; kecepatan komputasi 3658 cycle dan 5648 cycle, berturut-turut untuk enkripsi dan dekripsi. *Throughput* 31. kbps dapat dicapai untuk kristal osilator 11.059 MHz yang dapat dinaikkan lagi menjadi 114.3 kbps pada versi *enhancement* 8051 dengan kristal 40 MHz. *Timing analysis countermeasure* juga dilakukan sebagai bagian dari isu keamanan.**

***Kata kunci*—optimasi, cipher, assembly,microprogramming.**

I. PENDAHULUAN

"WINZIP 9.0 *Now with AES Encryption*," begitu bunyi *tag* iklan program kompresi file yang sangat populer (lebih dari 115 juta download)[1]. Paralel dengan fakta tersebut, dapat dikatakan bahwa 'enkripsi' telah menjadi kosa kata sehari-hari (populer pula). Yang tak kalah jamak adalah popularitas mikrokontroler 8051 (126 juta pengapalan pada 1993)[2] yang berharga US$ 4 - US$ 40 untuk varian Intel-nya. Jika dikonjungsikan: 8051 adalah salah satu jawaban bagi meningkatnya (popularitas) kebutuhan keamanan saat ini; ia memberikan rasio biaya-resiko yang masuk akal bagi implementasi perangkat enkripsi.

Rijmen dan Daemen (sebagai non-Amerika) cukup dikejutkan dengan diumumkannya algoritma Rijndael sebagai *Advanced Encryption Standard* (AES) oleh *National Institute of Standards and Technology* (NIST) pada November 2001 [6]. Proposal Rijndael menyisihkan empat finalis lainnya: MARS, RC6, Serpent, dan Twofish. AES sendiri memang diperuntukan bagi implementasi *software*, *firmware*, *hardware* atau kombinasinya dan dispesifikasikan untuk dapat *porting* ke prosesor 8 bit.

Penerjemahan langsung ke bahasa C dari algoritma AES dimungkinkan. Akan tetapi, pengembangan program mikrokontroler (*microprogramming*) dengan mengandalkan *compiler C* saja akan boros *cycle* dan ukuran program. Itulah alasan dibutuhkannya berbagai pendekatan interpretasi algoritma, terlebih generik 8051 hanya memiliki 128 byte *on chip-data memory* dan 4kB *on chip-code memory*. Makalah ini mengeksploitasi metoda-metoda yang diterapkan sehingga kecepatan yang layak diperoleh dalam perbandingannya dengan implementasi lain.

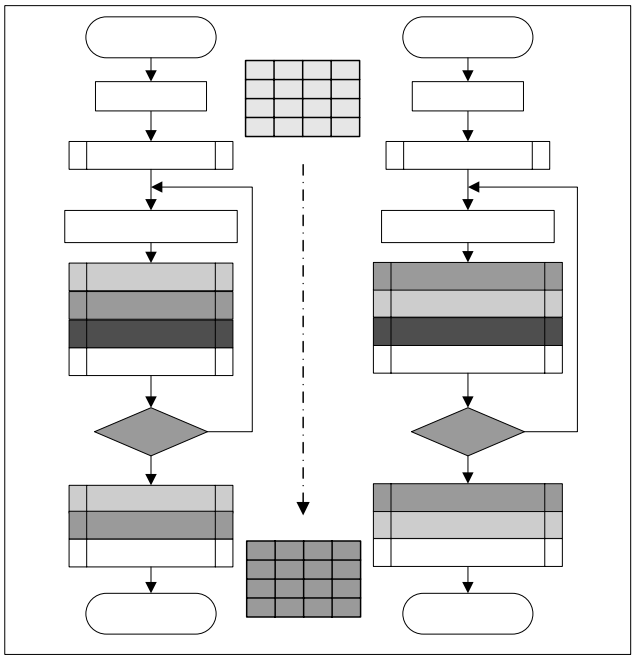
Manuskrip diajukan 8 Juni 2004. A. Kusbandono melakukan penelitian di VLSI-RG Dept. Teknik Elektro ITB (bandono@ee.itb.ac.id) di bawah bimbingan Dr. Ir. Sarwono Sutikno dan Mahmud Galela, ST.

[1] http://www.winzip.com/winzip.htm

[2] http://www.esacademy.com/automation/faq/8051

II. DASAR AES-128

AES-128 adalah *block cipher* simetrik yang memproses 128 bit *plain text* dan 128 bit kunci rahasia ('AES-128' mengacu pada panjang kunci 128, selainnya ada AES-192; AES-256) [6]. *Cipher* (rangkaian transformasi enkripsi) berlangsung dalam rentetan empat fungsi pembangun (primitif), **SubBytes()**, **ShiftRows()**, **MixColumns()**, dan **AddRoundKey()**. Rentetan tersebut dijalankan sebanyak ***Nr*** - 1 sebagai loop utama (*Nr* = 10 untuk AES-128). Setiap loop disebut ***round***. AddRoundKey() dieksekusi sebagai *round* inisial sebelum loop utama. Setelah loop utama tersebut berakhir (sembilan *round*), SubBytes(), ShiftRows(), MixColumns() , dan AddRoundKey() dieksekusi secara berturut-turut sebagai *final round*-nya.



Gmbr. 1. Diagram Alir *Cipher* dan *Inverse Cipher*.

Fungsi-fungsi *cipher* AES 'menyembunyikan' data menjadi bentuk tersandinya lewat empat langkah konversi: langkah **nonlinear**, langkah **dispersi**, langkah **difusi**, dan penambahan **kunci** [3]. Visualisasi data transformasi adalah *array state* dua dimensi 16 byte (1 **blok**) (Gambar 1). Konversi nonlinear diperoleh lewat pemetaan S-Box pada transfomasi SubBytes(). Dispersi dilakukan lewat permutasi byte-byte data dari kolom-kolom *array state* oleh fungsi ShiftRows(). Langkah difusi diwujudkan melalui kombinasi linear dari byte-byte data dalam satu kolom *array* lewat perkalian MixColumns(). Penambahan kunci (AddRoudKey()) dilakukan dengan operasi XOR antara data dengan kunci.

Tiap langkah [6] dengan prefiks **Inv-** adalah transformasi kebalikannya (dekripsi), kecuali AddRoundKey() yang kebalikannya adalah dirinya sendiri. Dekripsi berlangsung dengan invers dari setiap primitif enkripsi (*inverse cipher*). AddRoundKey() dieksekusi sebagai *initial round*, diikuti sembilan *round* rentetan **InvShiftRows()**, **InvSubBytes(), InvMixColumns()**, dan AddRoundKey(). *Round* ke-10 yang mengikutinya tidak menyertakan InvMixColumns serupa dengan *final round* enkripsi. Demikianlah bagaimana fungsi *round* dekripsi dibangun.

III. Implementasi Algoritma

A. Operasi Dasar

Operasi dasar penjumlahan diwujudkan melalui XOR per bit. Implementasi operasi dasar perkalian dalam GF(2^8) (notasi $\bullet$) adalah dengan tabel ${}^{\{03\}}\log\{xy\}$ dan ${}^{\{03\}}\text{antilog}\{xy\}$ [1] atau dengan pergeseran berulang (xtime()).

Pemanfaatan tabel log-antilog untuk perkalian dilakukan dengan memanfaatkan sifat penjumlahan pangkat, $a = g^{\alpha}$ dan $b = g^{\beta}$ maka $a \bullet b = g^{\alpha+\beta}$.

Bilangan biner $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ memiliki representasi polinomial berikut

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Untuk representasi di atas, perkalian xtime(b) = $x \bullet b(x)$ dapat diwujudkan [6] sebagai *left shift* yang diikuti XOR kondisional dengan {1b}

$$xb(x) = b_8x^8 + b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x$$

jika $b_8 = 1$, maka XOR dilakukan, jika $b_8 = 0$, maka XOR tidak dilakukan. *Exclusive-OR* kondisional tersebut tidak lain adalah operasi modulo dengan $m(x) = x^8 + x^4 + x^3 + x + 1$.

B. AddRoundKey() per Blok atau per Word

AddRoundKey() yang diwujudkan tidak terlepas dari ekspansi kunci yang dilakukan. Seperti telah disebutkan (Bagian I), generik 8051 hanya memiliki memori data internal 128 byte untuk keseluruhan program berjalan. Keterbatasan tersebut menjadikan ekspansi kunci dalam [6] tidak bisa dilakukan sekaligus (*single shot*) 44 *word* kunci (176 byte).

Jadi, pilihan jatuh pada ekspansi menggunakan *buffer* siklik (*on the fly*). Ukuran *buffer* yang dibuat menentukan AddRoundKey() yang dibuat. Ukuran lima *word* mengharuskan AddRoundKey() dipanggil setiap kali satu *word* kunci selesai dibuat (pembangkitan satu *word* kunci membutuhkan pengetahuan/memori dari empat *word* kunci sebelumnya). Penjumlahan per blok dimungkinkan dengan membuat minimal delapan *word* buffer (dua blok).

Tinggal kemudian, pilihan implementasi ekspansi kunci menggunakan *buffer* siklik di atas diwujudkan dengan menggunakan alamat absolut atau dengan *pointer*[3].

C. SubBytes() dan InvSubBytes dengan Tabel

Pilihan jatuh pada penggunaan *look up table* S-Box dan S-Box^{-1} [6] untuk SubBytes() dan InvSubBytes(). Konstanta yang digunakan sebagai tabel dialokasikan pada memori *code*. Perhitungan yang ditabelkan dapat dilakukan lebih cepat. Hasil perhitungan sudah berada di akumulator dalam lima *cycle*. Itupun sudah termasuk inisialisasi alamat awal *look up table*. Jadi, jika tabel yang sama masih digunakan oleh baris program berikutnya, kecepatan perhitungan menjadi tiga *cycle* untuk baris-baris 'melihat' tabel berikutnya. Harga yang harus dibayar dari penabelan konstanta adalah pemakaian memori program.

D. ShiftRows() dan InvShiftRows() Implisit

Permutasi baris-baris *state* yang diwujudkan melalui pergeseran siklik dapat dibuat implisit dalam fungsi lain yang mengikutinya [7]. Untuk ShiftRows() permutasi baris dapat dilakukan dengan mengubah indeks *state* yang ditransformasikan MixColumns(), sedangkan untuk InvShiftRows() dilakukan pengubahan indeks *state* masukan InvSubBytes.

E. Persamaan MixColumns() dan InvMixColumns()

Selain dapat menggunakan kedua jenis perkalian di Bagian III-A, persamaan MixColumns() dalam [6] juga dapat diubah menjadi

$$\left.\begin{aligned} p &= s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus s_{3,c} \\ s'_{0,c} &= \{02\} \bullet (s_{0,c} \oplus s_{1,c}) \oplus p \oplus s_{0,c} \\ s'_{1,c} &= \{02\} \bullet (s_{1,c} \oplus s_{2,c}) \oplus p \oplus s_{1,c} \\ s'_{2,c} &= \{02\} \bullet (s_{2,c} \oplus s_{3,c}) \oplus p \oplus s_{2,c} \\ s'_{3,c} &= \{02\} \bullet (s_{3,c} \oplus s_{0,c}) \oplus p \oplus s_{3,c} \end{aligned}\right\}$$

Penjabaran tersebut [4] berasal dari persamaan berikut yang diperoleh dari pergeseran berulang (Bagian III-A)

$$\{03\} \bullet b(x) = (\{02\} \bullet b(x)) \oplus b(x)$$

Dengan cara yang sama, persamaan InvMixColumns() yang asli juga dapat disusun kembali seperti berikut ini.

$$\left.\begin{aligned} s'_{0,c} &= \big(\{0d\} \bullet (s_{0,c} \oplus s_{2,c})\big) \oplus \big(\{09\} \bullet (s_{1,c} \oplus s_{3,c})\big) \oplus \big(\{02\} \bullet (s_{0,c} \oplus s_{1,c})\big) \oplus s_{0,c} \\ s'_{1,c} &= \big(\{0d\} \bullet (s_{1,c} \oplus s_{3,c})\big) \oplus \big(\{09\} \bullet (s_{0,c} \oplus s_{2,c})\big) \oplus \big(\{02\} \bullet (s_{1,c} \oplus s_{2,c})\big) \oplus s_{1,c} \\ s'_{2,c} &= \big(\{0d\} \bullet (s_{0,c} \oplus s_{2,c})\big) \oplus \big(\{09\} \bullet (s_{1,c} \oplus s_{3,c})\big) \oplus \big(\{02\} \bullet (s_{2,c} \oplus s_{3,c})\big) \oplus s_{2,c} \\ s'_{3,c} &= \big(\{0d\} \bullet (s_{1,c} \oplus s_{3,c})\big) \oplus \big(\{09\} \bullet (s_{0,c} \oplus s_{2,c})\big) \oplus \big(\{02\} \bullet (s_{0,c} \oplus s_{3,c})\big) \oplus s_{3,c} \end{aligned}\right\}$$

Penjabaran tersebut juga berasal dari hasil pergeseran berulang berikut ini.

[3] *Special Function Register* generik 8051 yang dapat dijadikan *pointer* alamat adalah R0 dan R1.

$$\left.\begin{aligned}\{0b\}\bullet b(x) &= (\{09\}\bullet b(x))\oplus(\{02\}\bullet b(x))\\ \{0e\}\bullet b(x) &= (\{0d\}\bullet b(x))\oplus(\{02\}\bullet b(x))\oplus b(x)\end{aligned}\right\}$$

Implementasi dalam program mikrokontrolernya cukup dengan menggunakan *look up table* perkalian GF(2^8) untuk pengali {02}, {09}, dan {0d}. Tabel perkalian {02} hanya dipakai enkripsi, sedangkan kesemua tabel dipakai saat dekripsi.

IV. Perangkat Pengembangan Program

Perangkat lunak pengembangan program yang digunakan adalah µVision 2 V2.23 (Keil Software Inc.) [15]. Perangkat lunak ini sudah memuat program C *compiler* dan *assembler* untuk set instruksi MCS-51™.

Library Keil untuk 8051 tersedia dari berbagai varian dan vendor, misalnya seri-seri Intel sendiri, seri 89C dan 89S Atmel, seri C500 Infineon, Phillips, dll. Terdapat pula library generik untuk semua varian 8051. Perincian target library generik tersebut adalah: mikrokontroler *8051-based* (CMOS atau NMOS), 32 jalur I/O, 2 timers, 5 *interrupts*/2 *priority levels,* 4 kB ROM, dan 128 bytes on-chip RAM.

V. Metoda Optimasi Program

Optimasi program implementasi ini dimulai dari tiap fungsi pembangun program. Dengan demikian, optimasi per primitif akan menghasilkan pelipatgandaan kecepatan sesuai pembobotan *cycle*-nya. Pembobotan untuk fungsi-fungsi *round* adalah 11 untuk AddRoundKey(), 10 untuk ShiftRows() dan SubBytes(), dan 9 untuk MixColumns(). Pembobotan tersebut juga berlaku untuk fungsi *round* dekripsi. Dengan menggunakan pergeseran implisit (Bagian III-D), otomatis pembobotan menjadi 1 untuk ShiftRows() dan 0 untuk InvShiftRows(), bobot sisanya dibebankan ke fungsi yang mengikutinya.

Selain yang berkaitan dengan pengubahan persamaan dari algoritma AES (Bagian III), optimasi juga berkaitan dengan gaya penulisan program yang dijelaskan berikut ini.

A. Penulisan dalam Campuran C – Assembly

Pendekatan optimasi yang pertama adalah dengan mengubah penulisan C ke bahasa *assembly*. Pendekatan ini dilakukan jika ternyata listing *assembly* yang dihasilkan compiler terlihat boros instruksi. Selama hasilnya sama, baris-baris *assembly* yang tidak perlu dapat dibuang. Optimasi tersebut sangat mengandalkan pemrogram (subyektif). Baru dapat diketahui efisien tidaknya setelah diperbandingkan secara kuantitatif dengan pencapaian implementasi milik orang lain (obyektif).

Listing *assembly* yang dihasilkan *compiler* dapat digunakan sebagai acuan penulisan ulang fungsi yang hendak dioptimasi. Harga yang harus dibayar untuk instruksi-instruksi yang dapat dipakai [8] menjadi acuan optimasi. Misalnya, secara umum instruksi berikut adalah lebih '**murah'** dibandingkan dengan operasi *storing* lainnya.

```
MOV A,<byte-asal>
MOV <byte-tujuan>,A
```

Harga instruksi di atas hanya satu siklus mesin. Evaluasi juga tidak bisa dilakukan per baris instruksi saja. Contoh berikut ini mengilustrasikan hal tersebut. Instruksi di bawah ini mengekspresikan perpindahan variabel yang sama dengan dua cara berbeda. Dengan memakai akumulator menjadi

```
MOV A,R7
MOV 08h,A
```

tanpa akumulator menjadi

```
MOV 08h,R7
```

Versi pertama (dua baris instruksi) memindahkan isi R7 ke alamat {08} dalam dua *cycle* dan panjang *code* tiga byte, sedangkan versi kedua melakukannya dalam dua *cycle* dan panjang *code* dua byte. Dalam konteks panjang *cycle*, keduanya setara, tetapi dalam konteks ukuran, versi kedua lebih kecil.

Listing *assembly* yang dibangkitkan *compiler* sangat bergantung kepada **gaya penulisan** program C-nya. Sebagai ilustrasi (potongan program dari fungsi `InvMixColumns()`), kedua set instruksi di bawah ini menghasilkan operasi yang sama tetapi *assembly* yang dibangkitkan kompilasinya berbeda. Yang pertama adalah

```
state[0][0] = mul_D[x0_2]^mul_9[x1_3]^state[0][0];
state[0][1] = mul_D[x1_3]^mul_9[x0_2]^state[0][1];
```

Instruksi kedua yang ekivalen adalah

```
state[0][0] = mul_D[x0_2] ^ state[0][0];
state[0][1] = mul_D[x1_3] ^ state[0][1];
state[0][0] = mul_9[x1_3] ^ state[0][0];
state[0][1] = mul_9[x0_2] ^ state[0][1];
```

Set instruksi yang kedua (dalam empat baris bahasa C) telah memiliki '**kemiripan**' dengan *mnemonic*-nya. Dengan menempatkan dua operasi memanggil array (dari tabel) `mul_D[]` secara berurutan maka inisiasi *pointer* array cukup dilakukan sekali untuk dua operasi sekaligus, mirip dengan 'cara berpikir' *assembly*-nya.

Versi empat baris sintaks C ilustrasi tersebut menghasilkan baris-baris *assembly* (hasil kompilasi) yang lebih sedikit daripada versi sintaks C dua baris ilustrasi sebelumya. Otomatis tidak perlu ada optimasi kecepatan lagi (penulisan ulang fungsi bahasa C dalam *assembly* yang lebih singkat dan murah *cycle*-nya).

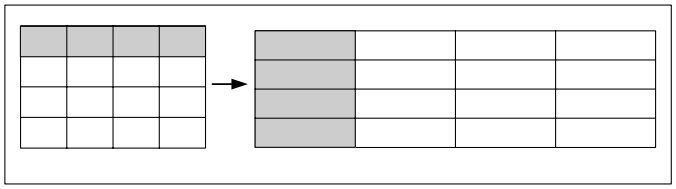
B. Alokasi Memori

Unit terkecil operasi AES-128 adalah byte. Operasi internal berlangsung lewat serangkaian transformasi pada *array* dua dimensi: *state*. Array *state* dideklarasikan (dalam bahasa C) sebagai array dua dimensi yang menempati alamat absolut {08} s.d. {17}.

Konvensi pengindeksan array $s_{\text{baris,kolom}}$ dalam AES-128 dipetakan ke dalam array `state[kolom][baris]` dalam program C (Gambar 2). Pengindeksan **dibalik** setelah menganalisis *assembly* hasil kompilasi. Sebagai contoh, `state[0][0]` setelah kompilasi akan diberi label[4] `state`, `state[0][3]` akan dilabeli dengan `state+03h`, dst. Jadi, jika pointer sedang menyimpan alamat awal array (`state`), maka *incremental* pointer akan menunjuk pada label `state+01h`, `state+02h`, dst. Jika digambarkan sebagai array dua dimensi,

[4] *Label* dalam program *assembly* menyatakan alamat dari baris-baris program (*mnemonic*) atau data yang mengikutinya.

kenaikan tersebut adalah kenaikan indeks dalam arah baris pada kolom yang sama (Gambar 3).



Gmb. 2. Pemetaan array *state* bentuk indeks state[kol][brs]. Indeks tersebut telah dibalik dari notasi algoritmanya.

Label `state` memiliki alamat absolut {08}. Susunan seperti ini menempatkan setengah blok *state* dalam bank register 1 dan setengahnya lagi dalam bank register 2 (Gambar 3). Setiap byte elemen *state* dapat diakses dengan alamat *direct*-nya ({08} – {17}) maupun sebagai register (R7 – R0). Alamat-alamat tersebut dikenali sebagai register jika bank register yang bersangkutan sedang aktif. Instruksi berikut memindahkan isi A ke $s_{0,0}$ dengan alamat langsungnya.

```
MOV 08h,A
```

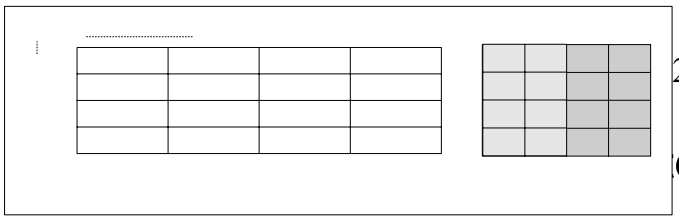
atau dengan menggunakan label seperti

```
MOV state,A
```

Dua cycle dan dua byte diperlukan oleh instruksi di atas. Jika bank register 1 aktif (lewat inisalisasi SFR `PSW`[5]), maka ekspresi berikut adalah ekivalen.

```
MOV R7,A
```

Satu cycle dan satu byte diperlukan oleh instruksi tersebut.



Gmb. 3. Pengalamatan *State* yang Bisa Dipakai.

VI. Validasi Kebenaran Fungsional

Selain digunakannya contoh vektor uji pada dokumen FIPS-197 [6] dalam simulasi selama pengembangan. Modul komputasi diuji dengan men-*download*-nya langsung ke sistem mikrokontroler AT89s8252. Vektor uji yang digunakan adalah yang tersedia pada dokumen *AES Algorithm Validation Suite* (AESAVS) [11] (bukan bagian standar AES-128).

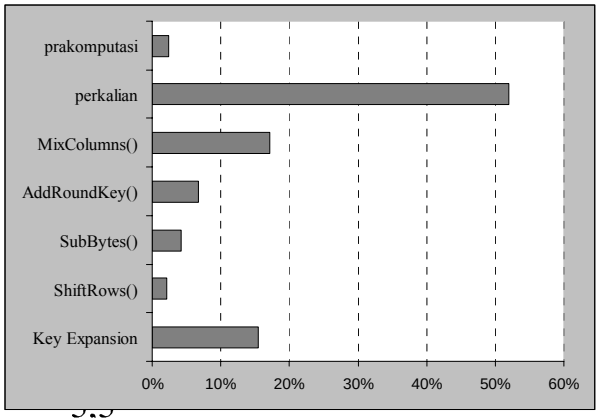
VII. Pusat Perhatian Optimasi Kecepatan

Versi pertama program dibuat untuk memperoleh pondasi pengembangan program mikrokontroler. Umpan balik yang diberikan berupa prioritas pusat pusat perhatian optimasi kecepatan. Deskripsi singkat dari program versi pertama in adalah sebagai berikut:

- pengindeksan: state[row][col]
- perkalian dengan tabel log-antilog
- buffer siklik ekspansi kunci: 5 *word*
- modul *assembly*: tidak ada (*full C*)

Program versi pertama yang dibuat memberikan total waktu eksekusi 16078 *cycle*. Dari jumlah tersebut, 11110 *cycle* disumbangkan oleh `MixColumns()`. Lebih jauh lagi, pembengkakan waktu eksekusi `MixColumns()` disebabkan oleh lambatnya perkalian. Fungsi perkalian jika dihitung terpisah dari `MixColumns()` mendominasi lebih dari 52% waktu eksekusi enkripsi (Gambar 4). Ini meskipun fungsi perkalian ini telah disederhanakan dari algoritma perkalian dengan tabel log-antilog semula dalam [1].

Berdasarkan Gambar 4, prioritas optimasi muncul dalam urutan `MixColumns()` (termasuk perkalian), ekspansi kunci, dan `AddRoundKey()`. Dari evaluasi *assembly* hasil kompilasi, fungsi-fungsi *round* lainnya secara subyektif sudah dianggap efisien.



Gmb. 4. Fungsi-fungsi yang menjadi pusat perhatian optimasi

VIII. Evaluasi Unjuk Kerja

Evaluasi program keseluruhan dilakukan lewat analisis versi terakhir program mikrokontroler. Umpan balik yang diberikan dari program pertama yang dibuat (Gambar 4) menghasilkan versi terakhir program. Fungsi komputasi yang ditulis dalam bahasa *assembly* (teroptimasi) adalah `AddRoundKey()` dan `InvMixColumns()` masing-masing dalam *file* `AddKeyModule.a51` dan `InvMixColumns.a51`. Fungsi I/O ditulis dalam bahasa *assembly* tanpa alasan khusus. Akan tetapi, `InputToState()` dan `StateToOutput()` dalam `I_O.a51` menggunakan konsep bank register (Bagian V-B).

Perbandingan dengan implementasi enkripsi mikrokontroler 8 bit yang lain terangkum dalam Tabel I. Perbandingan dilakukan per fungsi untuk sekali `EncryptRound()` dengan mengubah `Main.c` dan menghilangkan `I_O.a51` dari program versi terakhir. Ini dilakukan untuk memperoleh perbandingan yang obyektif. Pada tiap-tiap implementasi pembanding, masukan teks-kunci, ekspansi kunci, dan percabangan diakumulasi sebagai prakomputasi [7,10]. Juga disebutkan publikasi masing-masing bahwa masukan teks-kunci merupakan proses penyimpanan ke memori (*load/store/move*) pasca-*reset*/inisial. Jadi, ia tidak menggambarkan I/O yang sebenarnya ('*dummy*').

[5] Pada Program Status Word, seleksi bank register yang aktif dikontrol lewat bit PSW.4 dan PSW.3.

TABEL I
PERBANDINGAN ENKRIPSI

Fungsi	**8051†** cycle	byte	**AVR** cycle	byte	**PIC** cycle
prakomputasi	1015	1029	1015	1029	1015
add key	572	68	572	68	572
S-Box	680	84	680	84	680
ShiftRows()	34	41	34	41	34
MixColumns()	1431	229	1431	229	1431
TOTAL	3732	1451	3732	1451	3732

† implementasi AES-128 yang dibuat
sumber: K. Sung Ha [10]; R. Ward [7].

Ukuran program untuk implementasi PIC (Table I) adalah 1739 *word* (menggunakan *single word instructions program memory*). Jadi, secara keseluruhan program 8051 yang dihasilkan memiliki jumlah *cycle* dan ukuran program **terkecil** dari tiga implementasi yang diperbandingkan.

TABEL II
PERBANDINGAN UNJUK KERJA ENKRIPSI 8051

Implementasi 8051	*cycle*	*byte*	*throughput*
8051†	3732	1676	31.6 kbps
8051‡	3658	1451	32.2 kbps
8051 Rijmen *et al.* [4]	3168	1016	37.2 kbps

† dengan prakomputasi
‡ fungsi round saja
throughput untuk kristal 11.059 MHz

Campuran C – assembly yang digunakan terbukti masih efektif untuk menghasilkan kecepatan dan ukuran program yang representatif (Tabel II). Program (fungsi *round* saja) yang dihasilkan hanya 13.4 % lebih lambat dan 30 % lebih besar terhadap implementasi 8051 tercepat yang dipublikasikan. *Throughput* komputasi (31.6 kbps) bisa mencapai 114.3 kbps pada versi *enhancement* 8051 yang dapat bekerja dengan kristal 40 MHz (seri C50x Siemens) [9].

Unjuk kerja dekripsi beserta perbandingan dengan implementasi lain tertera pad Tabel III.

TABEL III
PERBANDINGAN DEKRIPSI

Fungsi	**8051†** *cycle*	*byte*	**AVR** *cycle*	*byte*
prakomputasi	119	175	2693	
InvShiftRows()	0	0	55	
AddRoundKey()	572	68	575	3870
ekspansi kunci	1587	768‡	14534	
InvSubBytes()	760	344‡	313	
InvMixColumns()	2610	1117‡	1402	1546‡
TOTAL	5648	2472	22973	5416

† implementasi AES-128 yang dibuat, fungsi *round* dekripsi saja, tidak ada I/O teks-kunci.
‡ sudah termasuk tabel yang digunakannya
sumber: I. Hermawan [2].

IX. TIMING ANALYSIS COUNTERMEASURE

Perkalian dengan tabel log-antilog [1] maupun pergeseran berulang memiliki dependensi terhadap operan, ada percabangan program (*conditional jump*) yang tidak dieksekusi untuk operan tertentu. Sebagai hasilnya, muncul variasi *cycle* `MixColumns()` untuk *plain text* yang disimulasikan.

Dependensi waktu komputasi terhadap data dihindari dengan membuat *code coverage* fungsi 100% untuk semua operan. Pada `MixColumns()` yang menggunakan tabel log-antilog maupun pergeseran berulang di atas, sejumlah `NOP` perlu disisipkan agar tiap operan perkalian menempuh panjang *cycle* yang sama untuk tiap percabangan yang dilaluinya [3,5]. Program versi terakhir tidak memerlukan penyisipan `NOP` tersebut karena tidak ada *conditional jump* dalam `MixColumns()` yang dibuat (persamaan Bagian III-E). Begitu halnya dengan `InvMixColumns()`. Laporan *code coverage* 100% ini dilihat pada *debugger* Keil.

X. SIMPULAN

Metoda optimasi program yang berfokus pada constraint kecepatan telah berhasil meningkatkan unjuk kerja program komputasi AES-128 keseluruhan. Secara keseluruhan program AES-128 yang dihasilkan dapat dimuat ke dalam mikrokontroler 8051 generik (4kB code memory dan 128 byte data memory) dengan menyisakan kurang dari 600 byte code memory untuk program lainnya. Dengan begitu, semua varian 8051 lain dapat menggunakan modul komputasi ini.

REFERENCES

[1] Brian Gladman, "A Specification for Rijndael, the AES Algorithm," 2002.
[2] I. Hermawan, "Implementasi Algoritma Dekripsi AES-128 pada Mikrokontroler Atmel AT90s8535," Februari 2004.
[3] J. Daemen, V. Rijmen, "Efficient Block Ciphers for Smartcards," http://www.usenix.org/publications/library/proceedings/smartcard99/full_papers/daemen/daemen_html, 1999.
[4] J. Daemen, V. Rijmen, "AES Proposal: Rijndael," 1999.
[5] M. Janke, P. Laackmann, "Power and Timing Analysis Attack Againts Security Controllers," http://www.silicon-trust.com/pdf/secure_5/40_techno_3.pdf
[6] NIST, "Advanced Encryption Standard," Federal Information Processing Standards Publication 197, November 2001.
[7] R. W. Ward, T. C. A. Molteno, "A CPLD Coprocessor for Embedded Cryptography," http://www.physics.otago.ac.nz/electronics/papers/WardMolteno103.pdf
[8] Scott MacKenzie, "The 8051 Microcontroller, 2nd Ed.," *Prentice Hall*, 1995.
[9] Sencer Yeralan, Ashutosh Ahluwalia, "Programming and Interfacing the 8051 Microcontroller," *Addison-Wesley Publishing Co.*, 1995.
[10] Sungha Kim, Ingrid Verbauwhede, "AES Implementation on 8-bit Microcontroller," 2003.
[11] Situs NIST Computer Security Resource Center http://csrc.nist.gov/CryptoToolkit/
[12] Situs FAQ debugger Keil http://www.keil.com/uvision2/debuggerfaq.asp

Arif Kusbandono Born in Bogor, November 24th, 1980, after having completed high school at SMUN I Bogor Senior High, the author is currently an udergraduate student of Institut Teknologi Bandung (ITB) Indonesia majoring in electrical engineering.

By Oct 2003, he has finished an ISO7816 memory card (telecard) reader project at PT INTI Indonesia during his co op. He also submitted an FPGA design implementation of the S-Box circuit in the 7th LSI Design Contest of

the Ryukyu Univ., Okinawa. Mr. Kusbandono is a member of the VLSI-Cryptography research group since 2003.