Studi mengenai *Side Channel Based Attack*, akibatnya terhadap implementasi-implementasi kriptografi dan penanganannya

Nugroho Muhtarif – NIM: 13502054

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl Ganesha 10, Bandung
E-mail: if12054@students.if.itb.ac.id

Abstrak

Pada awalnya kriptanalisis hanya berkutat dalam bidang matematik untuk mencari teknik untuk membongkar kerumitan yang dimiliki sistem kriptografi modern saat ini. Tetapi saat ini muncul jenis serangan baru pada sistem kriptografi yaitu Side Channe Based Attack. Kriptanalisis Side Channel adalah bahan kajian baru dalam kriptanalisis terapan yang terus mendapatkan perhatian sejak pertengahan tahun 90-an. Penelitian di bidang ini telah menunjukkan bahwa kebocoran tidak sengaja yang disebabkan implementasi langsung dari algoritma yang sebenarnya aman dapat begitu penting dalam keamanan. Contohnya adalah pemanfaatan kebocoran untuk menghasilkan komponen-komponen kunci dari implementasi kriptografi yang digunakan. Metode *Side Channel Attack* yang pertama kali dikenalkan adalah *Timing Attack*, selanjutnya ada yang dikenal dengan *Power Analysis* dan *Electromagnetic Emanation*. Teknik yang akan dibahas secara mendetil pada makalah ini adalah *Cache Based Attack* yang termasuk dalan Timing Attack pada implementasi kriptografi *Data Encryption Standard* (DES) beserta penanaganannya yang mungkin dilakukan.

Kata kunci: Kriptanalisis , *Side Channel Attack, Timing Attack, Cache Based Attack, Data Encryption Standard*

1. Pendahuluan

1.1 Kriptanalisis

Kriptanalisis berasal dari bahasa yunani kuno, terdiri dari dua kata *"kyptos"* yang berarti tersembunyi dan *"analyein"* yang berarti melepaskan atau memisahkan. Kriptanalsis berarti adalah studi mengenai metode-metode untuk mendapatkan arti atau makna yang sebenarnya dari informasi yang telah terenkripsi, tanpa memiliki akses ke informasi yang bersifat rahasia yang biasanya diperlukan. Biasanya kriptanalisis melibatkan pencarian kunci rahasia, dalam bahasa non-teknis dikenal dengan *codebreaking,*atau *cracking the code* (pemecaan kode).

Kriptanalisis juga dapat diartikan sebagai segala usaha yang bermaksud menghindari keamanan dari berbagai algoritma kriptografi dan protokol keamanan lainnya. Tetapi metode-metode seperti penyogokan, pemaksaan dengan kekerasan, perampokan, keylogging, dan yang lainnya tidak termasuk dalam kriptanalisis, walaupun aksi-aksi di atas merupakan hal yang penting untuk diperhatikan dalam kemanan komputer dan saat ini menjadi lebih efektif daripada kriptanalisis yang didefinisikan secara tradisional.

Walaupun tujuan kriptanalisis tetap sama sampai sekarang, metode-metode dan teknik-teknik yang digunakan sekarang berubah secara drastis karena makin kompleksnya sistem kriptografi saat ini. Hasil dari kriptanalisis pun sekarang telah berubah. Saat ini tidaklah mungkin meraih kesuksesan tak terbatas dalam pemecahan kode

1.2 Side Channel Based Attack

Biasanya kriptanalisis yang menggunakan pendekatan matematik mengasumsikan jika alat kriptografi (perangkat keras) merupakan mesin abstrak yang mengolah input dan output data dengan algoritma kriptografi tertentu. Tetapi pada dunia yang sebenarnya, serangan

mungkin untuk dilakukan jika penyerang memiliki akses kepada mesin dengan mengobeservasi kelakuan mesin, misalnya penggunaan daya listrik, waktu pengolahan dalam melukan operasi kriptografi, dll.

Saat ini ada yang dikenal dengan *Implementation Attack* yang menyerang langsung ke mesinnya. Serangan dapat berupa *Active Attacks* jika merubah kondisi lingkungan sampai membuka secara fisik mesin. Serangan juga dapat berupa Passive *Attacks* dengan hanya melakukan observasi terahadap kebocoran daril alat kriptografi yang bersifa *inherent*. Serangan pasif menjadi lebih berbahaya karena serangan ini tidak meninggalkan jejak apapun terhadap alat sehingga serangan seringkali disadari belakangan.

Passive Attack dapat dibagi menjadi dua jenis, yaitu: 1) *Side Channel Attack* 2) *Logical Attack.* Pada makalah ini akan dikupas permasalahan mengenai *Side Channel Attack.* Pada praktiknya, *Side Channel Attack* dilakukan dalam beberapa metode atau teknik. Pada tahun 1995, *Timing Attack* adalah *Side Channel Attack* pertama yang dipublikasikan. Teknik ini mengukur waktu eksekusi untuk operasi kriptografii secara keseluruhan. Dalam teknik ini, penyerang harus dapat mensimulasikan atau memprediksi perilaku alat berkaitan dengan waktu. Pada tahun 1998, Paul Kocher memperkenalkan *Power Analysis* yang lebih efektif dari *Side Channel Attack* sebelumnya. Metode ini mengamati konsumsi daya listrik selama alat melakukam operasi kriptografi. Hasil observasi ini akan menghasilkan informasi yang dapat dieksploitasi dengan Simple *Power Analysis* dan *Differential Power Analysis* untuk mendapatkan informasi rahasia. Pada tahun 2000/2001, penggunaan Electromagnetic Emanation sebagai *Side Channel Attack* didemonstrasikan oleh Jean-Jacques Quisquater dan David Samyde yang mengobservasi pancaran gelombang elektromagnetik yang dihasilkan alat selama melakukan operasi kriptografi. **Salah satu teknik** lain dari Side Channel Based Attack adalah **Cache Based Attack** yang akan **dibahas lebih mendalam dalam makalah** beserta penanganan yang mungkin dilakukan

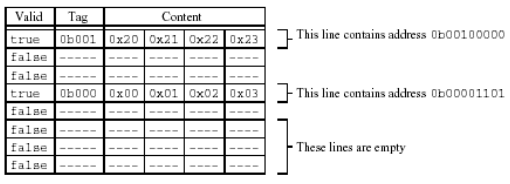
2. Cache Based Attack

2.1 Cache Memory

Memori cache dalam konteks microprocessors adalah area kecil dari memori yang sangat cepat yang diletakan di antara processor dan memori utama. Karena akses ke main memory lambat dibandingkan dengan kecepatan processor, kinerja komputer seringkali memburuk karena operasi yang memerlukan pengambilan dan penyimpanan data. Cache yang beroperasi dengan kecepatan yang hampir menyamai processor membantu untuk memecahkan masalah tersebut dengan menyimpan item yang paling sering digunakan, sehingga mengurangi biaya pengkasesan. Akses baik kepada data dan instruksi-instruksi dapat di-*cache* dengan cara ini, seluruh sistem bergantung pada asumsi bahwa terdapat sekumpulan data yang sedang bekerja yang paling serign diakses. Karena akses terhadap sekumpulan data yang bekerja akan diakselerasikan dengan penggunaan cache, maka kinerja sistem secara keseluruhan akan meningkat.

Tetapi karena memori cache lebih cepat dan lebih mahal, maka cache lebih kecil dari memori utama dan hanya dapat menampung sebagian data pada level tinggi. Kebijakan penggantian mengatur item data mana yang harus disimpan di cache saat mereka diakses oleh processor. Dengan kata lain, item-tem pada cache yang belum pernah digunakan terkadang dikeluarkan dan digantikan dengan item baru. Saat processor melakukan akses terhadap sistem memori, alamat pada memori yang sedang dirujuk pertama kali dipetakan ke *cache line.* Pada kasus yang paling sederhana dapat dituliskan dengan persamaan matematis sebagai berikut:

cache line = address mod cache size



Gambar 1 (Struktur cache)

Contoh struktur cache dengan delapan line dan empat elemen per line ditunjukkan dengan gamabar 1.

Persamaan matematik di atas berlaku walaupum untuk skema pemetaan yang lebih kompleks. Hal yang perlu diperhatikan juga adalah karena cache lebih kecil dari memori utama, skema pemetaan menyebabkan efek "*wraparound*" yang berarti lebih dari alamat dapat dipetakan pada *line* yang sama. Sebuah *cache line* akan mengandung tiga informasi utama, yaitu: *cache line* memiliki informasi yang valid di dalamnya, tag cache yang menjelaskan apa yang terkandung pada *line,* dan data cache yang memegang isi memori saat itu. Pada seluruh cache terutama cache berukuran kecil, *cache line* dapat memuat data dari beberapa alamat memori yang berbeda dalam *field* muatan yang sama. Mungkin juga terjadi bahwa seluruh alamat dipetakan ke *line* yang sama dan akan dimuat pada *line* tersebut. Hal ini memungkinkan blok-blok dengan ukuran yang lebih besar ditransfer dari dan ke memori utama secara bersamaan yang lebih efisien daripada melakukan transfer kecil-kecilan dengan sering. *Field* tag cache memungkinkan kekompleksitasan dalam skema pemetaan dengan menawarkan identifikasi secara unik sehingga selalu dapat mencocokan sebuah alamat dengan isi *cache line* yang telah dipetakan.

Uraian di atas dapat dituliskan dengan persamaan matematik seperti di bawah ini:

Cache element = (address) mod line size

Cache line = (address>>log2(line size)) mod cache size

Cache tag = (address>>(log2(line size)+(log2(cache size))))

Lambang >> adalah pergeseran ke kanan bit-wise. Yaitu jumlah elemen yang dihitung dengan memilih bit-bit nol dan salah satu alamat dengan bit dua, tiga, dan empat menunjukkan nomor line. Sedangkan tag cache dihitung dengan memilih seluruh bit-bit lain kecuali bit-bit yang telah digunakan untuk menunjukkan nomor line. Dari persamaan matematis di atas dapat menunjukkan bahwa kombinasi nomor elemen, nomor line, dan tag dapat membedakan pemetaan dua alamat. Contohnya adalah pemetaan di bawah ini yang merupakan hasil dari persamaan di atas:

Address = 0b00001101 -> tag 0b000, line 0b011, elemen 0b01.

Address = 0b00000000 -> tag 0b000, line 0b000, elemen 0b00.

Address = 0b00100000 _-> tag 0b001, line 0b0000, elemen 0b00.

Kita dapat melihat jika alamat 0b00001101 dipetakan ke line tiga dan nilai dari elemennya adalah satu. Sedangkan dua alamat lainnya dipetakan keduanya ke line nol, tetapi dua alamat tersebut dapat dibedakan dari nomor tagnya.

Saat processor mengakses memori melalui cache dengan menggunakan pemetaaan dengan bentuk seperti yang telah dicontohkan di atas, akan terkadi suatu permasalahan. Jika line mengandung data yang invalid, maka kita perlu untuk memuat isi memori dari memori utama karena cache tidak memuatnya. Jika tag cache pada cache yang telah dihitung tidak cocok dengan tag alamat, kita juga perlu untuk memuat isi memori karena line tidak memuat informasi yang sedang dicari. Kedua permasalahan di atas disebut dengan cache miss karena cache tidak mengandung data yang diinginkan, jadi data tersebut harus diambil dari memori. Jika line memuat data yang valid dan tag-tag yang cocok dengan alamat, maka cache akan ditandai karena data yang diinginkan ada dan dapat digunakan tanapa diurut ulang di memori utama yang lebih lambat. Jika operasi cache yang berhasil lebih banyak dari cache miss, maka operasi sistem secara keselurhan diakselerasikan karena akses ke memory yang lebih lambat lebih sedikit dan lebih banyak akses ke cache.

2.2 Asumsi Serangan

Cara naif untuk melakukan pendekatan memanfaatkan perilaku cache adalah pembuatan database yang mencocokkan keterangan keberhasilan/kegagalan (miss) yang dikumpulkan dari serangan dengan keterangan yang telah dihitung sebelumnya, terhadap enkripsi sebuah plainteks dengan semua kunci yang mungkin. Walaupun serangan seperti ini akan berakhir sukses,

jumlah data yang disimpan dan penggunaan daya listrik untuk komputasi serangan seperti ini sangatlah besar.

Untuk memperbaiki pendekatan yang naif di atas, kita fokuskan pekerjaan kita untuk menganalisis pada ilmu dan algoritma kriptografi yang sedang diserang. Analisis ini akan membentukk hubungan antara bagian-bagian dari informasi rahasia dengan perilaku cache saat algoritma kriptografi dijalankan. Hubungam di atas dapat dimanfaatkan untuk melemahkan cipher, atau mungkin secara langsung menghasilkan kunci rahasia, dengan pemrosesan lebih lanjut setelah mendapatkan keterangan tentang cache. Saat mengkaji teknik-teknik ini, diperlukan beberapa asumsi tentang kemampuan si penyerang dan komposisi alat yang sedang diserang.

Pertama, kita mengasumsikan bahwa alat kriptografi yang sedang diserang mengakses memori utama melalui memori cache yang konvesional. Contoh serangan yang akan dijelaskan didasarkan pada kemampuan untuk mendeteksi *cache hit* (berhasil) dan *cache miss* (gagal) lalu menyimpannya dalam sebuah keterangan sehingga alat kriptografi yang mengakses memori secara langsung tanpa menggunakan cache tidak akan dibahayakan dengan teknik ini.

Kita juga mengasumsikan bahwa seorang penyerang memiliki kemampuan untuk menangkap informasi tentang penggunaan cache menggunakan metode yang telah dipahami secara baik. Kita membayangkannya bahwa hal tersebut dapat dilakukan dengan mengamati fitur-fitur yang telah diketahui berkaitan dengan daya listrik atau keterangan elektromagnetik dengan cara yang sama dengan *Simpel Power Analysis* menyerang algoritma kriptografi berbasis eksponensial. Dengan mengamati perbedaan aktivitas antara *cache hit* dan *cache miss,* penyerang dapat membuat sebuah keterangan yang menjelaskan kejadian-kejadian apa saja yang terjadi pada cache untuk beberapa akses yang dilakukan. Kita juga mengasumsikan pengetahuan mengenai struktur cache, contohnya ukuran *line*, dan membuat serangan berdasarkan fakta-fakta tersebut. Walaupun serangan perlu berubah berdasarkan atribut-atribut memori cache pada alat sesungguhnya, sedangkan prisnsip umum lainnya serangan tetaplah sama.

Karena contoh serangan yang akan dijelaskan didasarkan pada akses-akses terhadap strukr-struktur S-box yang ditemukan pada banyak *block cipher*, maka diasumsikan bahwa nilai dari S-box diambil dari sebuah tabel di memori dan tidak dihitung secara langsung menggunakan instruksi-instruksi processor yang diimplementasikan untuk pemetaan. Hal ini berarti akses ke memori akan berjalan melalui cache dan menghasilkan sebuah ketrangan mengenai akses yang tanpa keterangan ini serangan-serangan yang dilakukan menjadi tidak berguna. Asumsi ini pada dasarnya berarti kita perlu berhubungan dengan sistem memori deterministik.

Asumsi lainnya adalah bahwa cache dikosongkan pada saat alat kriptografi dimatikan dan algoritma kriptografi (operasi) dimulai dengan cache yang kosong, dalam artian dari data yang berhubungan dengan algoritma kriptografi. Asumsi yang terakhir adalah bahwa kita dapat mencocokkan informasi yang didapat dengan kemampuan menangkap informasi yang dimiliki dengan operasi-operasi pada *source code*. Yaitu saat terjadi akses ke memori melalui cache, kita dapat memeberitahu struktur data yang mana pada algoritma kriptografi yang sedang diakses.

Walaupun sepertinya asumsi-asumsi yang telah dibuat membatasi secara luas manfaat dari serangan pada alat sungguhan. Namun fakta yang terjadi adalah perilaku memori cache pada alat kriptografi akan menghasilkan setidaknya beberapa informasi yang berguna . Serangan-serangan akan selalu perlu dirancang untuk keadaan tertentu, tetapi secara umum,makin banyak informasi akan selalu lebih baik dari kacamata penyerang.

Pada contoh serangan yang akan dijelaskan, diasumsikan bahwa keberadaan sebuah processor yang akan memberikan sedikit hasil dari akses-akses terhadap cache. Untuk melakukan simulasi informasi yang bisa didapatkan seorang penyerang dari alat kriptografi, pustaka simulasi cache Dinero digunakan yang telah dikonfigurasi ke model kilobyte, empat byte per line, variansi

pemetaan langsung dari apa yang ditemukan langsug pada ini processor. Simulator cache menghasilkan daftar akses-akses setiap penjalanan algoritma dimana setiap akses pada daftar tersebut berhubungan dengan struktur S-box yang sedang dibaca.

2.3 Contoh Serangan

Contoh serangan dilakukan terhadap *Data Encryption Standard* (DES) yang inti dari algoritmanya dapat dituliskan dengan pseudeo code pada gambar 2. Pada makalaih ini notasi yang digunakan dalam pembahasan seranagn terhadap DES adalah ID, E, P dam FD yang masing-masing melambangkan Initial Data, Expansion, P-Box, dan Final Datahasil permutasi. Juga terdapat notasi J, IK, Cyang melambangkan key rotation, Initial Key, dan Compression Permutations. Sedangkan S melambangkan S-box secara keseluruhan atau transformasi substitusi dan SBn melambangkan akses terhadap S-box. Hal yang perlu diperhatikan juga adalah asumsi bahwa struktur-struktur S-box telah diatur kembali untuk cara indexing array yang mudah yang biasa ditemukan pada implementasi perangkat lunak DES.

Ada beberapa notasi tambahan untuk memilih dan melakukan konkat nilai-nilai bit dari dan ke nilai yang lebih besar. Notasi x[y..z] melambangkan pemilihan nomor bit-bit di antara index y dan z, dari nilai x dan membentuk kembali nilai baru. Sedangkan notasi x[y,z] berarti pemilihan individu bit-bit dari index y sampai z dan membentuk individu-individu tersebut ke dalam nilai baru. Operasi @ melambangkan konkatenasi, atau kombinasi ulang, mengelompokkan bit-bit ke nilai yang lebih besar.

```
void des( D, K[] )
{
  D = ID( D );

  L' = D[ 63 .. 32 ];
  R' = D[ 31 .. 00 ];

  for( round = 0; round < 16; round++ )
  {
    L  = R';
    R  = E( R' );

    R  = R ^ K[ round ];

    R  = SB0[ R[ 47 .. 42 ] ] @ SB1[ R[ 41 .. 36 ] ] @
         SB2[ R[ 35 .. 30 ] ] @ SB3[ R[ 29 .. 24 ] ] @
         SB4[ R[ 23 .. 18 ] ] @ SB5[ R[ 17 .. 12 ] ] @
         SB6[ R[ 11 .. 06 ] ] @ SB7[ R[ 05 .. 00 ] ];

    R  = P( R );

    R  = L' ^ R;

    L' = L;
    R' = R;
  }

  D = FD( R' @ L' );
}
```

Gambar 2 (pseudo-code DES)

Tujuan dari serangan yang dilakukan terhadap algoritma DES ini adalah untuk mengungkap kunci utama yang disimpan sebagai informasi rahasia pada alat yang diserang dan digunakan untuk membangkitkan jadwal kunci untuk operasi enkripsi dan dekripsi. Pengetahuan tentang kunci tersebut akan memungkinkan penyerang untuk membuat alat serupa yang dapat digunakan untuk tujuan kecurangan karena alat serupa tersebut akan mereplikasi perilaku sebenarnya dari alat kriptografi yang sedang diserang.

2.3.1 Formulasi

Persamaan matematik dari sebuah serangan terhadap DES mengasumsikan pengetahuan yang memadai baik mengenai algoritma alat kriptografi yang sedang diserang maupun perilaku operasional dan memori cache. Penyerang hanya perlum mempertimbangkan dua putaran pertama dari DES yang digambarkan dengan diagram alir pada gambar 3. Dari diagram ini, penyerang dapat mudah menulusuri aliran informasi melalui algoritma dan menunjukkan bagaiman nilai tertentu dihitung dari nilai sebelumnya.

Fungsi S0 dan S1 serta diagram melambangkan proses transformas-transformasi substitusi pada putaran ke-0 dan pertama secara berurutan yang dimplementasikan dengan akses-akses terhadap struktur-struktur S-box di memori utama. Kita dapat menulusuri dari diagram

untuk menghasilkan dua persamaan untuk index-index I0 dan I1, dan yang melakukan akses-akses terhadap S-box dalam proses transformasi-transformasi S0 dan S1.

$$I_0 = K_0 \oplus E_0(R_0)$$
$$I_1 = K_1 \oplus E_1(L_0 \oplus P_0(S_0(K_0 \oplus E_0(R_0))))$$

Persamaan 1

Persamaan untuk I0 dan I1 memegang *subset* bit-bit yang berbeda untuk masing-masing di S-box untuk setiap transformasi. Contohnya, S-box ketujuh diakses dengan menggunakan bit-bit nol sebanyak lima sampai index yang dimasukkan. Karena ketergantungan pada perilaku cache pada setiap akses terhada S-box, penyerang dapat menghubungkan satu index dengan index lainnya. Jika kita menangkap perilaku saat S-box ketujuh sedang diakses dalam transformasi-transformasi S0 dan S1 dengan mempertimbangkan sub-set bit-bit yang tepat yang digunakan untuk index-index, kita akan menghasilkan persamaan matematik seperti di bawah ini:

$$I_0[05..02] = I_1[05..02]$$

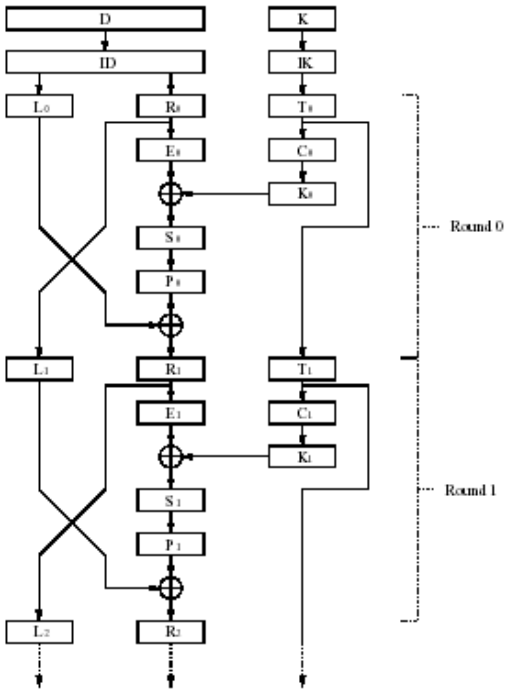
saat S1 menghasilkan cache hit dan

$$I_0[05..02] \neq I_1[05..02]$$

saat S1 menghasilkan cache miss.

Persamaan 2

Dimana =LINE dan ≠LINE adalah ekivalensi-ekivalensi cache line yang masuk dalam perhitungan karena fakta yang terjadi adalah setiap cache line dapat menyimpan lebih dari satu item data, indeks-indeks dapat dihasilkan dari cache line yang sama yang sedag diakses tanpa nilai indeks yang sama. Tag cache daril alamat I0 dan I1, yang berarti pemetaan terhadap cache line dapat bernilai sama tanpa nilai indeks I0 dan I1 yang berbeda.



Gambar 3 (Diagram alir DES)

Pada persamaan 2 secara sederhana berarti jika penyerang berkonsentrasi pada profil cache pada satu S-box pada putaran pertama dan menghasilkan cache hit (berhasil), penyerang dapat mengetahui bahwa indeks I0 dan I1 pasti dipetakan ke line yang sama di cache. Penyerang dapat memaksakan line-line cache untuk menghasilkan persamaan-persamaan yang langsung sama, tergantung dari ukuran cache line. Kita pertimbangkan yang memiliki ukuran line empat yang berarti setiap cache line dapat memuat empat elemen S-box. Hal ini berarti bahwa dua bit terbawah dari segala akses akan digunakan sebagai pemilih pada cache line yang dimaksud daripada mengubah pemetaan rujukan cache line. Kita dapat menuliskan pengetahuan di atas ke dalam persamaan yang dijelaskan dalam persamaan 3:

$$I_0[05..02] = I_1[05..02] \quad \text{when } S_1 \text{ yields a cache hit}$$
$$I_0[05..02] \neq I_1[05..02] \quad \text{when } S_1 \text{ yields a cache miss}$$

Persamaan 3

Dengan persamaan di atas, kita dimungkinkan untuk secara langsung menghubungkan nilai dari I0 dan I1 satu sama lain dengan sebuah kejadian cache hit atau cache miss pada akses-akses terhadap S-box, pada kasus ini adalah S-box ketujuh. Kita lanjutkan analisis dengan menggunakan pemilihan bit pada indeks-indeks yang dimaksud untuk menguak lapisan-lapisan permutasi untuk menyederhanakan persamaan-persamaan untuk I0 dan I1, misalkan dengan persamaan sebagai berikut:

$$P_0(X)[04..01]$$

Persamaan 4

Kita dapat menulusuri bit-bit yang kita perlukan pada output melalui permutasi untuk mendeduksi bit-bit dari tempat bit-bit tersebut datang:

$$P_0(X)[04..01] = X[26, 10, 21, 28]$$

Persamaan 5

Contoh di atas menunjukkan bahwa jika kita melaukan permutas P0 kepada nilai X dan hanya berfokus pada sub-set bit-bit pada output, dimana kita memerlukan nilai X untuk dipertimbangkan ditempatnya. Dengan menggunakan teknik ini kepada persamaan yang kita miliki untuk I0 dan I1 kita dapat menyederhanakannya menjadi sebagai berikut:

$$\begin{aligned} I_0 &= (K_0 \oplus E_0(R_0))[05..02] \\ &= K_0[05..02] \oplus E_0(R_0)[05..02] \\ &= K_0[05..02] \oplus R_0[04..01] \end{aligned}$$

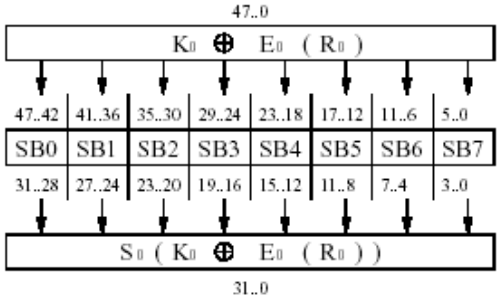
$$\begin{aligned} I_1 &= (K_1 \oplus E_1(L_0 \oplus P_0(S_0(K_0 \oplus E_0(R_0)))))[05..02] \\ &= K_1[05..02] \oplus E_1(L_0 \oplus P_0(S_0(K_0 \oplus E_0(R_0))))[05..02] \\ &= K_1[05..02] \oplus (L_0 \oplus P_0(S_0(K_0 \oplus E_0(R_0))))[04..01] \\ &= K_1[05..02] \oplus L_0[04..01] \oplus P_0(S_0(K_0 \oplus E_0(R_0)))[04..01] \\ &= K_1[05..02] \oplus L_0[04..01] \oplus S_0(K_0 \oplus E_0(R_0))[26, 10, 21, 28] \end{aligned}$$

Persamaan 6

Kunci dari penggunaan persamaan-persamaan di atas terletak pada instansiasi dari transformasi S0 yang merepresentasikan akses-akses S-box pada putaran ke-0. Hal yang penting untuk diperhatikan dalam kasus ini adalah perilaku cache yang tidak relevan dan kita hanya tertarik pada nilai hasil dari transformasi pada persamaan di atas. Karena kita mengetahui bit-bit keluaran yang kita perhatikan, kita dapat menjabarkan persamaan sebelumnya dengan menggunakan pengetahuan aliran data melalui S-box yang digambarkan pada gambar 4. Kita dapat melihat dari gambar tersebut bahwa keluaran dari transformasi S-box dengan bit-bit keluaran yang kita perhatikan, dihasilkan dari keluaran akses S-box yang berbeda-beda. Maka kita rumuskan persamaan menjadi:

$$\begin{aligned} S_0(X)[26] &= SB1(X[41..36])[02] \\ S_0(X)[10] &= SB5(X[17..12])[02] \\ S_0(X)[21] &= SB2(X[35..30])[01] \\ S_0(X)[28] &= SB0(X[47..42])[00] \end{aligned}$$

Persamaan 7



Gambar 4 (Diagram alir S-box DES pada putaran ke-0)

Dari persamaan tujuh kita dapat mengubah persamaan I0 dan I1 menjadi sebagai berikut:

$$\begin{aligned} I_0 &= K_0[05..02] \oplus R_0[04..01] \\ I_1 &= K_1[05..02] \oplus L_0[04..01] \oplus Z \end{aligned}$$

$$\begin{aligned} Z = {} & SB1((K_0 \oplus E_0(R_0))[41..36])[02] \oplus \\ & SB5((K_0 \oplus E_0(R_0))[17..12])[02] \oplus \\ & SB2((K_0 \oplus E_0(R_0))[35..30])[01] \oplus \\ & SB0((K_0 \oplus E_0(R_0))[47..42])[00] \end{aligned}$$

Persamaan 8

Dengan menulikan seluruh instansiasi dari K0 dan K1 kita dapat menemukan:

$$\begin{aligned} K_0[05..02] &= K[19,50,51,02] \\ K_1[05..02] &= K[27,58,59,10] \end{aligned}$$

$$\begin{aligned} K_0[41..36] &= K[31,07,62,55,45,22] \\ K_0[17..12] &= K[17,34,59,11,41,35] \\ K_0[35..30] &= K[61,29,38,39,20,06] \\ K_0[47..42] &= K[54,13,30,04,15,47] \end{aligned}$$

Persamaan 9

Lalu kita dapat mensubstitusikan persamaan 9 dengan persamaan 8 dengan menghilangkan segala operasi-operasi permutasi yang tersisa sehingga membentuk persamaan akhir sebagai berikut:

$$\begin{aligned} I_0[05..02] &= K[19,50,51,02] \oplus R_0[04..01] \\ I_1[05..02] &= K[27,58,59,10] \oplus L_0[04..01] \oplus Z \end{aligned}$$

Persamaan 10

dimana

$$\begin{aligned} Z = {} & SB1(K[31,07,62,55,45,22] \oplus R_0[31,39,47,55,63,05])[02]@ \\ & SB5(K[17,34,59,11,41,35] \oplus R_0[27,35,43,51,59,01])[02]@ \\ & SB2(K[61,29,38,39,20,06] \oplus R_0[63,05,13,21,29,37])[01]@ \\ & SB0(K[54,13,30,04,15,47] \oplus R_0[57,07,15,23,31,39])[00] \end{aligned}$$

Dari hasil akhir di atas dan hubungan antara I0 dan I1 yang dijelaskan pada persamaan 3, kita telah membuat persamaan yang mengikat bagian-bagian yang berbeda dari bahan-bahan kunci untuk masukan tertentu secara bersamaan. Saat indeks-indeks untuk pengkasesan-pengkasesan terhadap S-box pada putaran ke-0 dan putaran pertama bernilai sama (terdapat catatan mengenai cache hit untuk akses terhadap S-box putaran pertama), persamaan di atas memberikan tes persamaan yang dapat kita manfaatkan untuk memeriksa validitas kombinasi bit bahan-bahan material kunci. Komposisi dari tes persamaan tersebut akan bergantung pada S-box yang melakukan pembangkitan persamaan untuk I0 dan I1 tetapi akan membentuk persamaan secara umum seperti perrsamaan di bawah ini:

$$K[x..y] \oplus R_0[x..y] = K[x..y] \oplus L_0[x..y] \oplus Z$$

Dimana

$$\begin{aligned} Z = {} & SBn(K[x..y] \oplus R_0[x..y])[z]@ \\ & SBn(K[x..y] \oplus R_0[x..y])[z]@ \\ & SBn(K[x..y] \oplus R_0[x..y])[z]@ \\ & SBn(K[x..y] \oplus R_0[x..y])[z] \end{aligned}$$

Persamaan 11

Karena kita dapat mengendalikan nilai dari D, juga L0 dan R0, maka kita dapat memaksakan pembentukan hubungan-hubungan ke tes dengan memilih nilai R0 dan mencari nilai dari L0 yang mengakses dalam perhitungan satu putaran akan menghasilkan cache hit. Nilai dari L0 akan dipanggil yang memicu cache hit yang dimaksud dengan nilai tetap R0. Dengan menggunakan teknik ini dan memilih secara acak nilai dari R0, kita dapat membuat tes sebanyak yang kita inginkan dengan melakukan query-query enkripsi ke alat yang akan diserang, lalu kita menangkap profil cache dan melakukan tes apakah L0 memicu cache hit pada putaran pertama saat S-box yang dipilih diakses.

Inti dari pelaksanaan serangkaian tes di atas adalah jika kita dapat memanfaatkan serangkaian tes tersebut untuk melakukan pencarian sub-key yang efisien yang akan memberi kita sedikit nilai-nilai kandidat sub-key untuk masing-masing S-box. Misalnya dengan nilai akhir untuk S-box tujuh pada persamaan 10 sebelumnya, kita dapat memilih nilai R0 dengan nol, setelah mencari nilai tetap L0 0x0E, lakukan tes seperti sebagai berikut:

$$K[19,50,51,02] = K[27,58,59,10] \oplus 0x0E \oplus Z$$

dimana

$$\begin{aligned} Z = {} & SB1(K[31,07,62,55,45,22])[02]@ \\ & SB5(K[17,34,59,11,41,35])[02]@ \\ & SB2(K[61,29,38,39,20,06])[01]@ \\ & SB0(K[54,13,30,04,15,47])[00] \end{aligned}$$

Persamaan 12

Jika kita melakukan pencarian secara *brute force* dengan 31 kombinasi bit-bit kunci yang unik, dengan bit 59 telah digunakan dua kali , hanya sedikit proporsi dari kombinasi-kombinasi bit akan valid dalam persmaan ini. Karena kita dapat melakukan banyak tes terhadap S-box, dengan nilai pasangan R0 dan L0 yang berbeda-beda untuk setiap tesnya, maka kita dapat meyakinkan hanya sedikit dari kombinasi tersebut yang terhitung valid.

2.3.2 Implementasi

Misalkan kita akan melakukan serangan terhada kunci rahasi K = 0x012345679ABCDEF dan jika kita mengeluarkan bit-bit parity untuk kejelasan K akan bernilai sama dengan K= 0x0022446688AACCEE. Hal yang penting untuk diperhatikan bahwa semua kunci valid mungkin telah digunakan dan keberadaan bit-bit parity tidak berpengaruh kepada ketepatan kinerja serangan karena bit-bit parity tidak ada pada tes yang dilakukan.

Serangan dimulai dengan membangkitkan 32 tes kepada S-box nol, denagan memilih nilai R0 secara acak dan mencari nilai tetap L0 yang berkaitan. Setelah mencari 32 pasangan nilai tetap secara acak dan lalu membuat bilangan yang sama untuk persamaan, kita mengurangi jumlah kandidat sub-kunci dengan mencari pada bit-bit kunci yang telah digunakan dan menggunakan semua tes-tes persamaan ke semua kombinasi.

Selanjutnya kita akan mendapatkan delapan kombinasi sub-kunci yang lolos 32 tes, di mana Ci,j melambangkan kandidat kunci nomor j yang didapatkan dari putaran ke-i dari metode serangan.

$$C_{0,0} = 0x00024420C88024AA$$
$$C_{0,1} = 0x00024420C88034BA$$
$$C_{0,2} = 0x00024420C8A004AA$$
$$C_{0,3} = 0x00024420C8A014BA$$
$$C_{0,4} = 0x00024460888024AA$$
$$C_{0,5} = 0x00024460888034BA$$
$$C_{0,6} = 0x0002446088A004AA$$
$$C_{0,7} = 0x0002446088A014BA$$

Persamaaan 13

Ternyata dengan menjalani 32 tes kepada setiap kombinasi sub-kunci menjadi berlebihan karena hasil yang sama bisa didapatkan dengan hanya menjalani sembilan tes. Untuk bergerak mendekati hasil yang diinginkan, kita akan melakukan 32 tes lagi pada putaran kedua. Tes kali ini dilakukan pada S-box satu. Pencarian dengan tes ini kembali mengungkapkan 31 bit untuk ruang kunci dan menghasilkam delapan kandidat sub-kunci.

$$C_{1,0} = 0x000200468000A086E$$
$$C_{1,1} = 0x000200468000A88EE$$
$$C_{1,2} = 0x400200468000A084E$$
$$C_{1,3} = 0x400200468000A88CE$$
$$C_{1,4} = 0x808200468000A086E$$
$$C_{1,5} = 0x808200468000A88EE$$
$$C_{1,6} = 0xC08200468000A084E$$
$$C_{1,7} = 0xC08200468000A88CE$$

Persamaan 14

Sekarang seluruh paket tes perlu dilakukan unruk mempersempit kandidat sub-kunci agar jumlahnya cukup sedikit untuk diatur. Karena kandidat sub-kunci dari setiap putaran tes menggunakan bit-bit kunci yang berbeda, maka sewajarnya kita harus mempertimbangkan semua kombinasi operasi logika terhadap semua kandidat.

$$
\begin{array}{ccc}
C_{0,0} & \vee & C_{1,0} \\
C_{0,0} & \vee & C_{1,1} \\
 & \vdots & \\
C_{0,7} & \vee & C_{1,6} \\
C_{0,7} & \vee & C_{1,7}
\end{array}
$$

Persamaan 15

Tetapi karena kita mengetahui bahwa kedua putaran saling tumpang tindih dalam 17 bit penggunaan kunci, maka kita dapat mengurangi banyak kombinasi operasi logika. Hal ini ini mungkin dilakukan memperhatikan kumpulan sub-kunci lalu membuang yang tidak sejalan dengan penggunaan 17 bit pada sub-kunci yang tumpang tindih. Jika C0,i dan Ci,j melambangkan dua kandidat sub-kunci pada putaran ke-0 dan pertama dan M0 melambangkan *mask* dari bit-bit yang tumpang tindih pada kedua putaran tes, jika:

$$(C_{0,i} \oplus C_{1,j}) \wedge M_0 = 0$$

Persamaan 16

Dan kombinasi sub-kunci di bawah ini diterima sebagai kandidat yang valid.

$$C_{0,i} \vee C_{1,j}$$

Persanaan 17

Setelah menghitung mask M0 yang melambangakan bit-bit yang tumpang tindih, dan melakukan tes persamaan 16 dengan nilai yang didapatkan dari persamaan 13 dan 14, kita dapat mempersempit jumlah kandidat yang potensial menjadi empat yang dituliskan pada persamaan 19.

$$M_0 = \texttt{0xC0821840840000AA}$$

Persamaan 18

$$
\begin{aligned}
C_{2,0} &= \texttt{0x000244668888AACEE} \\
C_{2,1} &= \texttt{0x000244668888ABCFE} \\
C_{2,2} &= \texttt{0x000244668888AA9CEE} \\
C_{2,3} &= \texttt{0x000244668888AA9CFE}
\end{aligned}
$$

Persamaan 19

Pada tahapan serangan saat ini, kita mungkin menghadapi putaran tes-tes selanjutnya untuk menyempitkan kandidat kunci-kunci menjadi hasil akhir. Tetapi setelah menjalani tes putaran ke-0 dan pertama, kita telah menemukan 49 dari 56 bit-bit parity dan mengetahui bahwa tes putaran selanjutnya akan menemukan sedikit lagi bit-bit yang diperlukan untuk menemukan hasil akhir. Pada putaran ke-0 dan pertama, pendekatan brute force akan memakan banyak biaya. Sedangkan pada tes putaran selanjutnya akan memakan biaya yang tidak sebanyak pada putaran sebelumnya, maka pendeketan brute force akan digunakan pada tahapan ini.

Jika Mu melambangkan mask untuk bit-bit kunci, Mu perlu dimasukkan dalam pencarian karena belum mengungkap kunci yang dicari pada putaran sebelumnya. Nilai dari Mu adalah sebagai berikut:

$$M_u = \texttt{0x0630800000044000}$$

Persamaan 20

Lalu kita dapat melakukan tes di bawah ini untuk menyempitkan kandidat-kandidat kunci ke hasil akhir:

$$DES(P, K) = DES(P, C_{2,i} \vee \overline{M_u})$$

Persamaan 21

Jika cipherteks diproduksi dengan melakukan enkripsi pada plainteks yang telah diketahui sebelumnya dengan kunci rahasia sama dengan cipherteks diproduksi dengan enkripsi pada plainteks yang telah diketahui dengan rancangan kandidat kunci, maka kita telah menemui hasilnya. Pada tahapan ini, pertama kali kita menggunakan pasangan plainteks dan cipherteks alat kriptpgrafi yang sedang diserang untuk membandingkan hasil dalam

pencarian yang sedang dilakukan. Sehingga akhirnya kita akan menemukan hasil akhirnya yaitu kunci seperti sebgai berikut:

$$C_{3,0} = 0x0022446688AACCEE$$

Persamaan 22

2.3.4 Efisiensi

Secara umum metode serangan yang digunakan memakan biaya lebih hemat secara signifikan dari pada menggunakan pendeketan brute force.Pada tahapan pertama, metode serangan mengumpulkan informasi cache alat kriptografi yang sedang diserang untuk membangun persamaan-persamaan. Jika kita bermaksud melakukan tes sebanyak *r* putaran yang masing-masing putaran terdiri dari *t* putaran terhadap setiap kombinasi kunci, maka kita perlu melakukan tes pasling banyak sebanyak:

$$\alpha = r \times t \times 2^4$$

Sejumlah -operasi DES sebanyak α pada alat kriptografi yang diserang untuk mengungkap nilai tetap dan kemudian membuat persamaan. Jika hal ini telah dilakukan, kita akan masuk ke tahap pemcarian yang merupakan tahap paling memakan biaya. Secara keseluruhan kita perlu melakukan tes sebanyak β:

$$\beta = r \times t \times 2^{31}$$

Hal ini disebabkan semua tes pada S-box akan menggunakan paling banyak 31 bit kunci yang unik. Terakhir kita harus menyeleksi kandidat kunci dari tahapan sebelumnya untuk menemukan bit-bit kunci yang belum terungkap dengan pendekatan brute force. Pada tahapan ini, kita akan melakukan operasi DES sebanyak γ:

$$\gamma = c \times m$$

Di mana c adalah jumlah kandidat kunci yang dihasilkan dari tahapan sebelumnya dan m adalah bilangan konstan yang melambangkan jumlah bit yang belum terungkap dari tahapan sebelumnya. Biasanya baik c dan m akan bernilai kecil sehingga tahap akhir tidak akan terlalu memakan biaya seperti pada tahapan sebelumnya. Misalkan kita memilih r = 2 dan t = 32 yang berarti terdapat dua putaran dan 32 tes terhadap setiap kombinasi kunci pada setiap putaran. Karena pada tes S-box 0 dan S-box 1 terdapat 13 bit yang sama digunakan, maka pada tahapan ini 49 bit dari kunci telah terungkap. Sisa 7 bit bit non parity yang diperlukan akan dicari melalu pendekatan brute force sebagai tahapan akhir dari serangan. Serangan dilakukan dengan rician jumlah operasi sebagai berikut:

$$\begin{aligned} \alpha &= 2 \times 32 \times 2^4 \\ &= 2^{10}\text{DOPS} \\ \beta &= 2 \times 32 \times 2^{31} \\ &= 2^{37}\text{TOPS} \\ \gamma &= 4 \times 2^{56-(31+31-13)} \\ &= 2^9\text{DOPS} \end{aligned}$$

Kita secara sengaja memilih untuk melakukan tes sebanyak 32 kali terhadap setiap kombinasi kunci karena sebenarnya hampir sama dengan melakukan satu kali proses DES dari awal sampai akhir. Karena 32 operasi tes sebnding dengan sekali operasi DES, maka total operasi DES yang dijalani sebanyak:

$$2^{10}\text{DOPS} + 2^{32}\text{DOPS} + 2^9\text{DOPS} \simeq 2^{32}\text{DOPS}$$

Hasil di atas menunjukkan bahwa dengan bantuan informasi dari cache dari processor yang diserang, 56 bit kunci DES adalah setara dengan 32 bit kunci DES dalam masalah keamanan. Salah satu hal yang perlu diperhatikan adalah bahwa hanya tahapan pertama saja yang termasuk proses on-line. Artinya penyerang hanya akses terhadap alat kriptografi yang diserang selama waktu yang diperlukan proses pada tahapan pertama untuk membangkitkan nilai tetap untuk operasi-operasi tes persamaan. Dalam hitungan operasi berarti waktu akses yang diperlukan adalah selama 2^10 operasi DES. Sedangkan tahapan berikutnya yang banyak memakan biaya sekitar 2^32 operasi DES dapat dilakukan secara off-line dalam waktu lenggang yang dimiliki penyerang dan tanpa memerlukan akses terhadap alat kriptografi yang diserang.

Hal menarik yang perlu dicatat adalah pentingnya minimasi operasi serangan secara on-line sebagai faktor keberlangsungan Side Channel Analysis Attack karena tanpa operasi pengumpulan profil yang hemat biaya, maka akses terhadap alat kriptografi yang bermanfaat akan menjadi sangat terbatas. Perbandingan antara karakteristik biaya proses off-line dengan on-line pada serangan ini dapat dianggap sebagai keuntungan dibandingkan teknik-teknik laini yang memerlukan lebih banyak proses on-line. Karena kita memiliki waktu dan sumber daya yang lebih banyak dalam melakukan proses off-line, sangatlah masuk akal untuk melakukan serangan yang memiliki karakteristik sejenis.

3. Penanganan yang dapat dilakukan terhadap *Cache Based Attack*

Walaupun Cache Based Attack mungkin menawarkan metode-metode yang efektif dalam menyerang keamanan algoritma-algoritma kriptografi, terdapat beberapa penanganan yang dapat dilakukan untuk membatasi kerusakan yang dapat diakibatkan oleh serangan. Cara terbak untuk mencegah serangan terhadap cache adalah dengan memindah cache itu sendiri dari rancangan processor. Walaupun dalam pertimbangan antara biaya dan kinerja akibat pemidahan cache mungkin dapat diterima pada situasi tertentu, teknik ini menjadi cacat karena penurunan kinerja dari alat sangatlah berharga untuk diabaikan.

Ada beberapa metode-metode yang mungkin digunakan untuk melindungi alat dari cache based attack yang didasari pada modifikasi perangkat keras, pengubahan algoritma ataupun keduanya. Solusi idealnya adalah solusi yang pendekatannya masih mengutamakan kinerja tetapi tanpa permasalahan-permasalahan serangan yang memanfaatkan informasi dari alat. Penemuan mekanisme pertahanan terhadap cache based attack yang baru masih terbuka lebar melalui penelitian-penelitian di kemudian hari. Dari sekian banyak solusi yang ditawarkan, tidak ada satupun solusi tanpa pembuangan cache yang dapat menjamin secara penuh karena masing-masing solusi menawarkan tingkat keamanan yang probabilistik. Hal ini disebabkan utama karena asumsi penyerangan yang menyebutkan bahwa penyerang mampu membangun informasi dari perilaku cache daripada dengan menggunakan informasi statistik seperti waktu eksekusi. Walaupun serangan yang didasari pengumpulan profil lebih sulit untuk dilakukan, lebih memerlukan hardware yang lebih canggih, tetaplah mungkin jika terdapat model serangan yang lebih realistis dalam menyerang alat menjadi teknik serangan yang efektif. Jika kita membandingkan serangan berbasis konmsi waktu dengan perilaku cache, mekanisme pertahanan dapat dilakukan dengan relatif lebih mudah dengan mendeviasikan penggunaan waktu atau memasukkan operasi dummy untuk mengubah waktu eksekusi. Mekanisme pertahanan untuk cache based attack adalah dengan menyamarkan kemunculan dari perilaku cache, dengan cara yang sama akses terhadap register harus juga disamarkan untuk mengatasi Differential Power Analysis, yang pelaksanannya dalam praktiknya lebih sulit untuk dilakukan.

Mekanisme pertahanan yang menawarkan keamanan terbaik dibandingkan dengan rasio biayanya adalah pengurutan akses non-deterministik. Hal ini disebabkan karena mekanisme pertahanan ini menawarkan tingkatan perlindungan berdasarkan paramater-parameter yang dapat dihitung dan juga probabilistik seperti tingkatan penjadwalan non-deterministik di antara pengaksesan cache. Tetapi mekanisme pertahanan ini memerlukan perubahan perangkat keras yang signifikan yang dapat sangat memakan biaya dalam kaitannya dengan penggunaan cache untuk meningkatkan kinerja alat.

3.1 Pemanasan acak atau penuh pada cache

Untuk mengubah keterangan/profil tentang cache hit dan cache miss di antara operasi-operasi pada algoritma, kita dapat mengubah source code algoritma yang melakukan pemanasan terhadap cache dengan mengisi cache dengan data. Pengisian data dapat dimulai dari pengisian penuh S-box ke cache sehingga akan terhindar dari cache miss yang merupakan informasi yang diperlukan sampai dengan pengisian elemen S-box secara acak ke cache sehingga informasi yang diperoleh dari cache akan berkurang keakuratannya. Contohnya kita dapat mengubah keterangan/profil cache hit dan cache miss ke

struktur S-box dengan melakukan pemanasan cache elemen acak dari struktur tersebut dengan menambah kode porgram dengan:

```
for( i = 0; i < warming factor; i++ )
{
  dummy = sbox[ random number % sbox size ]
}
```

Dengan melakukan operasi di atas pada awal algoritma, kita akan mengisi cache dengan elemen acak dari S-box mengubah profil mengenai cache hit dan cache miss pada cache tergantung pada elemen-elemen yang telah dimuat. Pendekatan ini terlihat mudah untuk dijalankan sebagai permulaan dari algoritma kriptografi yang sebenarnya daripada menggunakan teknik lain yang mungkin memerlukan perangkat keras tambahan. Tetapi mekanisme ini tidak memberikan jaminan perlindungan karena sebenarnya mekanisme ini hanya menyamarkan akses-akses sebenarnya dengan tingkat perlindungan tergantung pada ukuran S-box dan proporsi dari elemen-elemen S-box yang dimuat ke cache.

Untuk memberikan jaminan bahwa mekanisme ini merupakan pengananan yang efektif, kita perlu melakukan pemanasan dengan seluruh elemen dari S-box daripada hanya menggunakan sebagian elemen secara acak. Tetapi cara seperti ini sama buruknya dengan tidak menggunakan cache dalam masalah kinerja. Walaupun begitu, pemanasan penuh pada cache telah diimplementasikan dengan tidak merugikan pada beberapa algoritma seperti Khufu di mana S-box dihitung dengan pengaruh dari bahan-bahan kunci. Pada kasus ini, keseluruhan S-box akan tersentuh dan akan dimuat ke cache sebelum eksekusi algoritma kriptografi. Dengan kata lain asumsi bahwa penyerang dapat membangun informasi mengenai cache hit dan cache miss tidak lagi berlaku.

3.2 Efek Rapid Avalanche

Cara-cara konvensional atribut yang diharapkan pada fungsi hash dan rancangan cipher adalah pemanfaatan efek *avalanche* (longsor). Efek ini berarti bit apapun sebagai masukan harus dapat memberikan pengaruh sebanyak mungin terhadap ouput. Jika efek ini terjadi dengan cepat pada algoritma akan mengakibatkan analisis aliran menjadi sulit karena jumlah informasi yang tidak diketahui yang mempengaruhi tahapan-tahapan serangan yang ada sangat besar.

Permasalagan yang ditemukan pada DES yaitu 54 dari 56 bit pada kunci mempengarui perilaku algoritma pada putaran kedua. Hal ini secara signifikan membuat persamaan-persamaan untuk tes menjadi lebih kompleks karena akses cache ke struktur S-box manapun berarti memperhitungkan output dari empat S-box selanjutnya telah diberikan indeks dengan bahan-bahan kunci. Jika kasusnya tidak seperti ini, hubungan-hubungan yang akan menjadi lebih sederhana dan menghasilkan lebih sedikit informasi yang tidak diketahui sehingga akan menjadi serangan yang lebih efektif terhadap algoritma.

Karena pengaruh yang cepat dari efek avalanche akan membuat formasi hubungan-hubungan berdasarkan cache menjadi lebih rumit. Sehingga hal ini menjadi pertimbangan untuk dijadikan penanganan terhadap serangan, yaitu dengan mengkondisikan sistem kriptografi untuk melakukan efek avalanche dengan cepat.

3.3 Pengurutan akses non-deterministik

Processor non-deterministik dibuat dimaksudkan sebagai mekanisme pertahanan umum terhadap Side Channel Based Attack. Processor jenis ini memanfaatkan paralelisme tingkatan instruksi, yang kemudian akan diturunkan kepada implementasi algoritma kriptografi, untuk menjalankan instruksi-instruksi dengan urutan yang berbeda-beda untuk setiap run algoritma kriptografi dengan tetap menjaga keterkaitan di antara instruksi-instruksi tersebut. Karena processor akan menjalankan isntruksi-instruksi dengan urutan yang berbeda-beda pada setiap run algoritmanya, maka profil/keterangan penggunaan daya listrik akan berubah sehingga serangan akan lebih sulit untuk dilakukan.

Prinsip yang sama dapat diimplementasikan untuk membuat pertahanan terhadap cache based attack dengan mengizinkan akses terhadap memori terjadi di luar urutan seperti

ketidakurutan eksekusi instruksi-instruksi di register pada processor non-deterministik. Walaupun keterkaitan di antara instruksi-instruksi perlu diamati untuk mencegah proses kerusakan penulisan setelah pembacaan atau kerusakan penulisan setelah penulisan, serangkaian pembacaan memori berturut-turut dapat disusun ulang sehingga menghasilkan profil cache yang berbeda-beda setiap urutannya.

```
temp0 = sbox0[ address ] = hit   ⇒   temp2 = sbox2[ address
temp1 = sbox1[ address ] = hit   ⇒   temp3 = sbox3[ address
temp2 = sbox2[ address ] = miss  ⇒   temp0 = sbox0[ address
temp3 = sbox3[ address ] = miss  ⇒   temp1 = sbox1[ address
```

Akses-akses memori di atas dapat juga disusun ulang secara valid dengan profil cache yang tetap sama:

```
temp0 = sbox0[ address ] = hit   ⇒   temp1 = sbox1[ address
temp1 = sbox1[ address ] = hit   ⇒   temp0 = sbox0[ address
temp2 = sbox2[ address ] = miss  ⇒   temp3 = sbox3[ address
temp3 = sbox3[ address ] = miss  ⇒   temp2 = sbox2[ address
```

Walaupun begitu, penyerang tidak dapat meyakinkan dirinya bahwa pengurangan-pengurangan yang dilakukan dari informasi akses cache adalah valid karena penyerang sudah tidak dapat lagi untuk mencocokan akses pada profil dengan referensi pada source code. Terlebih, penambahan sebuah unit mutasi aliran instruksi dapat mengakibatkan akses memori salah yang akan membuat profil/keterangan mengenai akses ke cache yang ditangkap menjadi lebih kompleks.

Penyerang yang yakin dapat berpikir jika mereka dapat menghitung jumlah total dari cache hit dan cache miss pada putaran ke-0 dan pertama pada eksekusi algoritma DES. Lalu dengan hanya mengubah bit-bit input yang mempengaruhi satu S-box pada putaran pertama dan mengamati perubahan terhadap total jumlah cache hit dan cache miss yang sebelumnya pernah tercatat, penyerang masih dapat melihat terjadi cache hit atau cache miss. Misalkan penyerang dapat mencatat jumlah cache hit dan cache miss pada putaran pertama seperti sebagai berikut:

```
temp0 = sbox0[ address ] = hit
temp1 = sbox1[ address ] = hit
temp2 = sbox2[ address ] = miss
temp3 = sbox3[ address ] = miss
```

Lalu penyerang akan menelusuri bit-bit pada alamat masukan data yang diketahui dapat mengubah pengaksesan hanya terhadap satu S-box, dalam hal ini adalah S-box satu. Pada kenyataannya penyerang dapat benar pada sebuah nilai yang mengubah cache hit dari pengaksesan-pengkaksesan terhadap S-box ke cache miss pada putaran pertama. Walaupun akses-akses telah diurutkan kembali, penyerang masih memiliki pengetahuan bahwa sebelumnya telah terjadi dua kali cache hit dan pada hit selanjutnya, pengubahan yang penyerang lakukan hanya mempengaruhi akses-akses terhadap S-box satu. Hal ini dapat dituliskan seperti sebagai berikut;

```
temp2 = sbox2[ address ] = miss
temp3 = sbox3[ address ] = miss
temp0 = sbox0[ address ] = miss
temp1 = sbox1[ address ] = hit
```

Oleh karena itu, penyerang dapat melakukan klaim bahwa hasil akses-akses ke S-box satu akan menghasilkan cache hit karena terdapat hanya satu cache hit sebelum dan sesudah pengubahan alamat yang peyerang lakukan. Deduksi yang berlawanan bernilai true jika penyerang menemukan satu hit lagi daripada menemukan satu hit setelah melakukan operasi penghitungan. Tetapi pada kenyataannya, serangan seperti ini tidak dapat bekerja karena efek avalancehe pada DES, karena memungkinkan untuk mencari cukup bit-bit pada alamat yang akan melakukan siklus sehingga kita dapat meyakinkan diri untuk dapat mengubah pola pengaksesan dan membuat perbedaan.

Jika skema pertahanan seperti ini diimplelemntasikan sebagai perpanjangan dari processor non-deterministik dimana penjadwalan memungkinkan derajat tinggi yang memadai untuk perpindahan instruksi-instruksi. Hala ini dapat mengurangi determinisme keterangan/profil cache yang diobservasi secara signifikan dan membuat cache based attack menjadi lebih sulit untuk dilakukan. Tetapi mekanisme pertahanan seperti ini menyediakan perlindungan probabilistik yang dibatasi oleh kemungkinan pengurutan kembali memori pada algoritma. Jika algoritma kriptografi yang sedang diserang memiliki sedikit sekali akses terhadap memori yang dapat diurut ulang dan diakibatkan karena keterkaitan-keterkaitan

instruksi yang ada misalnya, maka mekanisme pertahanan ini secara keseluruhan akan menjadi tidak efektif dalam memeberikan perlindungan terhadap cache based attack.

3.4 Penempatan cache non-deterministik

Penggunaan pemetaan cache yang telah diacak telah dimaksudkan untuk sebagai metode menghindari konflik cache yang mungkin disebabkan oleh, misalnya pola pengaksesan maju. Skema pertahanan seperti ini dapat bekerja dengan memanfaatkan beberapa pembelokan pada bagaimana cache line ditentukan dari alamat, sehingga alamat yang sama dapat dipetakan ke cache line yang berbeda-beda. Teknik seperti ini menambahkan non-determinisme pada sistem kriptografi dengan mengubah fungsi pemetaan di antara run-run pada algoritma dan masih dapat memberikan kinerja yang baik.

Pada pandangan pertama, kita berpikir dapat menggunakan mekanisme pertahanan yang dijelaskan di atas untuk melindungi alat kriptografi dari serangan cache based attack dengan memanfaatkan beberapa derajat non-determinisme pada operasi cache. Tetapi dengan melihat permasalahan lebih dalam, ternyata sepertinya mekanisme pertahanan ini tidak mungkin untuk diimplementasikan. Jika kita memngimplementasikan kebijakan pemetaan yang diacak pada cache, kitadapat secara non-deterministik mengubah kinerja, dalam hal ini cache hit dan cache miss, di antara run-run pada algoritma. Walaupun hal ini mungkin terjadi, mekanisme pertahanan ini tidak terjamin karena karakteristik-karakteristik pemetaan yang dipilih secara keseluruhan akan menentukan kinerja dengan arsitektur yang tetap dan beban kerja yang dimiliki.

Pertama, permasalahan terjadi karena tidak seperti differntial power analysis, kita tidak tertarik pada perbedaan kinerja di antara run-run pada algoritma, tetapi kita lebih tertarik pada kemunculan deterministik cache hit pada akses S-box yang dimaksud pada putaran kedua eksekusi data encryption standard. Hal ini yang tidak dapat disembunyikan dengan mengubah pemetaan di antara isntruksi-instruksi.

Kedua, pengubahan pemetaan dengan cara yang lebih baik yaitu pada cache miss, akan menyebabkan seluruh data pada cache tidak dapat diambil setiap kali proses ini dilakukan. Hal ini disebabkan pemetaan-pemetaan yang dilakukan secara efekti menentukan pada cache mana elemen baru akan ditempatkan daripada menetukan pada cache mana data dapat ditemukan. Dengan mengubah pemetaan-pemetaan, semua yang telah dilakukan adalah untuk meyakinkan bahwa semua data tidak dapat ditemukan tetapi data baru yang memasuki cache akan ditempatkan pada lokasi yang berbeda. Efek netto menggambarkan permasalahan kinerja yang besar yang tidak menawarkan kinerja yang lebih baik daripada tidak memiliki cache sama sekali.

Oleh karena itu, pemanfaatan non-determinisme melalui pemetaan cache yang telah diacak tidak memberikan mekanismen pertahanan yang efektif karena mekanisme pertahanan ini tidak menjamin tingkat keefektivitasan dan tidak mampu untuk menyembunyikan informasi side channel yang diperlukan penyerang.

4. Penelitian lebih lanjut

Penelitian mengenai teknik yang memanfaatkan perilaku cache untuk tujuan kriptanalisi telah berhasil dilakukan. Contoh yang diberikan dimulai dengan dengan menganalisis aliran data yang melalui algoritma dengan tujuan membangun hubungan-hubungan di antara bit-bit dari informasi rahasia. Hubungan-hubungan ini ditemukan dengan kemampuan untuk menjelaskan perilaku cache saat algoritma kriptografi sedang dijalankan. Kita mampu mengendalikan masukan plainteks kepada algoritma kriptografi dengan tujuan membangun hubungan-hubungan dan memeriksa nilai yang memungkinkan dari kunci rahasia. Proses ini mempersempit nilai-nilai yang mungkin sehingga pencarian kunci rahasia akan menjadi lebih efektif.

Sebenarnya serangan cache based attack yang dilakukan seperti serangan side channel based

attack lainnya, berfokus pada algoritma kriptografi yang sedang diserang dan tidak dapat digeneralisasikan terhadap algoritma lain walaupun algoritma-algoritma tersebut memiliki struktur yang sama. Terdapat beberapa metode yang potensial efektif dalam menyerang algoritma-algoritma kriptografi secara luas dan mengidentifikasi beberapa permasalahan yang akan berpengaruh pada kemampuan serangan yang dilakukan pada praktik keseharian, yaitu:

- Asumsi tentang ukuran cache line akan mengubah bagaimana kita menyerang alat kriptografi sungguhan. Karena inti dari serangan yang dilakukan adalah kemampuan untuk menjelaskan arti dari kejadian cache it atau cache miss, fakta bahwa mengubah ukuran cache line akan mengubah profil cache perlu diperhitungkan pada serangan di dunia sebenarnya.
- Contoh algoritma yang diserang didasarkan pada struktur jaringan feistel. Akan sangat menarik untuk dilakukan penelitian mengenai algoritma lain seperti Advanced Encryption Standard (AES), yang didasari oleh desai jaringan permutasi/substitusi. Juga sangat menarik untuk melakukan penelitian mengenai algoritma Khufu yang membangkitkan S-box berdasarkan kunci untuk melihat apakah operasi penjadwalan rapuh untuk diserang.
- Status contoh serangan yang dilakukan adalah produk pertama dari riset, oleh karena itu masih sangat terbuka untuk penelitian lebih lanjut. Khususnya akan lebih menarik jika memperbaiki serangan terhadap DES sehingga menghasilkan hasil yang lebih bermanfaat dengan usaha komputasi yang lebih sedikit dari sisi penyerang. Hal yang penting untuk dilakukan juga adalah implementasi dan pengetesan teradap ide-ide penanganan pertahanan terhadap cache based attack yang telah diajukan.
- Contoh serangan yang dilakukan anya mempertimbangkan perilaku cache pada S-box berukuran besar, walaupun berpotensi lebih rumit untuk melakukan analisis, mungkin ada struktur data lain yang digunakan oleh sebuah algoritma yang bersedia untuk diperiksa atau dites dengan teknik serangan yang sejenis. Walaupun penelitian harus didasarkan per algoritma, hal ini tetap menawarkan area yang menarik untuk diteliti, mungkin penelitian difokuskan kepada penggunaan fungsi call stack.
- Berkaitan dengan permasalahan di atas, kita dapat menyelidiki implementasi kriptografi seperti DES, yang melakukan permutasi dengan menggunakan tabel lookup. Walaupun hal ini memakan biaya pada perangkat keras, tetapi dengan ukuran memori yang semakin besar, teknik ini dapat menjadi optimasi yang valid pada perangkat lunak yang dapat dieksploitasi oleh cache based attack.

5. Kesimpulan

Karena kemampuan S-box memanfaatkan kenon-linearan kepada cipher blok, S-box secara umum diperitungkan sebagai alat pertahanan vital teradap serangan kriptanalisis berbasis matematik. Tetapi dalam makalah ini ditunjukkan bahwa S-box memberikan ruang untuk side channel based attack. Side channel attack merupakan gangguan dari segi pembuat alat keamananan karena penyerang dapat melewati cara konvensional. Dengan menyerang langsung ke alat implementasi daripada ke spesifikasi algoritmanya, informasi rahasia bisa didapatkan dengan biaya yang efektif dan praktis. Penggunaan memori cache tanpa pertimbangan-pertimbangan dapat memungkinkan penyerang untuk medapatkan detil-detil tentang apa yang sedang berjalan pada processor dengan mengamati perilaku akses cache.

Pada makalah ini telah dijelaskan serangan pada DES yang telah melemahkan enkripsi setelah mendapatkan informasi mengenai perilaku cache. Hasilnya menunjukkan bahwa DES dengan 56 bit setara dengan 32 bit dalam masalah keamanan jika penyerang dapat mengumpulkan informasi mengenai cache hit dan cache miss yang terjadi selama eksekusi algoritma. Contoh yang digunakan telah

memberikan petunjuk tentang serangan dengan memanfaatkan perilaku cache pada sistem kriptografi lain dan juga sejumlah penanganan yang dapat membuat serangan lebih sulit untuk dilakukan.

Yang jelas, contoh teoritis dan penanganan yang dipertunjukkan pada makalah ini perlu diteliti lebih lanjut agar dapat mengasilkan serangan yang berguna dan dapat diimplementasikan pada alat kriptografi sungguhan. Hal yang perlu diperhatikan juga adalah cache based attack dapat memberikan informasi yang diperlukan oleh penyerang. Informasi ini akan berbahaya jika dengan informasi ini penyerang dapat mempersempit kandidat kunci yang akhirnya akan mengasilkan kunci rahasia. Pada akhirnya, pendesain sistem keamanan harus mempertimbangkan segala kebocoran informasi sebagai bahaya karena penyerang akan menggunakan apapun untuk kepentingan mereka

DAFTAR PUSTAKA

[1]http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html

[2]http://www.wikipedia.org/cryptanalysis.htm

[3]Theoritical use of cache memory as cryptanalitic side channel, Department of Computer Science, Univesity of Bristol