Penggunaan Timing Attack Sebagai Salah Satu Jenis Serangan pada Kriptografi

Widhi Ariandoko - 13508109
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
widhi_ariandoko@itb.ac.id

***Abstract*—Teknik kriptografi untuk menyembunyikan pesan telah berkembang selama berabad-abad, mulai dari kriptografi klasik yang paling sederhana seperti Caesar Cipher hingga kriptografi modern seperti Advanced Encryption Standard (AES) yang banyak digunakan hingga sekarang. Bersamaan dengan perkembangan teknik kriptografi tersebut, berkembanglah pula serangan-serangan untuk memecahkan cipher text dalam kriptografi. Salah satu dari serangan tersebut adalah Timing Attack. Timing Attack adalah salah satu jenis side channel attack yang menggunakan waktu yang dibutuhkan untuk mengeksekusi algoritma kriptografi sebagai alat untuk memecahkan kode kriptografi. Dalam makalah ini akan dibahas lebih lanjut studi smengenai Timing Attack sebagai salah satu jenis serangan pada kriptografi**

***Index Terms*—timing attack, serangan, kriptografi, kriptoanalis, side channel attack**

I. PENDAHULUAN

Pada bab ini akan dibahas mengenai latar belakang, rumusan masalah dan batasan masalah dalam pembuatan makalah ini.

A. Latar Belakang

Dengan berkembangnya teknologi digital, maka berkembang pula teknik kriptografi modern sebagai alat untuk menjamin keamanan penyampaian pesan rahasia tanpa diketahui oleh pihak ketiga. Namun demikian, dengan berkembangnya teknik kriptografi, berkembang pula teknik-teknik serangan untuk memecahkan, merusak atau memodifikasi cipher teks yang dihasilkan oleh teknik kriptografi untuk kepentingan pihak lain. Di samping untuk tujuan yang tidak baik, kadang serangan-serangan ini juga bertujuan positif jika digunakan oleh pihak yang berwenang untuk memecahkan cipher teks yang digunakan oleh penjahat dalam berkomunikasi. Dari sini berkembanglah teknik yang disebut kriptoanalisis.

Dalam perkembangannya, serangan pada kriptografi mulai menggunakan informasi yang bersifat fisik dari proses kriptografi seperti waktu proses dan kebutuhan energi dari sistem. Serangan seperti ini disebut side channel attack. Side channel attack yang menggunakan waktu proses sebagai alat untuk memecahkan kriptografi disebut Timing Attack. Timing Attack dalam prakteknya telah dapat memecahkan algoritma kriptografi yang populer digunakan saat ini yaitu Advanced Encryption Standard atau AES.

B. Rumusan Masalah

Makalah ini akan mengulas kembali sedikit mengenai side channel attack dan jenis-jenisnya sebelum mulai membahas mengenai timing attack sebagai salah satu jenis side channel attack.

Masalah pokok yang akan dibahas dalam makalah ini adalah pengertian dan cara kerja timing attack, beberapa implementasi timing attack dan kemungkinan pencegahan pada serangan timing attack.

C. Batasan Masalah

Pada makalah ini akan dibahas hal-hal mengenai teori timing attack secara umum dan konseptual. Adapun pengaplikasian dan contoh-contoh codenya hanya akan dibahas sekilas dan tidak mendalam. Pengulasan kembali algoritma kriptografi yang dapat diserang oleh timing attack akan dilakukan secara sepotong-sepotong sebatas potongan algoritma tersebut dapat digunakan untuk menjelaskan cara kerja timing attack.

Mengingat timing attack adalah salah satu jenis dari side channel attack, maka pengertian side channel attack secara umum juga akan secara dangkal dibahas dalam makalah ini. Jenis-jenis side channel attack selain timing attack hanya akan dijelaskan sekilas tanpa pembahasan lebih lanjut.

II. DASAR TEORI

Pada bab ini akan dibahas mengenai pengertian dan cara kerja timing attack dam penjelasan kembali beberapa algoritma kriptografi yang dapat diserang.

A. Timing Attack Sebagai Salah Satu Jenis Side Channel Attack

Dalam perkembangannya, serangan pada kriptografi tidak hanya menyerang kelemahan-kelemahan dari

algoritma kriptografi secara matematika namun juga menyerang kelemahan-kelemahan algoritma kriptografi dalam implementasi fisiknya untuk mendapatkan data-data seperti kunci rahasia atau plain teks. Serangan yang memanfaatkan kelemahan algoritma kriptografi dalam implementasi fisiknya ini disebut Side Channel Attack.

Side Channel Attack dapat dibagi menjadi beberapa jenis berdasarkan implementasi fisik yang diserang. Adapun pembagiannya adalah sebagai berikut:

1. Timing Attack
   Timing Attack menyerang kriptografi dengan menggunakan lama waktu eksekusi algoritma kriptografi yang diserang.
2. Power Monitoring Attack
   Power Monitoring Attack atau biasa juga disebut Power Attack menggunakan kebutuhan konsumsi energi dari sistem dalam mengeksekusi algoritma kriptografi yang akan diserang.
3. Elektromagnetik Attack
   Elektromagnetik Attack menggunakan radiasi elektromagnetik yang dihasilkan oleh proses eksekusi algoritma kriptografi yang diserang.
4. Acoustic Cryptanalys
   Acoustic Cryptanalys menggunakan suara yang dihasilkan dalam proses radiasi untuk mendapatkan data-data seperti kunci rahasia atau plain teks.
5. Cache Attacks
   Cache Attacks menyerang kriptografi dengan menggunakan data-data antar proses yang tersimpan pada cache memory.
6. Differential Fault Analysis
   Differential Fault Analysis adalah serangan yang menggunakan kesalahan-kesalahan implementasi fisik yang tak terduga dari suatu proses kriptografi.

Selanjutnya kita akan membahas lebih lanjut mengenai Timing Attack, salah satu jenis Side Channel Attack yang menggunakan waktu eksekusi untuk menyerang kriptografi. Dalam hal ini penggunaan algoritma yang waktu enkripsinya tergantung pada kunci atau plai teks akan memudahkan penggunaan timing attack untuk menyerang algoritma tersebut.

B. Tabel Look Up pada Kriptografi

Pada Kriptografi, tabel look up (*look up table*) sering kali digunakan untuk menambah efek *diffusion* dan *confusion* dari proses enkripsi. Namun jika kurang berhati-hati, penggunaan tabel ini akan memudahkan suatu algoritma untuk diserang dengan timing attack.

Tabel look up adalah struktur data yang biasanya berupa array atau matriks. Tabel ini digunakan untuk mencari nilai dari suatu indeks dalam tabel tersebut atau sebaliknya untuk mencari indeks dari suatu nilai dalam tabel tersebut. Pencarian bisa menggunakan metode *sequential search*, *binary search* atau metode yang lain. Namun pada makalah ini hanya akan dibahas mengenai penggunaan tabel look up dengan metode pencarian berurut (*sequential search*).

Ada dua jenis tabel look up yang sering digunakan dalam kriptografi yaitu P-Box yang merupakan singkatan dari Permutation Box dan S-box yang merupakan singkatan dari Substitution Box.

PBox atau Permutation Box biasanya digunakan untuk operasi permutasi atau transposisi baik pada plain teks maupun pada kunci untuk menambah efek *diffusion*.

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

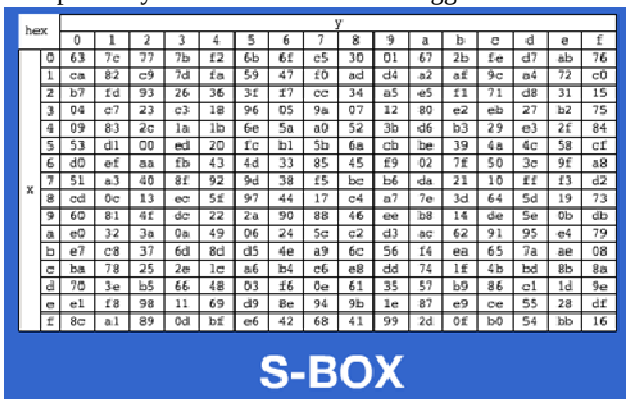
Gambar 1: Contoh P-Box pada algoritma DES

Pada contoh P-Box yang digunakan sebagai matriks permutasi awal (IP) pada algoritma DES di atas misalnya, bit pertama dari plain teks akan dipindah menjadi bit ke-58 pada hasil permutasi awal, bit ke-2 akan dipindah ke bit-50 dan seterusnya.

Berbeda dengan P-Box yang digunakan untuk permutasi dan transposisi, S-Box digunakan untuk operasi substitusi dan subbytes. S-Box biasanya berupa matriks atau array 2 dimensi.

Pada operasi substitusi, bit-bit pada teks biasanya dibagi menjadi dua bagian, satu bagian dipakai sebagai indeks kolom pada S-Box dan bagian yang lainnya dipakai sebagai indeks baris. Operasi substitusi ini tidak selalu menghasilkan jumlah bit yang sama dengan masukannya. Pada algoritma DES misalnya terdapat S-Box yang memetakan 6 bit masukan menjadi 4 bit keluaran yang disebut S-Box 6 x 4

Berbeda pada operasi susbstitusi, operasi subbytes tidak bekerja dalam satuan bit melainkan dalam satuan byte. Pada S-Box yang digunakan pada operasi subbytes algoritma Rijndael misalnya, byte-byte dalam teks yang akan dioperasikan direpresentasikan dalam bentuk heksadesimal dari 0 hingga FF. Byte ini kemudian dibagi menjadi dua yaitu digit pertama dan digit kedua (jika byte hanya terdiri dari satu digit maka digit pertama adalah o). Kemudian digit pertama dijadikan sebagai indeks baris, digit kedua dijadikan indeks kolom dan hasilnya adalah nilai dari elemen S-Box dengan indeks baris dan indeks kolom tersebut. Hasil dari S-Box Rijndael ini juga merupakan byte heksadesimal dari 0 hingga FF.



Gambar 2: Rijndael S-Box

II ANALISA

Dari dasar teori di atas, kita dapat melihat bahwa timing attack akan dengan mudah menyerang algoritma kriptografi yang waktu enkripsinya tergantung pada kunci atau plain teks.

Algoritma kriptografi pada umumnya telah dipublikasikan secara umum, sementara yang dirahasiakan hanyalah kunci dan plain teksnya. Dengan mengetahui algoritma kriptografi yang akan diserang serta waktu yang dibutuhkan untuk menjalankan satu operasi (operasi penjumlahan, atau perkalian misalnya) maka akan mempermudah proses timing attack.

Berikut adalah contoh-contoh aplikasi penggunaan timing attack.

A. Timing Attack untuk Mencari Panjang Kunci pada Algoritma Kriptografi Sederhana

Pada kriptografi sederhana, seringkali didapatkan operasi penjumlahan seluruh atau sebagian byte-byte kunci untuk mendapatkan suatu nilai yang akan dipergunakan untuk keperluan enkripsi selanjutnya. Jika kita mengetahui berapa waktu yang digunakan untuk melakukan satu kali operasi penjumlahan, maka dengan mudah kita bisa mendapatkan panjang dari kunci tersebut.

Penggunaan iterasi yang berulang sebanyak panjang dari kunci juga akan memudahkan penggunaan timing attack. Sebagai contoh, potongan algoritma dalam bahasa C# berikut akan menjumlahkan byte-byte dari kunci yang berindeks ganjil lalu hasilnya akan digunakan untuk menggeser plaintext ke kanan sebesar hasil tersebut.

```
private static byte[] shift(byte[] key, byte[] plaintext)
{
  byte[] result = new byte[plaintext.Length];
  int initnumber = 0;

  for (int i = 0; i < key.Length; i++)
  {
      if (i % 2 == 1)
      {
       initnumber += key[i];
    }
  }

  for (int i = 0; i < plaintext.Length; i++)
  {
      result[i] = plaintext[(i + initnumber) %
plaintext.Length];
   }

   return result;
}
```

Asumsikan total waktu yang dibutuhkan untuk menjalankan iterasi yang pertama (baris ke-6) adalah X. Jika waktu yang dibutuhkan satu putaran iterasi yang pertama jika kondisi dalam if terpenuhi adalah A, dan waktu yang dibutuhkan untuk satu iterasi jika kondisi dalam if tidak terpenuhi adalah B maka kita akan mengetahui panjang kunci adalah genap jika X habis dibagi A + B dan ganjil jika tidak habis dibagi.

Selanjutnya panjang kunci dapat diperoleh dengan rumus:

- Jika genap: panjang kunci = X / (A + B)
- Jika ganjil: panjang kunci = (X – B) / (A + B

B. Penggunaan Timing Attack pada P-Box

P-Box sering digunakan untuk mengacak posisi byte atau bit plain teks maupun kunci, namun jika kita menggunakan P-Box dengan kurang hati-hati maka algoritma kita akan lebih mudah diserang dengan timing attack.

Sebagai contoh, pada potongan algoritma yang ditulis dalam bahasa C# di bawah ini, terdapat sebuah P-Box yang digunakan untuk mengacak posisi bit-bit plain teks sebelum masuk ke proses selanjutnya. Fungsi di bawah ini menerima 16 bit plainteks dan mengembalikan nilai berupa 16 bit hasil pengacakan plainteks. Untuk setiap bit ke-i pada hasil pengacakan akan berisi bit ke PBox[i] pada plain teks.

```
private static BitArray
Permutation(BitArray plaintext)
//fungsi menerima 16 bit plainteks
{
   int[] PBox = {10, 6, 14, 1,
                 0, 12, 13, 8,
                 2, 7, 11, 4,
                 3, 5, 15, 9};

   BitArray result = new BitArray(16);
   for (int i = 0; i < 16; i++)
   {
       int pos = 0;
       while (PBox[pos] != i)
       {
           pos++;
        }
        result[i] = plaintext[pos];
   }

   return result;
   //fungsi mengembalikan hasil pengacakan
plain teks
}
```

Selain tidak efektif, fungsi di atas juga memudahkan penggunaan timimg attack. Asumsikan kita tidak mengetahui isi dari P-Box (pada kasus lain dimungkinkan P-Box dibangkitkan dari kunci, jadi untuk kunci yang berbeda P-Box yang digunakan juga berbeda). Namun jika kita mengetahui waktu yang dibutuhkan untuk mendapatkan result[i] untuk setiap nilai i dan kita mengetahui waktu yang dibutuhkan untuk satu kali proses penjumlahan dan proses perbandingan, maka kita akan

bisa menyusun isi dari P-Box.

Misalkan waktu yang dibutuhkan untuk mendapatkan result[i] adalah X, waktu yang digunakan untuk satu kali proses perbandingan adalah A, waktu yang dibutuhkan untuk satu kali proses penjumlahan adalah B, dan waktu yang dibutuhkan untuk menginisialisasi nilai pos adalah C, maka akan ada suatu nilai n di mana

$$X = (n+1)\,A + nB + C$$
$$\Leftrightarrow X = nA + A + nB + C$$
$$\Leftrightarrow X - (A + C) = n\,(A + B)$$
$$\Leftrightarrow n = (X - A - C) / (A + B)$$

dengan n adalah nilai di mana PBox[n] = i. Jika kita bisa mengetahui besar X untuk semua nilai i, kita akan bisa memetakan semua nilai i ke dalam P-Box dan P-Box akan dapat tersusun secara lengkap.

Selain menggunakan satu for dan satu while, fungsi ersebut juga dapat dituliskan dengan menggunakan dua for dan satu if, sehingga penulisannya akan menjadi seperti berikut

```
private static BitArray
Permutation(BitArray plaintext)
//fungsi menerima 16 bit plainteks
{
   int[] PBox = {10, 6, 14, 1,
                 0, 12, 13, 8,
                 2, 7, 11, 4,
                 3, 5, 15, 9};

   BitArray result = new BitArray(16);
   for (int i = 0; i < 16; i++)
   {
       for (int j = 0; j < 16; j++)
       {
           if (PBox[j] == i)
           {
               result[i] = plaintext[j];
           }
       }
   }

   return result;
   //fungsi mengembalikan nilai hasil
pengacakan plain teks
}
```

Fungsi di atas selain semakin tidak efektif juga masih rentan terhadap serangan timing attack. Jika waktu yang dibutuhkan untuk menjalankan satu putaran iterasi for kedua (baris 13) untuk setiap nilai j dalah T(j), maka akan ada saat di mana j = x dengan T(x) lebih besar daripada nilai T(j) untuk seluruh $j \neq x$. Dengan mengetahui data tersebut maka kemungkinannya adalah PBox[x] = i. Dengan menggunakan cara tersebut untuk seluruh nilai i, kita akan bisa memetakan semua nilai i ke dalam P-Box dan P-Box akan dapat tersusun secara lengkap.

Selain dengan dua cara sebelumnya, fungsi untuk mengacak plain teks di atas dapat juga dituliskan lebih singkat dengan cara sebagai berikut.

```
private static BitArray
Permutation(BitArray plaintext)
//fungsi menerima 16 bit plainteks
{
   int[] PBox = {10, 6, 14, 1,
                 0, 12, 13, 8,
                 2, 7, 11, 4,
                 3, 5, 15, 9};

   BitArray result = new BitArray(16);
   for (int i = 0; i < 16; i++)
   {
       result[PBox[i]] = plaintext[i];
   }

   return result;
   //fungsi mengembalikan nilai hasil
pengacakan plain teks
}
```

B. Penggunaan Timing Attack pada S-Box untuk Operasi Subbytes

Seperti halnya pada P-Box, S-box yang digunakan untuk operasi Subbytes juga mempermudah penggunaan timing attack jika digunakan dengan kurang berhati-hati.

Sebagai contoh mari kita perhatikan fungsi yang dituliskan dalam Bahasa C# di bawah ini. Kedua fungsi di bawah ini meminta masukan plain teks yang berupa array of String yang tiap elemennya berisi byte dari plain teks yang dituliskan dalam bentuk heksadesimal (diasumsikan masukan selalu terdiri dari dua digit heksadesimal) menggunakan tipe String. Nilai heksadesimal dalam tiap elemennya bernilai antara 0 sampai FF (255 dalam desimal). Hasil keluarannya juga merupakan byte dalam String heksadesimal.

```
private static String[,] SBox =
{{"59", "68", "ff", "a6", "f5", "88", "6",
"1d", "8d", "eb", "46", "e5", "d6", "b2",
"2e", "d9"},

{"d7", "12", "f9", "5a", "41", "78", "bd",
"3c", "ed", "fe", "71", "ab", "6b", "14",
"52", "7a"},

{"9f", "1b", "5f", "da", "c8", "91", "c3",
"9e", "f4", "3a", "3", "38", "be", "13",
"e0", "96"},

{"84", "af", "90", "e1", "c2", "4b", "7",
"8a", "95", "72", "0", "7f", "31", "92",
"5c", "79"},

{"fc", "4a", "67", "97", "a3", "e6", "2",
"57", "b5", "3f", "5e", "ca", "f7", "e7",
"64", "e9"},
```

```
{"70", "9c", "8f", "c0", "2b", "f3", "8",
"db", "ac", "66", "99", "39", "48", "25",
"c4", "bf"},

{"51", "36", "a7", "9", "cd", "fd", "54",
"a1", "dd", "6d", "69", "77", "fa", "93",
"5d", "ae"},

{"1c", "6f", "61", "dc", "9a", "6e", "bc",
"ea", "3b", "73", "58", "89", "2c", "e8",
"76", "cb"},

{"35", "c5", "e", "b7", "43", "b1", "f2",
"d2", "a2", "d0", "30", "33", "b3", "86",
"17", "8c"},

{"f6", "5", "3e", "65", "45", "83", "9b",
"7c", "7b", "7d", "f0", "9d", "7e", "d5",
"94", "6c"},

{"11", "10", "ee", "d8", "ec", "c", "b",
"1", "a8", "d1", "6a", "f", "47", "16",
"55", "a5"},

{"d", "28", "80", "aa", "c1", "c6", "37",
"e4", "f8", "18", "8e", "62", "98", "74",
"2d", "53"},

{"32", "e2", "b6", "60", "cc", "b9", "21",
"50", "b0", "4e", "ef", "d4", "40", "29",
"2a", "49"},

{"c9", "fb", "4f", "26", "df", "44", "bb",
"24", "4", "ce", "1f", "a9", "cf", "e3",
"4d", "3d"},

{"85", "75", "23", "56", "2f", "a4", "ad",
"b4", "22", "8b", "27", "c7", "63", "42",
"19", "1e"},

{"de", "15", "1a", "34", "a0", "5b", "b8",
"81", "d3", "20", "4c", "ba", "82", "a",
"87", "f1"}};

private static String[] Subbytes(String[]
text)
{
    int[] baris = new int[text.Length];
    int[] kolom = new int[text.Length];

    String[] result = new
String[text.Length];
    for (int i = 0; i < text.Length; i++)
    {
        baris[i] =
Convert.ToInt32(text[i].Substring(0, 1),
16);
        kolom[i] =
Convert.ToInt32(text[i].Substring(1, 1),
16);

        result[i] = SBox[baris[i],
kolom[i]];
    }

    return result;
}

private static String[]
InverseSubbytes(String[] text)
{
    String[] result = new
String[text.Length];

    for (int i = 0; i < text.Length; i++)
    {
       for (int j = 0; j < 16; j++)
       {
          for (int k = 0; k < 16; k++)
          {
             if (text[i].Equals(SBox[j,
k]))
                {
                result[i] =
j.ToString("X") + k.ToString("X");
                }
          }
       }
    }

return result;
}
```

Pada fungsi InverseSubbytes di atas, timing attack dapat dilakukan dengan cara yang hampir sama dengan contoh kedua timing attack pada P-Box di sub bab sebelumnya.

Berhubung fungsi tersebut adalah fungsi untuk membaca Sbox secara terbalik maka dapat diperkirakan bahwa fungsi tersebut dipakai untuk proses dekripsi. Jadi timing attack di sini bertujuan untuk mencari keluaran fungsi tersebut (variabel result)

Pada fungsi tersebut terdapat 3 buah iterasi for, Jika kita ingin mencari result[i], maka kita harus mengetahui waktu yang dibutuhkan untuk satu kali putaran iterasi yang paling dalam (iterasi dengan variabel k). Untuk setiap i, iterasi k akan dilakukan sebanyak (j x k) = 256 kali. Dari ke 256 putaran tersebut kita cari satu putaran dengan waktu eksekusi yang paling lama dan kita beri nama putaran ke-n. Selanjutnya, nilai j dan k dapat kita cari dengan menggunakan persamaan

$$k = n \bmod 16$$

$$j = (n - k) / 16$$

selanjutnya hasil inverse SBox (dalam desimal byte) dapat dicari dengan persamaan

$$hasil = (j \times 16) + i$$

Untuk menghindari penggunaan timing attack pada S-Box, sebaiknya pada proses dekripsi dibuat sebuah tabel Inverse S-Box. Tabel Inverse S-Box dan fungsi InverseSubbytes untuk fungsi di atas adalah sebagai berikut:

```
private static String[,] SBox = {{"59", "68",
"ff", "a6", "f5", "88", "6", "1d", "8d",
"eb", "46", "e5", "d6", "b2", "2e", "d9"},

{"d7", "12", "f9", "5a", "41", "78", "bd",
"3c", "ed", "fe", "71", "ab", "6b", "14",
"52", "7a"},

{"9f", "1b", "5f", "da", "c8", "91", "c3",
"9e", "f4", "3a", "3", "38", "be", "13",
"e0", "96"},

{"84", "af", "90", "e1", "c2", "4b", "7",
"8a", "95", "72", "0", "7f", "31", "92",
"5c", "79"},

{"fc", "4a", "67", "97", "a3", "e6", "2",
"57", "b5", "3f", "5e", "ca", "f7", "e7",
"64", "e9"},

{"70", "9c", "8f", "c0", "2b", "f3", "8",
"db", "ac", "66", "99", "39", "48", "25",
"c4", "bf"},

{"51", "36", "a7", "9", "cd", "fd", "54",
"a1", "dd", "6d", "69", "77", "fa", "93",
"5d", "ae"},

{"1c", "6f", "61", "dc", "9a", "6e", "bc",
"ea", "3b", "73", "58", "89", "2c", "e8",
"76", "cb"},

{"35", "c5", "e", "b7", "43", "b1", "f2",
"d2", "a2", "d0", "30", "33", "b3", "86",
"17", "8c"},

{"f6", "5", "3e", "65", "45", "83", "9b",
"7c", "7b", "7d", "f0", "9d", "7e", "d5",
"94", "6c"},

{"11", "10", "ee", "d8", "ec", "c", "b", "1",
"a8", "d1", "6a", "f", "47", "16", "55",
"a5"},

{"d", "28", "80", "aa", "c1", "c6", "37",
"e4", "f8", "18", "8e", "62", "98", "74",
"2d", "53"},

{"32", "e2", "b6", "60", "cc", "b9", "21",
"50", "b0", "4e", "ef", "d4", "40", "29",
"2a", "49"},

{"c9", "fb", "4f", "26", "df", "44", "bb",
"24", "4", "ce", "1f", "a9", "cf", "e3",
"4d", "3d"},

{"85", "75", "23", "56", "2f", "a4", "ad",
"b4", "22", "8b", "27", "c7", "63", "42",
"19", "1e"},

{"de", "15", "1a", "34", "a0", "5b", "b8",
"81", "d3", "20", "4c", "ba", "82", "a",
"87", "f1"}};

private static String[]
InverseSubbytes(String[] text)
{
    int[] baris = new int[text.Length];
    int[] kolom = new int[text.Length];

    String[] result = new
String[text.Length];
    for (int i = 0; i < text.Length; i++)
    {
        baris[i] =
Convert.ToInt32(text[i].Substring(0, 1),
16);

        kolom[i] =
Convert.ToInt32(text[i].Substring(1, 1),
16);

        result[i] = SBox[baris[i],
kolom[i]];
    }
```

D. Kelebihan dan Kelemahan Timing Attack

Kelebihan dari timing attack adalah bahwa penggunanya tidak perlu mencari kelemahan-kelemahan pada algoritma kriptografi secara matematika melainkan harus mencari kelemahan algoritma dalam implementasi nyatanya yang berhubungan dengan waktu. Mencari kelemahan suatu program secara matematika membutuhkan pengetahuan yang mendalam mengenai matematika.

Kelemahan dari timing attack adalah bahwa timing attack membutuhkan perhitungan secara akurat terhadap waktu yang dibutuhkan program untuk mengeksekusi algoritma kriptografi. Timing attack akan lebih efektif jika digunakan pada piranti-piranti yang berkecapatan rendah, namun untuk piranti yang berkecapatan tinggi, kesulitannya akan semakin bertambah.

III SIMPULAN

Dari analisa di atas dapat disimpulkan bahwa algoritma yang mudah diserang dengan timing attack adalah algoritma yang tergantung pada kunci atau plain teks. Operasi yang dilakukan secara berulang sebanyak panjang kunci sebaiknya dihindari jika panjang kunci dirahasiakan.

Penggunaan tabel look up sebaiknya tidak menggunakan sequential search untuk menemukan elemen yang sesuai. Pada pembacaan S-Box secara terbalik sebaiknya dibuat tabel Inverse S-Box untuk menghindari

pencarian secara berurut.

REFERENCES

[1] *Timing Attack*, http://en.wikipedia.org/wiki/Timing_attack, diakses pada 5 Maret 2011

[2] Evars Petterson, "*Timing Attack Beats Cryptographic Keys – Paul Kocher's Research Indicates that Computer Security Based on Cryptosystems may be more Vulnurable than Previously Thought – Brief Article"*, Science Service, Inc, Dec 1995

[3] *Side Channel Attack*, http://en.wikipedia.org/wiki/Side_channel_attack diakses pada 20 Maret 2011

[4] *Lookup table*, http://en.wikipedia.org/wiki/Lookup_table diakses pada 20 Maret 2011

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Maret 2011

Widhi Ariandoko
NIM 13508109