Analisis dan Implementasi Algoritma PMAC pada Aplikasi Pidgin

Herianto
Jurusan Teknik Informatika ITB, Bandung 40132, email: if14077@students.if.itb.ac.id

Abstract *– Makalah ini mendefinisikan dan menagalisis bagaimana sebuah algoritma untuk melalukan otentikasi sebuah message dengan mode block cipher yang parallelizable lalu menerapkannya dalam sebuah aplikasi messaging. Algoritma ini memiliki kemampuan paralelisme yang tidak ditujukan untuk mengganti efisiensi serial yang pada ketentuannya, kecepatan algoritmanya sampai beberapa persen (diturunkan sequential) dari CBC MAC. PMAC yang ditawarkan bersifat deterministic, menyerupai operasi hash standar (selain Carter-Wegman MAC), bekerja pada string dengan panjang bit berapapun, memakai sebuah key block cipher tunggal dan menggunakan maksimal {1, [|M| / n]} block cipher calls ke sebuah string M € {0,1} dengan sebuah n-bit block cipher.*

Melalui plugin pada aplikasi messaging ini terlihat PMAC aman, mengukur probabilitas pemalsuan dari pengirim message dalam hal kualitas block cipher sebagai pseudorandom permutation. Aplikasi messaging yang dipilih adalah Pidgin dengan engine version 2.2.0 sehingga pengembangan yang dilakukan menggunakan API untuk versi 2.2.0 juga. Untuk mengimplementasikan plugin ini penulis menggunakan libpurple, sebuah library yang berkorelasi dengan core Pidgin yang kemudian dimafaatkan untuk mengakses sebuah koneksi jaringan, account list, conversation dan lain-lain.

Kata Kunci *: mode block-cipher, message authentication code, mode operasi, libpurple*

1. PENDAHULUAN

Ada begitu banyak message authentication code (MAC) seperti CBC MAC dan HMAC yang pada umumnya bersifat sequential, maksudnya tidak mampu memproses block message yang ke-i sampai semua block message sebelumnya telah diproses. Kelemahan ini merupakan isu yang telah lama berkembang dan setelah bertahun tahun menjadi bahan utama komoditas dalam menawarkan paralelisme sebagai solusi yang lebih baik seiring meningkatnya kecepatan jaringan yang melebihi peningkatan kecepatan hardware kriptografi. Pada saat ini, kemampuan paralelisme yang dimiliki oleh PMAC menjadi daya tarik sendiri yang signifikan yang performansinya sangat baik pada hardware maupun software yang dibangun dari block cipher seperti AES (Advanced Encryption Standard).

Aplikasi messaging sebagai salah satu media untuk menyampaikan pesan merupakan komoditas tepat untuk mengimplementasikan algoritma yang lebih mangkus dalam meningkatkan keamanan pengiriman pesan. Walaupun awalnya aplikasi ini hanya digunakan sebatas chatting, namun sekarang ini tidak jarang pula orang memanfaatkannya untuk percakapan yang membutuhkan kerahasiaan. Oleh karena itu dibutuhkan suatu sarana untuk menjamin keamanan dan kerahasiaan suatu percakapan dalam aplikasi ini. Solusinya tentu saja dengan memberikan jaminan integritas data dan layanan otentikasi.

Ada beberapa pendekatan untuk mendesain MAC tersebut. Salah satunya dengan mengkonstruksi MAC lebih paralel dari yang konstruksi message lama. Contohnya adalah memecah message M[1] … M[2m] ke dalam M’ = M[1]M[3] … M[2m-1] dan M’’ = M[2]M[4] … M[2m] beserta proses MAC untuk setiap bagian. Namun pendekatan ini membutuhkan sesuatu untuk mengantisipasi jumlah masksimal paralelisme yang nantinya akan dipecah. Pada pekerjaan saat ini fully parallelizable MAC jauh lebih dibutuhkan dimana jumlah paralelismenya bisa diekstrak tanpa terbatas.

Ada beberapa ide untuk mewujudkan fully paralalelizable MAC yang salah satunya adalah PMAC seperti yang dibahas dalam makalah ini. Penggunaan paradigm Carter-Wegman adalah salah satu ide pendahulunya. Metode ini melakukan proses tertentu untuk meyakinkan pemakaian salah satu universal hash-function yang fully parallelizable. Namun seperti kita ketahui, konstruksi fast universal hash-function sangatlah rumit dan kompleks untuk dispesifikasikan maupun diimplementasikan. Ide berikutnya adalah dengan XOR MAC. Ini adalah sebuah parallelizable MAC dengan paradigm XOR. Prosesnya adalah dengan membagi sebuah message M ke pecahan-pecahan yang lebih kecil yaitu M[1] … M[h] dimana h adalah panjang yang kurang dari ukuran block, yang pada kondisi sebenarnya seperti contoh berikut. Sebuah message M[i] dengan ukuran 64 bit dimana ukuran block adalah n = 128 bits. Setiap pecahan M[i] ditandai dengan i yang terdahulu yang nomor i ini merupakan encode dalam sebuah angka 64 bit serta untuk setiap [i]||M[i] diaplikasikan pada sebuah block cipher E dengan kunci MAC yaitu kunci K.

Ketika satu block lagi di-enchipher, block ini mempunyai sebuat bit pertama bernilai 1 dan kemudian sebuah counter atau nilai random pada n-1 bits sisanya. MAC adalah counter atau nilai random tersebut dengan XOR dari semua h+1 block cipherteks. Dengan demikian MAC menggunakan h+1 ≈2m+1 block cipher yang diinvokasi untuk

mengotentikasi sebuah message yang terdiri dari m block dan n bits. Namun paradigma ini memiliki kelemahan yaitu cost yang dibutuhkan dalam memparalelkan message hampir dua kali lipat factor yang mempengaruhi serial speed. Kelemahan lainnya adalah kebutuhan akan nilai random atau state dan peningkatan length yang lebih besar.

2. PEMBAHASAN

2.1. MAC dan PMAC

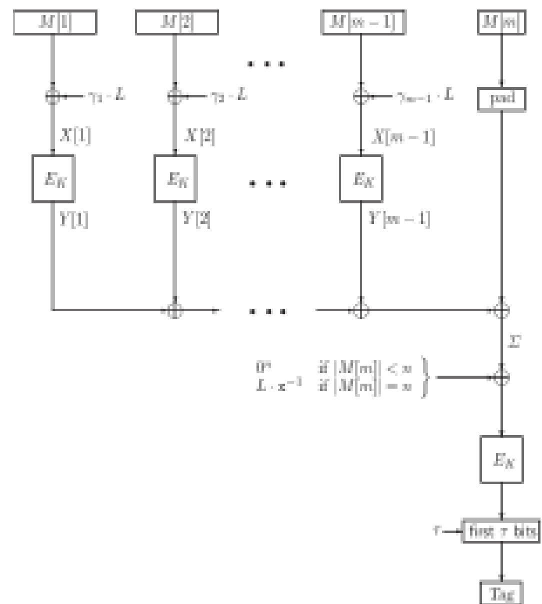
MAC adalah fungsi hash satu arah yang menggunakan kunci rahasia (secret key) dalam pembangkitan nilai hash. MAC disebut juga keyed hash function atau key-dependent one-way hash function. MAC memiliki sifat dan property yang sama seperti fungsi hash satu arah ditambah komponen kunci yang digunakan oleh penerima pesan untuk memverifikasi nilai hash. Secara umum MAC digunakan untuk otentikasi message tanpa perlu mengenkripsi pesan. Mula-mula pengirim pesan menghitung MAC dari pesan yang hendak dia kirim dengan menggunakan kunci rahasia K dengan asumsi kedua pihak sudah saling berbagi kunci rahasia. Penerima kemudian menggunakan kunci yang sama untuk menghitung MAC pesan dan membandungkannya dengan MAC yang diterimanya. Jika kedua MAC ini sama, maka dapa disimpulkan bawa pesan dikirim oleh orang yang benar dan isi pesan tidak diubah oleh pihak ketiga selama transmisi.

Karena PMAC adalah sebuah MAC yang berbasis block cipher makan perbedaan yang paling bisa lihat adalah pada MAC yang berbasis block cipher. Tidak seperti MAC dan XOR MAC, PMAC tidak menyia-nyiakan satupun invokasi block cipher karena tiap block memiliki index (tidak untuk counter atau nilai random). Hasil dari PMAC adalah membuat block cipher [|M| / n] ditujukan ke non-empty message M menggunakan sebuah n-bit block cipher. PMAC bersifat deterministic, membebaskan user dari kewajiban menyediakan sebuat counter atau random value serta mampu menciptakan MAC shorter. Pada PMAC digunakan key-setup yang sangat sederhana yaitu one block-cipher call. PMAC key merupakan single key yang mendasari block-cipher.

2.2. Algoritma PMAC

PMAC membutuhkan dua parameter yaitu sebuah block cipher dan sebuah tag length. Block cipher disini adalah sebuah fungsi E : $\mathcal{K} \times \{0,1\}^n$, dimana setiap $E(K,.) = E_K(.)$ adalah hasil permutasi pada $\{0,1\}^n$. $\mathcal{K}$adalah kumpulan kunci-kunci yan gmungkin sedangkan n adalah panjang block. Tag length itu sendiri adalah sebuah angka $_T$ yang menggambarkan panjang yang diambil dari [1..n]. $PMAC[E,_T]$ adalah sebuah fungsi yang mengambil sebuah key K dari $\mathcal{K}$ dan sebuah message M ϵ $\{0,1\}^*$ lalu mengembalikan sebuah string dalam $\{0,1\}^T$.

Jika kita asumsikan L adalah anggota dari kumpulan bits $\{0,1\}^n$, E adalah cipher block maka langkah pertama yang kita lakukan adalah meng-assign L dengan $E_K(0^n)$ dimana K adalah key. Jika block message |M| lebih besar daripada $n2^n$ maka fungsi selektornya harus mengembalikan 0^T dengan $_T$ adalah sebuah angka yang diambil dari [1 .. n]. Lalu block message M kita pecah menjadi M[1] sampai M[m]. Dari sini kita lakukan iterasi untuk melakukan proses XOR antara M[i] dengan hasil dot product γi dengan L itu sendiri. Hasil dari XOR ini dimasukkan ke block X[i]. Dalam iterasi ini kita juga melakukanproses meng-assign block Y[i] dengan $E_K(X[i])$. Tiap-tiap Y untuk i yang berurutan di XOR sampai Y[m-1] lalu di XOR lagi dengan block M[m] yang sudah dipadding. Hasilnya disimpan ke ∑. Jika block message |M[m]| = n, maka X[m] akan diproses dengan melakukan XOR kembali ∑ dengan hasil dot product L dan x^{-1}. Jika kondisi tidak terpenuhi maka ∑ akan dimasukkan ke X[m]. Langkah berikutnya adalah mengambil first $_T$ bits dari Y[m] yang sisimpan ke tag. Tag inilah yang nantinya akan dikomputasi oleh MAC. Secara sederhana jika Tag = Tag' maka message dinyatakan lulus otentikasi begitu juga sebaliknya.



Gambar 1 Ilustrasi PMAC

2.3 Contoh algoritma PMAC dalam bahasa JAVA

```
public final void pmac(byte[] data, int dataPos, int dataLen,
        byte[] tag, int tagPos)
        throws IllegalArgumentException
            {
    if (aes == null) {
        throw new RuntimeException("AES key not
initialized");
```

```
        }
        if (data == null || dataPos < 0 || dataLen < 0 || data.length -
dataPos < dataLen) {
            throw new IllegalArgumentException("Missing or
invalid data");
        }

        // Inisialisasi
        Arrays.fill(chksum, (byte)0);
        Arrays.fill(offset, (byte)0);

        // Memproses blocks 1 .. m-1.
        for (int i = 1; dataLen > BLOCK_SIZE; i++) {
            xorBlock(offset, offset, L[ntz(i)]);        // Update the
offset (Z[i] from Z[i-1])
            xorBlock(tmp, offset, data, dataPos);        // xor input
block with Z[i]
            aes.encrypt(tmp, tmp);
            // Update checksum and the loop variables
            xorBlock(chksum, chksum, tmp);
            dataPos += BLOCK_SIZE;
            dataLen -= BLOCK_SIZE;
        }

        // Memproses block m
        if (dataLen == BLOCK_SIZE) {   // full final block
            xorBlock(chksum, chksum, data, dataPos);
            xorBlock(chksum, chksum, L_inv);
        } else {                    // short final block
            Arrays.fill(tmp, (byte)0);
            System.arraycopy(data, dataPos, tmp, 0, dataLen);
            tmp[dataLen] = (byte)0x80;
            xorBlock(chksum, chksum, tmp);
        }
        aes.encrypt(chksum, tmp);
        System.arraycopy(tmp, 0, tag, tagPos, TAG_LENGTH);
    }
```

2.4 Implementasi PMAC pada Pidgin

Untuk menginisalisai plugin yang akan dimasukkan ke dalam pidgin kita harus memasukan kode PURPLE_INIT_PLUGIN(pidgin_plugin, init_plugin, info) kedalam source PMAC. File ini lah yang akan dikompile untuk membentuk sebuah file.dll yang akan dimasukkan ke dalam directory plugin pada pidgin. Prosedur ini adalah salah satu library dari libpurple. Parameter yang dipakai memiliki penjelasan sebagai berikut :

- pidgin_plugin : nama plugin
- init_plugin : fungsi untuk inisialisasi
- info : strktur data yang dipakai untuk menyimpan data yang dibutuhkan saat mengaktifkan plugin atan menonaktifkannya.

Untuk megimplementasikan source code PMAC pada Pidgin, event ini harus dtangkap sebuah event handler. Inisialisasi event handler ini dapat dilakukan dengan cara berikut :

purple_signal_connect(conv_handle, "receiving-im-msg", h, PURPLE_CALLBACK (PE_got_msg_cb), NULL); purple_signal_connect(conv_handle, "sending-im-msg", h, PURPLE_CALLBACK(PE_send_msg_cb), NULL);

Argumen purple_callback adalah fungsi yang akan dipanggil ketika event tersebut terjadi. Struktur data message lah yang akan ditangkap oleh fungsi event handler yang nantinya digunakan untuk mengubah dan memodifikasi pesan.

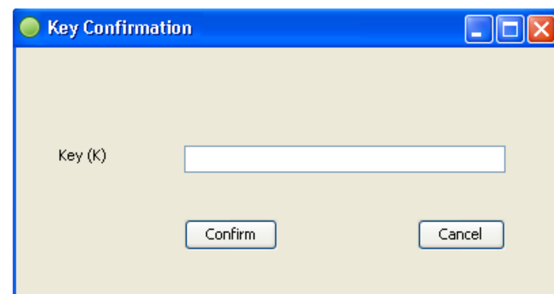
Untuk scenario teknis pada aplikasi, pihak satu (pengirim) harus menekan percakapan dengan tombol PMAC. Tombol inilah yang akan menginisialisasi implementasi PMAC dalam pidgin. Pihak satu dan pihak dua (penerima) harus memasukkan kunci yang sudah disepakati sebelumnya. Setiap pesan yang dikirim akan memiliki header dan nilai tag dari PMAC. Pihak dua harus mengecek otentikasi dengan proses PMAC juga sedangkan pihak satu akan selalu mengirimkan tag PMAC.

2.4 Eksperimen dan Pengujian Plugin

Sesuai langkah teknis berikut eksperimen yang dan pengujian yang dilakukan pada pidgin :

1. memilih tombol secure environment

2. memasukkan key



3. proses otentikasi

3. KESIMPULAN

PMAC jauh lebih bagus dibandingkan dengan metoede hashing maupun MAC lainnya. Metode ini sama bagusnya dengan fungsi pseudorandom (PRF) yang mempunyai variabel input length dan fixed output length. Selama cipher block dasar aman, hamper tidak mungkin untuk melakukan kecurangan atau pengambilan data secara pakas yang sudah dip roses dengan PMAC. Dengan plugin pada pidgin yang berhasil, kemungkinan besar hal ini akan menjadi komoditas baru dalam messaging security

DAFTAR REFERENSI

[1] http://pidgin.im
[2] J. Black, P.Rogaway, "A block Cipher Mode of Operation for Parallelizable Message Authentication ", *February 15. 2002*
[3] Munir, Rinaldi, Diktat Kuliah IF5054 Kriptografi, Institut Teknologi Bandung, 2007.