Studi Penggunaan *Rainbow Table* sebagai Metode Time-Memory Trade-Off untuk Kriptanalis pada MD5 dan NTLM

Ahmy Yulrizka [1)]

13504115

1) Program Studi Teknik Informatika Institut Teknologi Bandung, Bandung 13504115, email: if14115@students.if.itb.ac.id

Abstract **–** *Pada tahun 1980 Martin Hellman merumuskan suatu metode kriptanalis yang dimanakan time-memory trade-off yang mengurangi waktu yang dibutuhkan untuk melakukan kriptanalis dengan menggunakan pre-kalkulasi data yang disimpan dalam memori. Kemudian sebelum tahun 1982 Tehnik ini dikembangkan oleh Rivest sehingga mereduksi jumlah penelusuran memory secara signifikan pada saat melakukan kriptanalis. Philippe Oechslin mengemukakan metode mari yang dapat mempercepat lookup menjadi jauh lebih cepat. metode prekalkulasi ini tidak menggunakan teknik "distinguished points" sebagaimana digunakan Rivest. Tabel pre-kalkulasi ini dikenal dengan nama Rainbow Table[1]. Makalah ini akan mempelajari bagaimana penggunan Rainbow Table dalam melakukan kriptanalis terhadap message digest dari fungsi hash. Selain itu makalah ini juga akan membahas tentang bagaimana proses untuk menghasilkan Rainbow Table dan melakukan uji coba dan analisis pada kriptanalis NTLM hash dan MD5 dengan menggunakan program yang telah ada yang bernama RainbowCrack.*

Kata Kunci*: Rainbow Table, Time Memory Trade-off, kriptanalis, prekomputasi, RainbowCrack*

1. PENDAHULUAN

1.1. Pengertian dan Dasar Teori

Serangan Kriptanalis dengan menggunakan exhaustive search memerlukan proses komputasi yang besar. Apabila serangan yang sama dilakukan berkali – kali maka exhaustive search tersebut dapat dilakukan sebelumnya dan hasilnya disimpan pada memori. Metode tersebut dikenal dengan prekomputasi. Setelah prekomputasi tersebut telah dilakukan maka untuk melakukan kriptanalis menjadi jaub lebih cepat. Akan tetapi hal tersebut tidak praktis mengingat jumlah memori yang dibutuhkan sangat besar. Pada [2] Helman menawarkan metode *memory-trade* terhadap waktu.

Time-memory trade-off adalah suatu metode probabilistik. Kesuksesan dalam metode ini tidak dijamin. Kesuksesan tergantung pada waktu dan memori yang diberikan pada proses kriptanalis. Apabila bila diberikan plainteks P_0 dan C_0 adalah plainteks yang berkorespondensi dan N adalah semua kunci pada suatu sistem. metode ini mencoba menemukan sebuah kunci $k \in N$ yang digunakan untuk melakukan encipher plainteks menggunakan fungsi S. sehingga rumus menjadi

$$C_0 = S_k(P_0) \quad (1)$$

kita mencoba untuk menghasilkan semua cipherteks terlebih dahulu dengan cara melakukan *enciphering* suatu plainteks dengan semua kunci N. Cipherteks tersebut disusun membentuk suatu rantai (*chain*) dimana elemen yang disimpan adalah elemen pertama dan elemen terakhir. Operasi tersebut membentuk suatu *Trade-off* antara pengurangan memori dengan waktu yang digunakan untuk kriptanalis. Raintai tersebut dibuat dengan menggunakan **fungsi reduksi** yang berfungsi untuk menghasilkan sebuah kunci dari suatu cipherteks. Cipherteks tersebut lebih panjang dibanding kunci yang dihasilkan, oleh karena itu disebut reduksi. Dengan melakukan fungsi reduksi tersebut berkali-kali maka kita dapat membuat suatu rantai key dan cipherteks yang berubah-ubah.

$$k_i \xrightarrow{S_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} k_{i+1} \quad (2)$$

Suksesi dari $R(S_k(P_0))$ dapat ditulis menjadi f(k) yang melambangkan fungsi reduksi dan menghasilkan kunci baru dari kunci sebelumnya sehingga menciptakan suatu rantai kunci :

$$k_i \xrightarrow{f} k_{i+1} \xrightarrow{f} k_{i+2} \rightarrow \cdots \quad (3)$$

Rantai tersebut dibuat sebanyak *m* dengan panjang rantai *t* dan elemen pertama dan terakhir disimpan pada suatu tabel. Apabila terdapat Cipherteks C, kita dapat mencari apakah kunci yang digunakan untuk menghasilkan C terdapat diantara kunci yang digunakan untuk menghasilkan tabel tersebut. Untuk Melakukannya, kita membuat rantai baru dimulai dengan R(C) sepanjang *t*. Apabila memang C dihasilkan menggunakan kunci yang sama yang terdapat pada tabel tersebut, maka akhirnya kita akan menghasilkan kunci yang sama dengan kunci terakhir pada rantai di tabel tersebut. Dengan kunci terkahir tersebut kita dapat mencari kunci pertama pada tabel karena tabel itu berisi pasangan kunci awal dan kunci akhir yang dihasilkan dengan melakukan fungsi reduksi dengan jumlah *t.* Dengan menggunakan kunci awal ini, kita dapat menghasilkan rantai tersbut kembali. dan kunci yang muncul sebelum R(C) adalah kunci yang digunakan untuk menghasilkan cipherteks C dimaka kunci tersebut adalah kunci yang kita cari.

Tetapi terdapat suatu kemungkinan bahwa rantai yang dimulai oleh kunci yang berbeda bertubrukan (collide) dan akhirnya bergabung. Hal ini berarti bahwa terdapat kemungkinan bahwa rantai dengan kunci yang berbeda ditengah – tengah menajdi sama dan menghasilkan kunci akhir yang sama. Hal ini disebabkan oleh kenyataan bahwa fungsi reduksi R adalah fungsi reduksi absolut dari ruang cipherteks menjadi ruang kunci. Semakin besar suatu tabel maka semakin besar terjadi kemungkinan bahwa rantai yang baru akan berkoalisi dan bergabung menajdi rantai yang telah ada sebelumnya. Tiap penggabungan yang terjadi mengurangi jumlah *distinct keys* yang sebenarnya terlingkupi dalam suatu tabel.

Terdapat juga suatu kemungkinan bahwa ditemukan kunci akhir yang sama antara tabel dan ciphertkes tetapi tidak ditemukan kunci yang digunakan untuk melakukan *encipher*, hal ini dinamakan *false alarm*. Pada saat kita mencari suatu kunci pada tabel, menemukan suatu kunci akhir tidak memastikan bahwa kunci tersebut terdapat dalam tabel. Terdapat kemungkinan lain bahwa kunci tersebut adalah bagian dari rantai yang memiliki hasil akhir sama tetapi tidak terdapat dalam tabel. Pada kasus tersebut membuat rantai tersebut dari kunci awal yang telah disimpan dalam tabel tidak akan menghasilkan kunci yang *dicari. False alarm* juga terjadi pada saat kunci dalam suatu rantai merupakan bagian dari tabel yang bergabung dengan rantai lainnya pada tabel tersebut. Oleh karena itu beberapa kunci yang berbeda akan menghasilkan kunci akhir yang sama. Dalam kasus itu beberapa kunci awal harus *di generate* kembali sampai akhirnya menghasilkan kunci yang dipakai untuk menghasilkan cipherteks ditemukan.

1.2. **Batasan dan Parameter**

Pada metode time-memory trade-off terdapat tiga paramter yang dapat disesuaikan. batasan tersebut adalah : panjang rantai ***t***, banyaknya rantai tiap tabel ***m***, dan jumlah tabel yang diproduksi ***l***. Parameter ini dapat disesuaikan untuk memenuhi batas pada ukuran memori ***M***, waktu kriptakanis ***T***, dan kemungkinan kesuksesan (Success Rate) $P_{success}$. Batasan pada kemungkinan kesuksesan dapat dihitung dengan rumus :

$$P_{success} \geq 1 - \left(1 - \frac{1}{N}\sum_{i=1}^{m}\sum_{j=0}^{t-1}\left(1 - \frac{it}{N}\right)^{j+1}\right)^{\ell} \quad (4)$$

Sedangkan batasan pada memori *M* dihitung berdasakarkan jumlah rantai tiap tabel *m*, jumlah tabel *l*, dan ukuran memori m_0 yang digunakan untuk menyimpan kunci awal dan kunci akhir pada tabel. Batasan pada waktu *T* dihitung dari panjang rata – rata rantai *t*, jumlah tabel *l* dan rata – rata $1/t_0$ dimana plainteks dapat di *encipher.* Batasan ini merupakan kasus terburuk yang mungkin terjadi dimana kita harus mencari pada semua tabel. Tetapi pada perhitungan ini kasus *False Alarm* tidak dihitung.[1]

$$M = m \times \ell \times m_0 \quad (5)$$

$$T = t \times \ell \times t_0 \quad (6)$$

2. RAINBOW TABLE

Sampai saat ini kita telah membahas bagaimana cara kerja, kekuatan dan kelemahan dari metode time-memory trade-off selain RainbowTable. Batasan utama dari metoda yang original adalah ketika dua rantai saling bertubrukan (*collide*) pada tabel yang sama, rantai tersebut akan bergabung. Philippe Oechlin mengusulkan suatu tipe rantai baru yang apabila terjadi koalisi pada tabel yang sama, rantai-rantai tersebut tidak saling bergabung. Ia menamakan rantai tersebut *rainbow chains. [1]*

Rantai ini menggunakan fungsi reduksi yang bertahap untuk tiap titik dalalam suatu rantai. Fungsi tersebut berawal dari fungsi reduksi 1 sampai fungsi reduksi *t – 1*. Tetapi apabila terdapat dua rantai yang saling berkolisi, rantai tersebut hanya akan bergabung apabila koalisi muncul pada posisi yang sama di kedua rantai tersebut. Apabila kolisi tidak terjadi pada posisi yang sama, maka rantai tersebut akan dilanjutkan kembali dengan fungsi reduksi yang berbeda sehingga kedua rantai tersebut tidak akan bergabung. Untuk rantai dengan panjang *t*, jika kolisi terjadi, kemungkinan rantai tersebut bergabung hanya sebesar 1/*t*. Probabilitas kesuksesan pada satu tabel berukuran *m* x *t* adalah : [1]

$$P_{table} = 1 - \prod_{i=1}^{t}(1 - \frac{m_i}{N}) \quad (7)$$

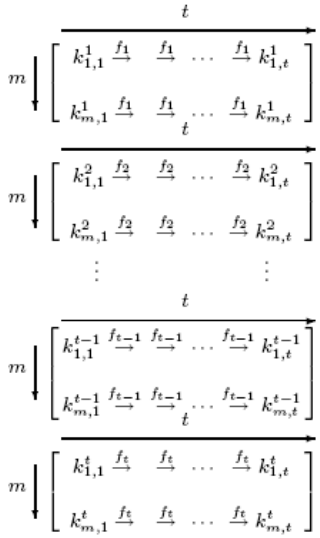
dimana

$$m_1 = m \quad \text{and} \quad m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right)$$

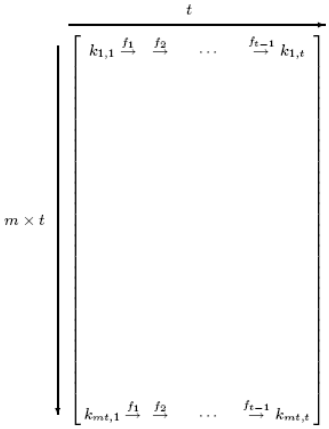
Probabilitas kesuksesan tabel klasik *t* dengan ukuran *m x t* hampir sama dengan sebuah rainbow table degan ukuran *mt x t.* Keduanya mencakupi kunci sebanyak mt^2 dengan fungsi reduksi sebanyak *t*. untuk tiap kolisi pada himpunan kunci *mt* (satu tabel klasik atau kolom dalam rainbow table) yang menghasilkan penggabungan, dimana kolisi dengan kunci sisanya tidak bergabung.

Keuntungan Rainbow Table adalah [1] :

- Jumlah pencarian pada tabel berkurang sebesar faktor *t* dibandingkan dengan metode original oleh Hellman
- Penggabungan akibat kolisi yang menghasilkan nilai akhiryang sama dapat dideteksi, sama seperti metode *distinguished points* oleh Rivest. Rainbow chains dapat digunakan untuk menghasilkan tabel yang terbebas dari penggabungan. Tetapi perhatikan bahwa tabel tersebut tidak *collision-free*
- Pada Rainbow chains tidak terdapat loop dimana setiap fungsi reduksi hanya muncul satu kali. Metode ini lebih baik dibandingkan dengan menditeksi dan menolak loop karena kita tidak menghasibkan waktu dan lingkupan dari rantai tersebut tidak berkurang.
- Rainbow chains memiliki panjang yang konstan dimana raintai yang menghasilkan *distinguished-points* memiliki panjang yang bervariabel sehingga mengurangi jumlah false *alarm* sehingga menjadi lebih efisien.



Gambar 1: tabel klasik berukuran *m x t [1]*



Gambar 2: Rainbow table berukuran *mt x t [1]*

Pada kedua kasus pada gambar diatas, kolisi dapat

terjadi penggabungan di kelompok kunci *mt* dan kolisi dapat terjadi dengan sisanya *m*(*t*-1). Hanya diperlukan setengah operasi untuk melakukan pencarian kunci pada rainbow table [1]

3. HASIL PERCOBAAN DAN PEMBAHASAN

Penelitian ini menggunakan suatu software yang dikembangkan oleh Zhu Shuanglei**.** Software tersebut bernama Rainbow Crack. Software tersebut terdiri dari beberapa program. Program yang digunakan pada penelitian ini adalah program yang digunakan untuk menghasilkan Rainbow table (rtgen), program untuk mengurutkan rantai (rtsort) dan program untuk melakukan pencarian pada tabel (rcrack). Tabel yang baru diciptakan diurutkan terlebih dahulu untuk mempercepat proses pencarian pada tabel.

Komputer yang digunakan untuk percobaan ini merukapan komputer *desktop* dengan spesifikasi yang dapat dilihat pada tabel 1

Prosesor	Intel Pentium 4 3.0Ghz
Memori	512 MB
Hardisk	80GB
Sistem Operasi	Ubuntu Linux 7.10 Kernel 2.6.20

Tabel 1: Spesifikasi Komputer untuk melakukan percobaan

Hasil ini disusun berdasakarkan penelitian sebagai berikut, Pertama akan dibuat Beberapa tabel yang dibuat dengan NTLM dan hash yang dibuat dengan fungsi MD5. Setelah itu akan dilakukan beberapa pencarian pada tabel tersebut. Waktu dan keberhasilan merupakan komponen utama yang akan di sampaikan pada makalah ini.

3.1. NTLM

NTLM adalah suatu protokol yang dibuat oleh Microsoft dan banyak digunakan pada produk Microsoft, sepertihalnya pada sistem operasi Windows, implementasi Microsoft pada SMTP, POP3 , IMAP, Telnet dll.

Pada makalah ini tidak akan dibahas lebih lanjut mengenai protokol NTLM. Untuk mengetahui lebih detail tentang NTLM pembaca dapat melihatnya pada [3]. Protokol ini dipilih karena sistem operasi windows adalah sistem operasi yang sangat umum digunakan sehingga protokol NTLM sangat banyak digunakan.

Untuk percobaan kali ini akan dibuat sebanyak 5 Rainbow table dengan algoritma LanMan Hash. Tabel tersebut memiliki spesifikasi yang dapat dilihat pada tabel 2

Algoritma Hash	lm
Karakter	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Range palinteks	1 -7
key space	26^1 + 26^2 + 26^3 + 26^4 + 26^5 + 26^6 + 26^7 = 8353082582
t (panjang rantai)	2100
m (jumlah rantai tiap tabel)	8000000
l (bayak tabel)	5
Ukuran Tabel	610.4 MB
waktu prekomputasi	1456 menit 2 detik (1 Hari 16 menit 2 detik)

Tabel 2:Spesifikasi Rainbow tabel untuk algoritma lm)

Untuk percobaan ini dibuat suatu uji kasus dengan menciptakan hash sebanyak 90 buah. 10 Buah dengan panjang 1 karakter, 10 buah dengan panjang 2 karakter dst, dan 10 terkahir adalah hash dengan plainteks menggunakan angka atau karakter yang tidak ada pada tabel. Plainteks untuk kasus uji ini bersifat case sensitif. Artinya akan dicoba untuk kemungkinan pasangan karakter dengan hurup kapital dan huruf kecil.df

Untuk percobaan ini dibuat program kecil yang berfungsi untuk menghasilkan plainteks random dengan panjang 1 sampai 7 karakter dan dengan mengkombinasikan huruf besar dan huruf kecil. Data tersebut berjumlah 70 buah kata yang dimasukan kepada fungsi NTLM hash. Dengan menggunakan rcrack di dapati data seperti pada tabel 3.

Panjang plainteks	1	2	3	4	5	6	7
Plainteks yang ditemukan	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)
Total Waktu disk akses	0.18s	0.47s	0.29s	0.54s	0.40s	0.40s	0.37s
Total waktu kriptanalis	17.54s	33.36s	30.54s	32.47s	15.06s	23.57s	18.00s
Total chain yang dilalui	12090342	22940660	19579244	19673847	8175712	13076671	9986590
False Alarm	6402	14789	14662	17372	7527	12255	9625
Total chain yang dilalui karena false alarm	5901372	11370254	11965044	13790249	7318418	11126129	8518016

Tabel 3: Hasil pencarian pada tabel (*look-up*) yang dilakukan pada LM hash menggunakan program rcrack. Data ditemukan berdasarkan pencarian 10 nilai hash untuk tiap panjang plainteks

Dari hasil pada tabel 3, dapat kita lihat bahwa lamanya proses untuk melakukan kriptanalis tidak ditentukan oleh panjang plainteks. Contohnya pada plainteks dengan panjang 5 karakter waktu yang dibutuhkan untuk mencari 10 hash adalah 15.06 detik sedangkan waktu yang dibutuhkan untuk mencari 10 hash dengan panjang plainteks 1 adalah 17.54 atau 2.48 detik lebih cepat. Hal tersebut dapat terjadi karena beberapa nilai hash yang panjang plainteks-nya lebih panjang dapat ditemukan menggunakan jumlah tabel yang lebih sedikit dibanding dengan beberapa cipherteks dengan panjang yang lebih sedikit dan memerlukan jumlah tabel yang lebih banyak. Dari percobaan ini dapat disimpulkan bahwa kelompok panjang plainteks dengan jumlah tertentu, mungkin saja tidak terletak pada tabel yang sama. Hal tersebut juga berlaku pada False Alarm, panjang plainteks tidak menentukan kemungkinan false alarm yang terjadi.

Hal yang menarik adalah untuk karakter yang lebih panjang dari 7 karakter masih dapat ditemukan. tetapi hash tersebut tidak langsung ditemukan melainkan dipecah menjadi beberapa karakter. contoh untuk plainteks dengan panjang 10 karakter “hjdDLSOpds” ditemukan menjadi “HJJDLSO” dan “PDS”. Ditemukan bahwa dengan tabel prekomputasi ini, nilai hash yang lebih panjang dari 7 karakter dipecah menjadi 2 nilai hash. Selain itu huruf kecil maupun kapital tidak mempengaruhi pencarian dalam algoritma ini. dapat dilihat pada tabel 2 bahwa himpunan karakter yang digunakan adalah huruf alfabet kapital, tetapi dapat memecahkan cipherteks yang *case-sensitive*. Hal ini disebabkan algoritma lm mengkonversi huruf kecil menjadi huruf kapital.

Untuk kasus terburuk, yaitu kasus plainteks yang dicari tidak terdapat pada tabel, waktu kriptanalis yang diperlukan adalah 18.30s. hal tersebut berbeda – beda untuk plainteks yang berbeda pula.

3.2. **MD5**

MD5 adalah fungsi hash satu-arah yang dibuat oleh Ronald Rivest pada tahun 1991. Algoritma ini menerima masukan beruba pesan dengan ukuran sembarang dan menghasilkan *message disgest* yang panjanganya 128 [4]. Makalah ini juga tidak akan membahas lebih lanjut mengenai algoritma MD5. untuk mempelajari lebih lanjut tentang algoritma MD5 dapat dibaca pada [4]. Alasan dipilihnya algroritma MD5 adalah algoritma hash ini sangat populer digunakan. Baik untuk autentikasi user di *web* maupun digunakan dalam autentikasi user pada sistem operasi Linux.

Sama halnya dengan percobaan pada algoritma NTLM, percobaan dengan algoritma MD5 ini juga menggunakan Tabel dengan spesifikasi yang dapat dilihat pada tabel 2, hanya algoritma-nya saja yang diganti dari LM menjadi MD5

Untuk percobaan kali ini digunakan kasus test yang berbeda. Tidak seperti LM, algoritma MD5 merupakan algoritma yang *case-sensitive*. Oleh karena itu kasus test disesuaikan dengan jenis huruf yang digunakan untuk menghasilkan tabel; yaitu karakter huruf besar dengan panjang plainteks 1-7 karakter.

Hasil dari percobaan ini dapat dilihat pada tabel 4

Panjang plainteks	1	2	3	4	5	6	7
Plainteks yang ditemukan	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)	10/10 (100%)
Total Waktu disk akses	0.40s	0.89s	0.48s	0.60s	0.24s	0.36s	0.39s
Total waktu kriptanalis	23.86s	35.01s	29.18s	28.89s	15.47s	19.43s	21.37s
Total chain yang dilalui	12104489	18654466	15384783	14724597	7490898	9549948	11183006
False Alarm	11697	17677	14454	14277	7053	8964	10600
Total chain yang dilalui karena false alarm	10554664	14337556	12360040	12782759	7205913	8878089	9031406

Tabel 4: Hasil pencarian pada tabel (*look-up*) yang dilakukan pada MD5 hash menggunakan program rcrack. Data ditemukan berdasarkan pencarian 10 nilai hash untuk tiap panjang plainteks

Dari hasil penelitian tersebut didapatkan hal yang sama seperti pada hasil pencarian menggunakan tabel LM. waktu yang digunakan untuk melakukan kriptkanalis terhadap 10 md5 hash tidak dipengaruhi oleh panjang plainteks. sama juga dengan chain yang harus dilalui semua tidak tergantung dengan panjang plainteks. Dari percobaan ini didapat bahwa sekumpulan plainteks dengan panjang yang sama, belum tentu terdapat pada tabel yang sama.

Hal menarik lainya adalah dengan data yang terbatas dapat dilihat bahwa waktu yang digunakan untu mencari plainteks dengan panjang 5 lebih singkat dibanding yang lain. Hal ini belum tentu berlaku terhadap semua kasus karena untuk percobaan ini penilis hanya mencoba pada 10 cipherteks untuk tiap panjang karakter plainteks.

Untuk kasus terburuk yaitu kasus dimana cipherteks tidak dapat dipecahkan melalui ke lima tabel, waktu yang diperlukan untuk menemukan 10 plaiteks adalah 193.70 detik dengan banyak chain yang dilalui sebesar 110092550. Kasus tersebut didapat dari mencari plainteks sebesar 7 karakter dengan huruf yang tidak ada pada himpunan huruf yang membuat tabel tersebut.

4. KESIMPULAN DAN SARAN

4.1. **Kesimpulan**

Dari percobaan yang telah dilakukan, didapat kesimpulan sebagai berikut :

- Metode Time-Memory Trade-Off sangat efektif dan efisien untuk memecahkan nilai hash. Tetapi waktu yang dibutuhkan untuk menghasilkan tabel adalah sangat lama. Metode ini cocok untuk memecahkan banyak hash
- Panjang plainteks tidak mempengaruhi waktu yang dibutuhkan untuk melakukan kriptanalis
- Pada algoritma LM, cipherteks tidak mempedulikan huruf kapital atau bukan, sedangkan pada MD5 lebih bersifat *case-sensitive*.
- Sekumpulan chipherteks dengan panjang plainteks yang sama belum tentu berada pada tabel yang sama.
- Rainbow Tabel menghilangkan rantai yang saling bergabung dengan membuat fungsi reduksi dengan jumlah tertentu

4.2. **Saran**

- Karena waktu yang lama dalam menghasilkan Rainbow Table. Maka apabila dibentuk komunitas online dimana semua orang dapat bekerja sama menghasilkan Rainbow Table. maka waktu untuk menghasilkan tabel tersebut akan menjadi singkat.

DAFTAR REFERENSI

1[1] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. Laboratoire de SecuritÂ´e et de Cryptographie (LASEC). Ecole Polytechnique FÂ´edÂ´erale de Lausanne FacultÂ´e I&C,

2[2] M. E. Hellman. A cryptanalytic time-memory trade off. IEEE Transactions on Information Theory, IT-26:401–406, 1980.

3[3] http://davenport.sourceforge.net/ntlm.html

4[4] Munir, Rinaldi Ir. M.T. *Diktat kuliah IF5054 Kriptografi.* Program Studi Teknik Informatika, Sekolah Tinggi Elektro dan Informatika, Institut Teknologi Bandung.