UNIVERSITAS INDONESIA

ANALISA DAN PERBANDINGAN HASIL IMPLEMENTASI ALGORITMA MD5 DAN SHA-1 PADA SISTEM KEAMANAN AUTENTIKASI SIMPLE-O

SKRIPSI

AHMAD SHAUGI
0806339010

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER
DEPOK
JUNI 2012

UNIVERSITAS INDONESIA

ANALISA DAN PERBANDINGAN HASIL IMPLEMENTASI ALGORITMA MD5 DAN SHA-1 PADA SISTEM KEAMANAN AUTENTIKASI SIMPLE-O

SKRIPSI

Skripsi ini diajukan untuk melengkapi
sebagian persyaratan untuk menjadi Sarjana Teknik

AHMAD SHAUGI
0806339010

FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER
DEPOK
JUNI 2012

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	**: Ahmad Shaugi**
NPM	**: 0806339010**
Tanda tangan	**:** 
Tanggal	**: 29 Juni 2012**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :

Nama : Ahmad Shaugi

NPM : 0806339010

Program Studi : Teknik Komputer

Judul Skripsi : Analisa dan Perbandingan Hasil Implementasi Algoritma MD5 dan SHA-1 pada Sistem Keamanan Autentikasi Simple-O

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia.

Pembimbing : Dr. Ir. Anak Agung Putri Ratna M.Eng. (.....)

Penguji 1 : Ir. A. Endang Sriningsih M.T Si (.....)

Penguji 2 : Prima Dewi Purnamasari ST., MT., MSc. (.....)

Ditetapkan di : Depok

Tanggal : 29 Juni 2012

KATA PENGANTAR

Bismillahirrahmanirrahim

" Qalu subhanaka la 'ilma lana "illa ma 'allamtana "innaka "anta al-'alimu al-hakimu "

"Mereka menjawab, 'Mahasuci Engkau, tidak ada yang kami ketahui selain apa yang telah engkau ajarkan kepada kami. Sungguh, Engkaulah Yang Maha Mengetahui, Mahabijaksana.," (QS 2:32)

Segala puji bagi Allah, yang menjadi tempatku bergantung dan memohon pertolongan, dan yang hanya atas izin dan ridho-Mu, maka segala hal yang aku usahakan dapat bermanfaat dan menjadi baik bagi diriku. Segala puji bagi Allah, yang atas izin dan rahmat-Mu ya Allah, dan atas segala kemudahan dan berkah yang Engkau berikan, maka aku dapat menyelesaikan penyusunan Skripsi ini.

Atas segala dukungan dan bantuan yang saya terima selama melakukan penyusunan Skripsi ini, maka saya ingin menyampaikan rasa terima kasih saya kepada :

1. Terima kasih kepada Dr. Ir. Anak Agung Putri Ratna M.Eng., atas segala bimbingan dan arahan yang diberikan kepada saya selama penyusunan skripsi ini.
2. Kedua orang tua, Ibu dan Ayah-ku, yang sungguh karena doa kalian, dan segala pengorbanan kalian, maka aku bisa mencapai segala apa yang ada pada diriku saat ini. Terima kasih atas kasih sayang, bimbingan, dan berbagai pelajaran hidup yang telah kalian berikan kepadaku semenjak aku kecil hingga hari ini.
3. Kakakku yang tersayang, Nabilah, atas segala doa, dukungan, keceriaan, dan kasih sayang yang kamu berikan kepadaku selama ini. Terima kasih juga saya ucapkan untuk kakak ipar saya, Yusuf, atas segala bantuan yang diberikan kepada saya hingga saat ini.

4. Terima kasih kepada Alifandi Yudistira, sahabat dan rekan saya dalam penyusunan skripsi ini. Terima kasih kepada sahabat saya Adityo Abdi Nugroho dan Achmad Farisy, atas segala saran dan bantuannya selama pengerjaan skripsi ini.
5. Dan juga terima kasih untuk sahabat-sahabat saya, Ahmad Dahlan, Imam Askolani, Ian Herahman, Slamet Budiyatno, dan seluruh rekan saya di Keluarga Besar Elektro-Komputer 2008 atas segala suka-duka dan berbagai keceriaan lain yang kita bagi bersama.

Saya memohon maaf apabila terdapat kesalahan dan berbagai kekurangan lain dalam penulisan Skripsi ini. Kritik dan saran yang membangun, sangat saya harapkan sehingga dapat membantu saya dalam melakukan penyusunan makalah yang lebih baik lagi di kemudian hari.

Depok, 2012

Ahmad Shaugi Shahab

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademika Universitas Indonesia, saya yang bertanda tangan di bawah ini :

Nama	: Ahmad Shaugi
NPM	: 0806339010
Program Studi	: Teknik Komputer
Departemen	: Teknik Elektro
Fakultas	: Teknik
Jenis Karya	: Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif** (***Non-exclusive Royalty Free Right***) atas karya ilmiah saya yang berjudul :

ANALISA DAN PERBANDINGAN HASIL IMPLEMENTASI ALGORITMA MD5 DAN SHA-1 PADA SISTEM KEAMANAN AUTENTIKASI SIMPLE-O

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non Eksklusif ini Universitas Indonesia berhak menyimpan, mengalih media / formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan skripsi saya selama tetap mencantumkan nama saya sebagai pemegang Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 29 Juni 2012

Yang menyatakan

(Ahmad Shaugi)

ABSTRAK

Nama : Ahmad Shaugi

Program studi : Teknik Komputer

Judul : Analisa dan Perbandingan Hasil Implementasi Algoritma MD5 dan SHA-1 pada Sistem Keamanan Autentikasi Simple-O

Simple-O, suatu aplikasi *essay grading* yang dikembangkan di Departemen Teknik Elektro Universitas Indonesia, menggunakan algoritma MD5+salt untuk melakukan proteksi terhadap data password user yang tersimpan pada *database*-nya. Namun dengan banyaknya kelemahan yang terdapat pada algoritma MD5, maka diterapkan algoritma SHA-1+salt pada aplikasi ini, yang kemudian dibandingkan dengan algoritma sebelumnya yaitu MD5+salt. Pengujian meliputi pengukuran waktu dan estimasi waktu *brute force* untuk masing-masing algoritma, serta mengukur *processing time* dan *CPU usage* saat melakukan login ke dalam system. Hasil pengujian brute force menunjukkan bahwa penerapan algoritma SHA-1 lebih kuat terhadap serangan *brute force* dibandingkan dengan MD5. Selisih *processing time* SHA-1+salt dengan MD5+salt berkisar antara 0.001 detik hingga 0.002 detik untuk tiap variasi panjang password. Sedangkan selisih *CPU usage* SHA-1+salt dengan MD5+salt sebesar 0.545%, 0.985%, dan 1.69% masing-masing untuk password sepanjang 8, 9, dan 10 karakter. Hasil ini menunjukkan bahwa penerapan algoritma SHA-1+salt tidak akan membebani kinerja aplikasi Simple-O.

Kata kunci : *keamanan sistem autentikasi Simple-O, algoritma MD5, algoritma SHA-1, brute force, processing time, CPU usage*

ABSTRACT

Name : Ahmad Shaugi

Majors : Computer Engineering

Title : Analysis and Comparison the Result of the MD5 and SHA-1 Algorithm Implementation in Simple-O Authentication System Security

Simple-O, an essay grading application that was developed at the Department of Electrical Engineering University of Indonesia, using MD5+salt algorithm to perform protection for password of user's which stored on its database. But with so many flaws contained in the MD5 algorithm, then SHA-1+salt algorithm was implemented in this application, which is then compared with the previous algorithm MD5+salt. The tests include measurements of time and estimated time of brute force for each algorithm, and measure the processing time and CPU usage when logging into the system. The test results show that the application of brute force algorithm SHA-1 is more robust against brute force attacks than MD5. Difference in processing time SHA-1+salt with MD5+salt was ranged from 0.001 seconds to 0.002 seconds for each length variation of the password. While the difference in CPU usage of SHA-1+salt with MD5+salt is 0.545%, 0.985%, and 1.69% respectively for the password with 8, 9, and 10 characters length. These results indicate that the implementation of the algorithm SHA-1+salt does not impose on the performance of Simple-O application.

Keywords : *Simple-O authentication system security, MD5 algorithm, SHA-1 algorithm, brute force attack, processing time, CPU usage*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Perkembangan aplikasi berbasis web dalam bidang pendidikan dewasa ini sudah semakin pesat. Berbagai jenis aktifitas pendidikan telah dijalankan dengan memanfaatkan teknologi ini. *E-learning*, atau *electronic learning*, adalah sistem pendidikan yang menggunakan aplikasi elektronik untuk mendukung belajar mengajar dengan memanfaatkan media Internet, jaringan komputer, maupun komputer *standalone.* Pada awalnya *e-learning* hanya berfokus pada sistem belajar jarak jauh (*distance learning*), yang bertujuan untuk mempermudah proses belajar mengajar yang dapat dilakukan dimana saja tanpa harus di kelas dengan memanfaatkan jaringan yang ada.

Seiring dengan berkembangnya teknologi web saat ini, maka cakupan *e-learning*-pun semakin luas, seperti dibuatnya web yang menyediakan berbagai bahan perkuliahan, yang didalamnya bisa juga dilakukan diskusi pelajaran, ujian online, dan lain-lain. Ujian online-pun memiliki beberapa jenis dan bentuk ujiannya, salah satunya adalah metode ujian online *essay grading*.

Essay grading adalah metode yang digunakan untuk melakukan penilaian dalam ujian esai secara otomatis. Tidak seperti ujian pilihan ganda yang hanya menghitung jumlah jawaban yang benar untuk memberi penilaian, dalam *essay grading* metode penilaiannya lebih rumit, karena harus menggunakan suatu algoritma pemrograman khusus untuk melakukan penilaian yang dapat bekerja seperti seorang pemeriksa ujian esai. Dibalik berbagai metode yang digunakan dalam *essay grading* ini, faktor keamanan untuk aplikasi ini juga tidak bisa ditinggalkan.

Simple-O, adalah salah satu contoh aplikasi *essay grading* yang dikembangkan di Departemen Teknik Elektro Universitas Indonesia. Didalam aplikasi ini, digunakan beberapa metode untuk mengamankannya, termasuk pada sistem autentikasinya untuk memberi proteksi terhadap data *username* dan *password*, dengan menggunakan algoritma MD5 untuk meng-*hash password* yang tersimpan dalam *database*. Namun dengan semakin banyaknya kelemahan yang

ditemukan dalam penggunaan algoritma MD5, maka penggunaan algoritma MD5 dalam sistem ini harus ditinjau lebih lanjut. Hal itulah yang menjadi latar belakang diterapkannya algoritma SHA-1 pada sistem autentikasi aplikasi Simple-O dan kemudian dibandingkan dengan penerapan algoritma sebelumnya.

1.2 Tujuan

Tujuan dari skripsi ini adalah:

1. Menguji, membandingkan, dan menganalisis kekuatan algoritma MD5 dan SHA-1 terhadap serangan *brute force.*
2. Menerapkan algoritma *hash* SHA-1 pada sistem autentikasi Simple-O.
3. Menguji, membandingkan, dan menganalisis hasil penerapan algoritma SHA-1 dengan algoritma yang diterapkan sebelumnya, yaitu MD5.

1.3 Pembatasan Masalah

Batasan Masalah pada skripsi ini adalah proteksi sistem autentikasi yang ada pada aplikasi berbasis web Simple-O. Proteksi yang dilakukan adalah dengan menerapkan algoritma SHA-1 pada sistem autentikasi dan kemudian dibandingkan dengan algoritma MD5 yang diterapkan sebelumnya, yaitu dibandingkan dari sisi kekuatan masing-masing algoritma terhadap serangan *brute force*, dan mengukur *processing time* dan *CPU usage* dari masing-masing penerapan algoritma untuk melihat efek penerapannya terhadap aplikasi Simple-O.

1.4 Metodologi Penelitian

Metodologi penulisan dilakukan dalam beberapa tahapan, yaitu:

a. Studi literatur

   Studi literatur melalui buku-buku, jurnal-jurnal, forum ataupun situs-situs lain di internet.

b. Penerapan dan pengujian algoritma baru

   Setelah proses pengumpulan informasi, maka berikutnya akan memasuki tahap penerapan algortima *hash* baru pada sistem autentikasi Simple-O, menguji algoritma yang baru diterapkan itu dengan mengukur kekuatannya

terhadap serangan *brute force* dan mengukur efek penerapan algoritma tersebut yang dilihat dari *processing time* dan *CPU usage* saat dilakukan login ke dalam sistem. Kemudian melakukan analisa terhadap hasil pengujian tersebut.

c. Kesimpulan

Dari hasil analisa, maka akan dibuat kesimpulan mengenai hasil pengujian terghadap algoritma *hash* yang baru diterapkan pada sistem autentikasi Simple-O.

1.5 Sistematika Penulisan

Skripsi ini akan dibagi menjadi 5 bab, yaitu :

a. Bab 1 : Pendahuluan

Berisi latar belakang masalah, tujuan, pembatasan masalah, metodologi dan sistematika penulisan

b. Bab 2 : Web Based Application, Kriptografi, dan Simple-O

Membahas dasar-dasar aplikasi berbasis web, dasar-dasar keamanan web, e-learning, *essay grading*, *hash*, dan Simple-O.

c. Bab 3 : Perancangan Sistem Software dan Hardware

Meliputi perancangan sistem untuk pengujian, tools-tools yang digunakan, serta metode pengujian.

d. Bab 4 : Pengujian dan Analisa Sistem

Pada bagian ini penulis memaparkan hasil testing yang dilakukan terhadap sistem, dan menganalisa hasil testing tersebut.

e. Bab 5 : Kesimpulan

Kesimpulan dari skripsi ini.

BAB II
WEB BASED APPLICATION, KRIPTOGRAFI, DAN SIMPLE-O

Simple-O, sebagai suatu aplikasi berbasis web bersinggungan dengan banyak hal yang berkaitan dengan bagaimana suatu aplikasi berbasis web berjalan, bagaimana mengamangkannya, apa-apa saja celah keamanannya, baik sebagai aplikasi berbasis web secara umum maupun sebagai suatu aplikasi e-learning, dan seterusnya. Pada bab ini akan dijelaskan berbagai teori dasar mengenai web based application, web security, kriptografi,penetration test, dan Simple-O.

Selain itu pada akhir bab ini akan dijelaskan juga mengenai beberapa tools yang digunakan untuk pada skripsi ini, baik yang digunakan untuk menjalankan aplikasi Simple-O, seperti Ubuntu 11.10-Oneiric Ocelot dan LAMP, serta yang digunakan dalam pengujian, yaitu Hashcat

2.1 Web Based Application

Web based application atau aplikasi berbasis web ialah segala jenis aplikasi yang untuk dapat menjalankannya dibutuhkan akses ke internet. Definisi lain dari *web based application* adalah suatu aplikasi yang sebagian atau seluruh bagian dari software aplikasi tersebut didownload dari Web tiap kali aplikasi tersebut dijalankan [1]. Berdasarkan kedua definisi tersebut, maka *web based application* dapat juga diartikan sebagai suatu aplikasi yang tidak perlu di-*install* terlebih dahulu di desktop PC untuk dapat menjalankannya, namun cukup dijalankan dengan bantuan *web browser* atau aplikasi *rich-client*, yang oleh karena itu *web based application* disebut juga sebagai *browser-based application* atau *client-based application*.

Di dalam konteks *web based application* sebagai *browser-based application*, instruksi dari program yang terkandung di halaman web disusun kembali oleh web browser untuk kemudian dapat ditampilkan. Sedangkan bila suatu *web based application* dijalankan menggunakan sistem *client*-based application, maka baik tiap-tiap sesi atau seluruh sesi dari *web application* tersebut di download oleh *client* dari *server* yang terhubung dengan *client*

tersebut. Ini mirip dengan arsitektur "*client/server*" sebelum munculnya internet, kecuali bila *server* tersebut berada di internet dan bukan di jaringan lokal.

Web based application ini sering kali dibandingkan dengan *desktop based application*. Dari beberapa perbedaan yang dimiliki kedua jenis aplikasi, maka ada satu perbedaan yang tergolong suatu perbedaan yang mendasar, karena perbedaan ini berkaitan dengan penggunaan kapasitas hard drive suatu komputer. Untuk dapat menjalankan *desktop based application*, maka pertama-tama aplikasi ini harus di-*install* terlebih dahulu. Instalasi aplikasi pada komputer ini tentu saja akan membutuhkan alokasi ruang penyimpanan di dalam *hard drive* komputer. Sedangkan pada *web based application*, aplikasi yang akan dijalankan tersebut tidak perlu diinstall terlebih dahulu pada komputer, sehingga tidak membutuhkan alokasi ruang penyimpanan pada hard drive komputer. Oleh karena itu, penggunaan *web based application* akan menghemat pemakaian kapasitas hard drive yang ada pada komputer.

Keunggulan-keunggulan lain *web based application* atas *desktop based application* terlihat juga dari sisi kepemilikan lisensi yang tidak dibutuhkan pada penggunaan aplikasi berbasis web, ketersediaan aplikasi yang dapat diakses kapanpun dimanapun, kompatibilitas terhadap PC user yang tinggi, media akses yang cukup beragam, dan lain-lain [2].

Namun meski memiliki banyak kelebihan bila dibandingkan dengan *desktop application*, *web based application* juga memiliki kelemahan. Kelemahan *web based application* ini terkait dengan keharusan untuk mengkases jaringan lokal ataupun internet agar aplikasi tersebut dapat dijalankan, sehingga apabila terdapat gangguan pada jaringan internet yang digunakan, maka gangguan tersebut akan secara langsung berdampak pada kinerja *web based application*.

2.1.1 Essay grading

Salah satu contoh dari penerapan aplikasi berbasis web ialah aplikasi *essay grading* yang diterapkan dalam aplikasi e-learning. *Essay grading* adalah suatu sistem penilaian ujian esai secara otomatis yang dilakukan dalam suatu aplikasi e-learning. Aplikasi ini merupakan hasil pemikiran dimana proses penilaian untuk jawaban esai merupakan hal yang memakan waktu, sehingga dibuatlah suatu

aplikasi yang berguna untuk melakukan penilaian jawaban esai ini secara otomatis.

Penilaian jawaban esai secara otomatis ini bukanlah suatu hal yang mudah, karena dibutuhkan metode yang tepat sehingga penilaian yang dilakukan ini merupakan suatu hasil penilaian yang objektif. Salah satu metode yang digunakan dalam aplikasi *essay grading* adalah *Latent Semantic Analysis* (LSA). LSA adalah teknik pengumpulan informasi yang sangat powerful, yang menggunakan statistic dan aljabar linear untuk menemukan makna "laten" dari suatu teks [3].

Hal paling utama yang diperhatikan pada suatu aplikasi *essay grading* adalah seberapa akurat sistem dapat sesuai dengan penilaian jawaban oleh manusia (*human raters)*, seperti pada algoritma LSA yang memiliki *human raters* 85% - 91% [4].

2.2 Web Security [5]

Keamanan web atau *web security* adalah metode ataupun proses untuk mengamankan suatu web. Proses ini berupa suatu mekanisme yang bekerja untuk mencegah akses, modifikasi, ataupun aktifitas lain oleh user yang tidak dikenal, terhadap data-data dari web yang tersimpan secara online.

World Wide Web (WWW) secara mendasar adalah suatu aplikasi *client/server* yang berjalan diatas internet atau TCP/IP intranet, sehingga keamanan suatu web masih terkait dengan keamanan jaringan komputer secara umum. Hanya saja keamanan yang dibutuhkan suatu web, tidak pernah diperhatikan dalam konteks keamanan jaringan dan komputer sebelumnya, sehingga ini menjadi sebuah tantangan baru dalam usaha untuk membuat suatu sistem keamanan web.

Dengan tetap terkaitnya topik sistem keamanan web dengan keamanan jaringan secara umum, maka keamanan suatu web juga harus memenuhi standar *security service* yang ada pada sistem keamanan jaringan. Secara umum terdapat enam dasar *security service* pada sistem keamanan jaringan yang harus dipenuhi, yang kemudian juga harus dipenuhui dalam sistem keamanan suatu web, yaitu:

- *Confidentiality* : semua data yang tersimpan pada aplikasi berbasis web harus tetap terjaga dalam kondisi apapun.

- *Authentication* : user dengan hak akses sajalah yang dapat mengakses web, terlebih lagi dalam mengakses *web server*.
- *Integrity* : memastikan bahwa semua data yang tersimpan dalam aplikasi berbasis web adalah data yang terjamin kebenarannya, tanpa ada modifikasi sedikitpun dari pihak lain yang tidak berwenang.
- *Nonrepudiation* : memastikan bahwa user yang asli tidak dapat menyangkal data yang terdapat pada aplikasi berbasis web, dan sebaliknya aplikasi berbasis web dapat menunjukkan identitas kepada user.
- *Access control* : terdapat mekanisme pembatasan akses ke *web server* dan aplikasinya, yang dilakukan oleh pihak lain yang tidak memiliki hak akses, melalui jalur-jalur komunikasi.
- *Availability* : aplikasi berbasis web harus dapat diakses oleh user yang berwenang, sesuai dengan spesifikasi waktu yang diminta oleh user tersebut.

Servis-servis di ataslah yang harus dijadikan dasar acuan dalam perancangan suatu sistem keamanan web, sehingga dapat dicapai suatu sistem pengamanan maksimal terhadap web tersebut.

2.2.1 Web Vulnerabilities

Web vulnerabilities adalah celah-celah dari sistem keamanan web yang memungkinkan bagi seorang hacker ataupun cracker untuk menembus sistem keamanan dari web tersebut. Keberadaan celah-celah keamanan ini bisa disebabkan oleh banyak hal. Untuk suatu *web application* yang tidak diterapkan suatu sistem keamanan dalam sistemnya, maka sudah jelas akan meninggalkan celah besar yang bisa dimanfaatkan oleh hacker maupun cracker untuk menembusnya. Sedangkan untuk suatu web yang sudah memiliki sistem keamanan di dalamnya, celah ini bisa saja muncul karena terlewatnya satu sisi keamanan dari web tersebut yang tidak diperhatikan sebelumnya saat perancangan

sistem keamanan, ataupun bisa juga dikarenakan meningkatnya kemampuan seorang hacker ataupun cracker untuk menembus sistem keamanan suatu web.

Target serangan seorang cracker ialah *electronic assets. Electronic assets* merupakan suatu hal yang sangat berarti baik bagi seseorang, bagi suatu perusahaan, ataupun instansi lain yang memanfaatkan suatu sistem jaringan komputer, dimana *electronic assets* ini tersimpan dalam bentuk kumpulan data dan program [6].

2.2.2 E-learning Security

Proteksi terhadap suatu aplikasi e-learning bukanlah hal yang mudah. Terdapat beberapa tantangan ketika akan dilakukan pengamanan terhadap suatu aplikasi e-learning. Tantangan ini muncul dari dua perspektif yang berbeda, yaitu dari sisi penyelenggara e-learning dan pengguna aplikasi e-learning [7].

Isu-isu keamanan dalam suatu aplikasi e-learning tentunya bersesuaian dengan isu-isu dasar yang harus dipenuhi suatu sistem keamanan dalam *web application*. Semisal kemungkinan manipulasi yang dilakukan oleh orang luar ataupun siswa untuk memasuki sistem yang ada. Hal ini bersesuaian dengan isu keamanan *confidentiality dan authentification*, dimana, misalnya akses bagi orang yang berperan sebagai pengajar harus berbeda dengan seorang siswa, dan oleh karena itu harus ada langkah pengamanan untuk hal ini, sehingga tidak ada siswa atau siapapun yang dapat memanipulasi informasi yang tersimpan pada sistem [7].

Integritas suatu data yang tersimpan dalam sistem e-learning juga harus dijaga keasliannya. Suatu data yang telah diinput ke dalam aplikasi, harus dipastikan tidak dapat dimodifikasi lagi (misal jawaban ujian) sehingga terjaga keasliannya, kecuali untuk hal-hal yang memang bersifat fleksibel untuk dimodifikasi. Kepastian bahwa aplikasi dapat diakses dari manapun juga harus menjadi hal yang diperhatikan, sehingga availability dari sistem dapat terpenuhi [7].

Berbagai riset yang dilakukan mengenai *e-learning*, menyimpulkan bahwa untuk dapat mencegah serangan terhadap suatu sistem aplikasi *e-learning*, maka diperlukan pengendalian akses terhadap sistem itu. Namun beberapa teknologi pengendalian akses ini tidak lagi menjadi sesuatu yang efektif untuk mencegah

serangan dari luar, karena terdapat celah keamanan ketika suatu serangan bisa juga dilakukan dari dalam sistem oleh orang dalam. Oleh karena itu diperlukan manajemen keamanan informasi yang tepat untuk dapat memastikan keberhasilan implementasi dari suatu sistem keamanan e-learning [7].

2.3 Kriptografi

Untuk dapat mengetahui definisi dari kriptografi, terlebih dahulu harus diketahui apa itu kriptologi. Kriptologi adalah studi yang bertujuan untuk mengamankan suatu komunikasi, dimana didalamnya meliputi kriptografi dan kriptanalisis. Kriptografi adalah suatu metode untuk mengubah suatu *plaintext* dengan menyandikannya menjadi bentuk *ciphertext* yang berbeda sama sekali dengan aslinya. Sedangkan kriptanalisis adalah cabang kriptologi yang bertujuan mengurai suatu *ciphertext* untuk mengembalikan informasi didalamnya, atau menguji otentikasi dari suatu informasi yang telah terenkripsi [8].

Ciphertext merupakan pesan acak yang merupakan output dari proses kriptografi. Pesan yang telah diubah tersebut hanya bisa diurai kembali (dekripsi) dengan kunci yang dimiliki oleh pesan tersebut.

Suatu sistem kriptografi biasanya terklasifikasi atas tiga dimensi, yaitu [8]:

1. **Tipe operasi yang digunakan untuk mengubah *plaintext* menjadi *ciphertext***. Seluruh metode kriptografi (enkripsi, *hash*, dan lain-lain) bekerja bedasarkan dua prinsip dasar, yaitu substitusi dan transposisi. Substistusi berarti memetakan setiap elemen penyusun *plaintext* (bit, huruf, kumpulan bit atau huruf) menjadi elemen lain. Sedangkan transposisi adalah mengubah susunan asli dari suatu *plaintext*. Beberapa sistem mengkombinasikan metode substitusi dan transposisi yang dilakukan secara berulang. Semua perubahan yang dilakukan terhadap *plaintext* tidak boleh mengubah atau bahkan menghilangkan informasi asli dari *plaintext* tersebut.
2. **Jumlah kunci yang digunakan**. Apabila suatu sistem menggunakan dua buah kunci yang sama pada sisi pengirim dan penerima, maka sistem tersebut disebut dengan simetrik. Apabila sistem tersebut menggunakan

dua buah kunci yang berbeda pada sisi pengirim dan penerima, maka sistem tersebut disebut asimetrik.

3. **Cara pemrosesan *plaintext***. Satu **block cipher**, akan menerima satu blok input dalam satu waktu, dan menghasilkan satu block output untuk tiap inputan. Sedangkan **stream cipher** akan memproses suatu elemen input secara kontinyu, dan akan menghasilkan output dari elemen tersebut selama inputan terus diberikan. Block cipher adalah algoritma enkrispi simetrik yang mengubah bit *plaintext* menjadi suatu block *ciphertext* yang memiliki panjang yang sama dengan inputnya. Stream cipher adalah algoritma enkripsi simetrik dimana output *ciphertext* diproduksi bit-per-bit atau byte-per-byte dari suatu input stream *plaintext*.

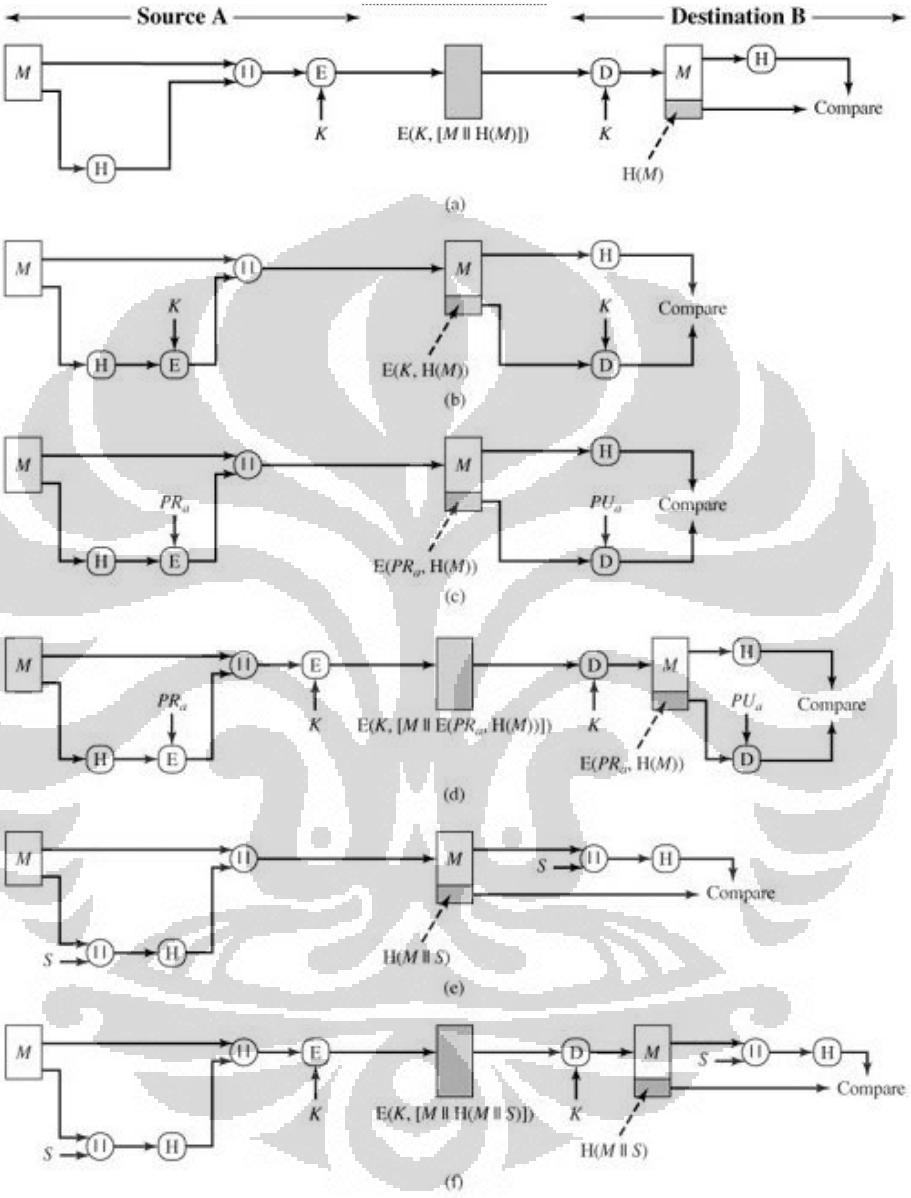
2.3.1 *Hash Function* [8]

Suatu mekanisme *message authentication* atau *digital signature* memiliki dua level yang berbeda bila dilihat dari fungsinya, yaitu *lower level* yang berfungsi untuk menghasilkan suatu kode yang digunakan sebagai otentikator dari suatu pesan, dan *high level* yang menggunakan fungsi dari *low level* untuk dapat digunakan oleh *receiver* agar kemudian dapat memverifikasi keotentikan dari pesan tersebut.

Ada tiga buah metode dari fungsi *low level* yang digunakan untuk menghasilkan suatu otentikator. Fungsi pertama adalah *message encryption*, dimana *ciphertext* dari keseluruhan pesan digunakan sebagai otentikator. Kedua adalah *message authentication code* (MAC), dimana fungsi dari pesan dan kunci rahasia yang menghasilkan *fixed-length value* digunakan sebagai otentikator. Dan yang ketiga adalah *hash function*, yang merupakan suatu fungsi yang memetakan suatu pesan dengan panjang beragam, menjadi suatu *fixed-length hash value*, yang kemudian digunakan sebagai otentikator.

Hash function akan menghasilkan suatu *hash code*. Jika MAC membutuhkan suatu kunci khusus untuk melakukan pengkodean, maka *hash function* tidak membutuhkan suatu kunci khusus untuk melakukan pengkodean. *Hash code* merupakan fungsi dari seluruh bit pada suatu pesan, yang digunakan untuk

mendeteksi error dari pesan tersebut, karena perubahan satu bit saja dari pesan yang asli, akan menghasilkan *hash code* yang berbeda.



Gambar 2.1 Penggunaan dasar dari suatu fungsi *hash* [8]

Gambar 2.1 menunjukkan berbagai penggunaan dasar dari *hash code* sebagai suatu otentikator. Gambar 2.1a menunjukkan suatu pesan dan *hash code* dari pesan tersebut di enkripsi dengan enkripsi simetrik oleh pengirim A dan kemudian dikirimkan ke penerima B. Karena hanya A dan B yang mengetahui kunci

rahasianya, maka pesan pastilah dari A, dan tidak mengalami perubahan. Hal ini dapat dibuktikan oleh B dengan cara mencari nilai *hash code* dari pesan yang dikirimkan tersebut, kemudian membandingkan nilai *hash code* tersebut dengan *hash code* yang dikirimkan bersama dengan pesan.

Gambar 2.1b menunjukkan bahwa hanya *hash code* saja yang dienkripsi menggunakan enkripsi simetrik oleh A, untuk kemudian dikirim bersama pesan asli menuju B. Hal ini akan mengurangi beban proses yang dilakukan sistem ketika akan mengenkripsi pesan asli dan *hash code*-nya, namun akan menghilangkan sisi *confidentiality* dari pesan yang dikirim tersebut.

Gambar 2.1c menunjukkan bahwa hanya *hash code* yang dienkripsi, hanya saja kini menggunakan *public-key encryption*. Metode ini akan menghasilkan suatu *digital signature*, karena hanya pengirim saja yang memiliki kunci rahasia untuk menghasilkan *hash code* yang terenkripsi. Inilah esensi dari penggunaan *digital signature*.

Pada Gambar 2.1d prosesnya hampir sama dengan teknik sebelumya, hanya saja *hash code* yang telah dienkripsi menggunakan public-key, dienkripsi kembali bersama dengan pesan aslinya dengan enkrispi simetrik, untuk kemudian dikirim dari A menuju B. Teknik ini merupakan teknik yang paling banyak digunakan.

Gambar 2.1e menunjukkan bahwa *hash function* dapat digunakan untuk otentikasi tanpa harus melibatkan proses enkripsi didalamnya. Hal ini dimungkinkan bila A dan B, memiliki nilai rahasi yang diketahui bersama, yaitu *S.* Nilai *S* kemudian digabungkan dengan pesan asli *M* untuk kemudian didapatkan nilai *hash* dari pengurutan kedua pesan ini. Setelah nilai *hash* didapatkan, kemudian A akan mengirimkan *hash code* tersebut bersama dengan pesan aslinya ke B, tanpa menyertakan *S.* Untuk dapat membuktikan otentikasi pesan tersebut, maka B akan kembali mengurutkan *M* dan dan nilai *S* yang juga diketahuinya, untuk kemudian dicari nilai *hash*-nya, dan dibandingkan dengan nilai *hash* yang sebelumnya dikirimkan oleh A. *Confidentiality* dari metode ini dapat ditambahkan dengan meletakkan proses enkripsi dari pesan M dengan nilai *hash* pengurutan M dan S sebelum dikirimkan kepada B, seperti terlihat pada Gambar 2.1f.

Saat *confidentiality* tidak terlalu menjadi kebutuhan, maka metode (b) dan (c) dapat dijadikan sebagai pilihan. Meskipun demikian, berbagai macam teknik yang tidak menggunakan enkripsi seperti pada Gambar 2.1e juga semakin berkembang. Hal ini disebabkan oleh beberapa hal, seperti proses enkripsi yang cenderung lamban, penggunaan hardware untuk enkripsi yang tidak murah dan tidak optimal untuk data berukuran kecil, dan metode enkripsi yang harus dipatenkan akan meningkatkan pengeluaran untuk paten dan lisensi.

2.3.2 Security dari *Hash Function*

Ada dua macam pendekatan yang dapat digunakan untuk menyerang sisi keamanan dari *hash function*. Dua macam pendekatan itu adalah *brute force attack* dan kriptanalisis. Seperti pada algoritma enkripsi simetrik, kriptanalisis memanfaatkan sisi kelemahan logika pada algoritma *hash* yang digunakan. Sedangkan kekuatan suatu *hash function* untuk menghadapi *brute force attack* bergantung pada seberapa panjang *hash code* yang dihasilkan oleh suatu algoritma dan kerumitan proses dari algoritma tersebut [8].

Algoritma *hash* biasanya juga dikombinasikan dengan penggunaan *salt*. *Salt* adalah pengacak tambahan dengan nilai tertentu yang biasanya digabungkan dengan *hash code* dalam suatu bentuk fungsi tertentu. Nilai dan fungsi dalam penggunaan *salt* ini bebas untuk ditentukan. Penggunaan *salt* ini bertujuan untuk meningkatkan kerumitan dari *ciphertext* hasil *hash* tersebut, sehingga lebih sulit untuk di *brute force*.

Untuk mendapatkan suatu *hash function* yang memiliki tingkat keamanan yang baik, maka dikembangkanlah berbagai macam jenis algoritma *hash function*. Dari banyak algoritma yang digunakan, algoritma MD5 dan SHA-1 merupakan dua algoritma yang paling umum digunakan.

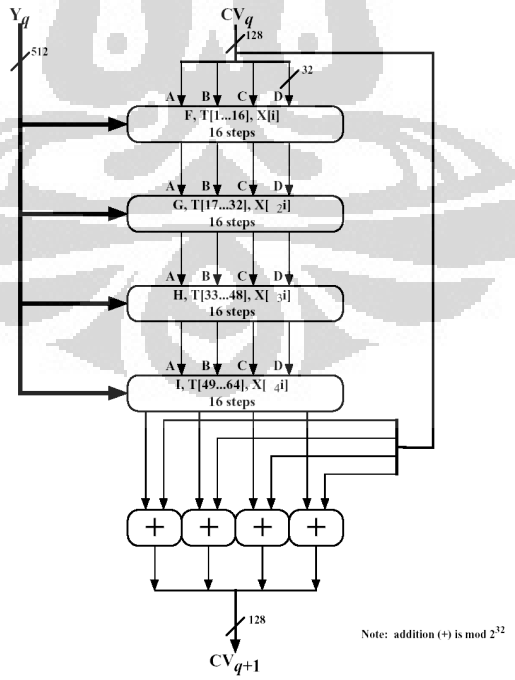
a. Message Digest 5 [9] [10]

Message digest 5 atau MD5 adalah suatu algoritma *message-digest* yang dikembangkan oleh Ron Rivest pada tahun 1991. MD5 merupakan algoritma yang dikembangkan untuk menggantikan algoritma pendahulunya yaitu MD4. Pada awal publikasinya, MD5 termasuk dalam salah satu algoritma *hash* yang paling

baik, namun lima tahun kemudian berbagai kelemahan yang terdapat pada algoritma ini mulai ditemukan [9].

Pada tahun 1996, Dobertian menemukan kolisi pada algoritma ini. Delapan tahun berselang, tepatnya pada Maret 2004, terdapat proyek yang bertujuan untuk menemukan kelemahan yang terdapat pada algoritma ini, yaitu proyek MD5CRK. Setahun kemudian, Arjen Lenstra, Xiaoyun Wang, dan Benne de Weger mendemonstrasikan dua sertifikt X.509 dengan kunci publik berbeda namun memiliki nilai *hash* yang sama. Beberapa hari setelahnya, Vlastimil Klima telah mampu menemukan kolisi yang terdapat pada algoritma ini dengan menggunakan komputer tunggal hanya dalam waktu beberapa jam. Pada tahun 2006, Klima mempublikasikan suatu algoritma yang dapat digunakan untuk menemukan kolisi pada MD5 dengan menggunakan komputer tunggal dan hanya membutuhkan waktu satu menit [9].

Algoritma MD5 mengambil input dengan panjang sembarang dan menghasilkan output berupa *fingerprint* atau *digest* sepanjang 128 bit. Input yang diterima oleh algoritma ini akan diproses dalam suatu blok berukuran 512 bit, yang kemudian akan dibagi kedalam16 subblok yang masing berukuran 32 bit. Gambar 2.2 menampilkan proses pengolahan input pada algoritma MD5.



Gambar 2.2 Proses pengolahan input Single 512-bit Block pada MD5 [8]

Pada proses MD5, keluaran dihasilkan akan selalu memiliki panjang 128 bit. Hal ini berlaku untuk inputan yang memiliki panjang tidak terhingga, dan juga untuk inputan bernilai kosong, semuanya akan menghasilkan output *hash* sepanjang 128 bit. Dengan output sepanjang 128 bit ini berarti terdapat 2^{128} kemungkinan hasil proses *hash* atas berbagai inputan yang diberikan. Kondisi ini memberi indikasi awal dimana algoritma ini rentan terhadap situasi dimana *f(x1)=f(x2)*, yang berarti suatu nilai inputan akan memiliki nilai output yang sama dengan suatu inputan yang lain. Situasi inilah yang disebut dengan *collision.*

Pada dasarnya 2^{128} bukanlah jumlah yang sedikit. Namun dengan kemungkinan input yang jauh lebih besar (mulai dari inputan berkarakter kosong hingga inputan dengan panjang karakter tak terhingga), maka kemungkinan terjadinya *collision* ini tidaklah sedikit. Kelemahan *hash* yang dimiliki MD5 juga semakin memperbesar peluang terjadinya *collision*. MD5 memiliki kelemahan yang memungkinkan ditemukannya dua file yang memiliki nilai *hash* yang sama hanya dalam waktu singkat [11]. Kelemahan tersebut bersesuain dengan persamaan (1) berikut ini:

$$IF\ MD5(X)=MD5(Y)\ THEN\ MD5(X+q)=MD5(Y+q) \quad \ldots.(1)$$

Karena algoritma MD5 ini digunakan pada aplikasi Simple-O, maka kelemahan yang dimliki oleh MD5 ini juga bisa dimanfaatkan pada aplikasi Simple-O. Apabila ada seseorang yang sebenarnya tidak memiliki hak akses terhadap akun dosen misalnya, maka dengan melakukan *brute force attack* terhadap aplikasi ini orang tersebut bisa saja menemukan *password* dari akun seorang dosen, dan bila dia berhasil menemukan akun tersebut, maka dia sudah memiliki semua hak akses yang dimiliki oleh seorang dosen terhadap berbagai informasi yang terdapat di dalam sistem.

b. Secure Hash Algorithm – 1 [8] [10]

Dalam beberapa tahun terakhir *Secure Hash Algorithm* atau SHA merupakan algoritma *hash* yang paling sering digunakan. SHA dikembangkan oleh *National Institute of Standards and Technology* (NIST) dan dipublikasikan sebagai *Federal Information Processing Standards* (FIPS 180) pada tahun 1993.

Saat kelemahan dari algoritma ini ditemukan, yaitu pada SHA-0, maka berbagai revisipun dilakukan untuk membuat suatu algoritma yang lebih baik lagi, dan revisi itu diterbitkan pada FIPS 180-1 pada tahun 1995 dan menjadi referensi pembuatan algoritma baru yaitu SHA-1.

Algoritma SHA ini terus berkembang. Pada tahun 2002, NIST kembali mengeluarkan revisi atas FIPS dan menghasilkan FIPS 180-2, yang mendifinisikan versi terbaru dari SHA dengan panjang output sebesar 256, 384, dan 512 bit. Algoritma SHA itu adalah SHA-256, SHA-384, dan SHA-512. Ketiga SHA tersebut dikenal dengan SHA-2. Tabel 2.1 menampilkan macam-macam anggota SHA family dan perbedaannya masing-masing.

Tabel 2.1 SHA Family [8]

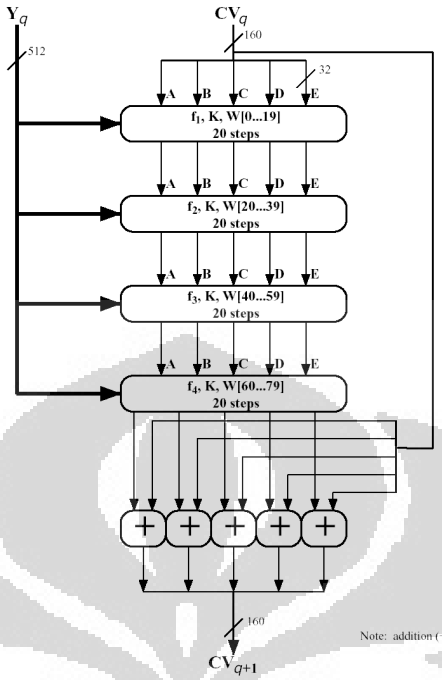
	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80
Security	80	112	128	192	256

Catatan :

1. Semua ukuran dalam bit
2. Keamanan mengacu pada suatu fakta bahwa suat *message digest* dengan panjang *n* akan menghasilkan suatu faktor kerja kolisi sebesar $2^{n/2}$

Seperti terlihat pada tabel diatas, algoritma SHA-1 dapat menerima inputan maksimal kurang dari 64 bit, dan menghasilkan output sepanjang 160 bit. Pembatasan input ini yang menjadi salah satu kekuatan dari algoritma ini, karena dengan pembatasan jumlah input, maka akan mengurangi kolisi yang mungkin terjadi dalam penggunaan algoritma ini.

SHA-1 mengolah input dalam suatu blok berukuran 512 bit, dan kemudian membaginya ke dalam 16 subblok yang masing-masing berukuran 32 bit. Gambar 2.3 menampilkan proses pengolahan input pada algoritma SHA-1.



Gambar 2.3 Proses pengolahan input Single 512-bit Block pada SHA-1 [8]

c. Perbandingan MD5 dan SHA-1

Setidaknya terdapat 4 (empat) poin penting yang membedakan antara algoritma MD5 dan SHA-1, yaitu [12] :

1. MD5 yang dapat mengolah input dengan panjang tak terhingga, menjadikan algoritma ini rentan terhadap kemungkinan terjadinya kolisi, tidak seperti pada SHA-1 yang membatasi inputannya hingga maksimal lebih kecil dari 64 bit.
2. Output SHA-1 yang lebih panjang 32 bit dibandingkan dengan MD5, menjadikan algoritma ini lebih tahan atas serangan *brute force attack*.
3. Dikarenakan lemahnya desain MD5, maka algoritma ini lebih mudah di kriptanalisis dibandingkan dengan SHA-1.
4. MD5 dan SHA-1 sama-sama dapat bekerja baik pada arsitektur 32 bit. Dikarenakan memiliki output yang lebih panjang, maka pemrosesan SHA-1 akan memakan waktu lebih lama bila dibandingkan dengan MD5.

Perbandingan kedua algoritma ini secara lebih mendetail dapat dilihat pada Tabel 2.2 :

Tabel 2.2 Perbandingan algoritma MD5 dan SHA-1 [10]

Subjek Analisa	MD5	SHA-1	Algoritma Yang Lebih Baik
Panjang data output	128 bit	160 bit	SHA-1
Penggunaan memory	30060 byte	30267,6 byte	MD5
Waktu proses	17,233 ms	17,633 ms	MD5
Variansi	$0{,}9013 \times 10^{-3}$	$0{,}4857 \times 10^{-3}$	SHA-1
Waktu untuk *brute force attack*	$6{,}8841 \times 10^{29}$ tahun	$2{,}8825 \times 10^{39}$ tahun	SHA-1

2.4 Penetration Test [13]

Penetration test adalah suatu metode yang digunakan untuk mencari kelemahan dari suatu sistem atau jaringan. Tujuan dari dilakukannya *penetration testing* ini adalah untuk mencoba efektifitas dari suatu sistem keamanan yang telah diimplementasikan pada suatu sistem tertentu, dan kemudian menemukan kelemahan dari sistem keamanan itu serta menemukan kemungkinan masih adanya celah-celah kemanan yang tertinggal atau tidak terproteksi oleh sistem keamanan yang telah diimplementasikan tersebut.

Ada 2 (dua) model pelaksanaan *penetration testing*, yaitu *flaw hypothesis model* dan *attack tree model*. *Flaw hypothesis model* adalah suatu model *penetration testing* dimana sebelum dilakukan testing, tester mengajukan hipotesa-hipotesa kemungkinan kelemahan dari sistem yang akan ditesnya tersebut. Testing model seperti ini biasanya dilakukan untuk lingkungan sistem yang telah dikenal sebelumya, sehingga tester dapat memperkirakan kelemahan dari sistem tersebut.

Testing model yang kedua adalah *attack tree model*. Testing model ini dilakukan untuk suatu sistem yang lingkungannya belum pernah dikenal sebelumnya. Karena tidak mengenal lingkungan sistem ini sebelumnya, maka dilakukanlah pengecekan sistem secara keseluruhan, tidak spesifik seperti testing model yang sebelumnya. Tentunya, karena testing model ini dilakukan secara

menyeluruh, maka waktu yang diperlukan juga menjadi lebih panjang bila dibandingkan dengan metode testing yang sebelumnya.

Salah satu jenis metode yang biasa digunakan dalam *penetration testing* adalah *brute force attack*. Suatu serangan *brute force* dapat didefinisikan sebagai suatu metode trial-and-error yang digunakan untuk bisa mendapatkan suatu informasi yang tidak diketahui, seperti *username* dan *password*. Pada serangan macam ini, suatu software yang bekerja secara otomatis, akan mencoba semua kemungkinan yang ada untuk dapat menemukan *username* dan *password* tadi, sehingga kemudian sistem dapat ditembus dan informasi yang dituju dapat ditemukan. Pada dasarnya kombinasi yang mungkin membentuk *username* dan *password* ini sangatlah banyak, sehingga proses *brute force* ini sangatlah lama. Namun dengan berkembangnya teknologi software dan hardware, maka kini proses *brute force* dapat dilakukan dalam kurun waktu yang lebih singkat [14].

2.5 Simple-O dan Sistem Keamanannya [15]

Simple-O adalah suatu aplikasi berbasis web, yang dikembangkan di Departemen Teknik Elektro Universitas Indonesia, dan digunakan untuk menilai ujian essai secara otomatis. Aplikasi ini bekerja dengan prinsip *client-server*, dimana mahasiswa yang akan melakukan ujian diminta untuk mengakses web Simple-O untuk melakukan ujian.

Aplikasi ini tentunya memerlukan suatu sistem keamanan di dalamnya. Oleh karena itu diterapkanlah sistem keamanan pada aplikasi Simple-O. Ada enam (6) sistem keamanan yang telah diterapkan pada aplikasi, yaitu :

a. *Session* dengan pewaktuan

Session adalah suatu mekanisme penyimpanan data tertentu saat suatu web diakses, seperti menyimpan user id dan *password*. Apabila tidak ada suatu mekanisme yang baik untuk menangani *session*, maka seseorang dapat masuk ke dalam sistem tanpa harus melakukan login terlebih dahulu. Oleh karena itu diterapkanlah sistem *session* dengan pewaktuan ini sehingga seorang user hanya dapat masuk ke dalam sistem setelah melalui proses login terlebih dahulu.

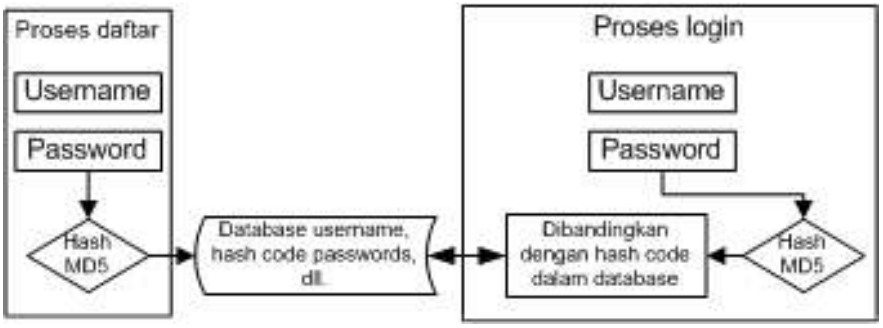
b. *Password strength meter*

Pada aplikasi Simple-O, user diminta untuk membuat sendiri kata sandi untuk user id-nya masing-masing. Karena user membuat *password*-nya masing-masing, maka ada peluang digunakannya suatu *password* yang berkekuatan lemah sehingga *password* itu mudah ditebak oleh orang lain. Maka digunakanlah *password* strength meter yang berfungsi untuk mengetahui kekuatan kata sandi yang dibuat. Suatu *password* dapat dikatakan sebagai *password* yang kuat apabila tersusun dari kombinasi huruf besar dan kecil, angka, dan batas minimal karakter untuk kata sandi.

c. *Hash* kata sandi dengan MD5

Jika tidak diterapkan suatu sistem pengaman tertentu dalam penyimpanan kata sandi di dalam *database*, maka *password* yang tersimpan dalam *database* itu akan sama seperti aslinya, sehingga apabila ada yang berhasil menembus sistem keamanan web tersebut, maka dia dapat memilki semua user id dan *password* dari setiap user yang menggunakan aplikasi tersebut. Oleh karena itu diterapkanlah sistem *hash* kata sandi dengan algoritma MD5 pada aplikasi Simple-O ini, sehingga semua *password* yang disimpan pada *database* sistem akan dalam bentuk acak yang akan mempersulit seseorang untuk dapat memecahkan kata sandi tersebut.

Ketika seseorang mendaftarkan dirinya untuk pertama kali pada aplikasi Simple-O, maka data autentikasinya yang berupa *password*, akan tersimpan dalam bentuk *hash code* dalam *database*. Setelah dia terdaftar dan akan melakukan login, maka dia akan memasukkan *username* dan *password*-nya dalam bentuk yang asli (*plaintext*) di login page Simple-O. Setelah pengguna tersebut men-*submit username* dan *password*nya tersebut, maka kemudian sistem akan mencari *hash code* dari *password* yang di-submit pengguna, dan hasil *hash code* tersebut akan dibandingkan dengan nilai *hash code* yang sebelumnya telah tersimpan dalam *database* untuk memeriksa otentikasi dari pengguna tersebut. Apabila hasil perbandingan sama, maka pengguna itu memiliki hak akses pada aplikasi Simple-O. Gambar 2.4 menunjukkan alur proses daftar dan login yang terdapat di dalam aplikasi Simple-O.



Gambar 2.4 Proses login Simple-O

d. Lupa kata sandi dan ubah kata sandi

Lupa kata sandi merupakan hal yang terkadang terjadi pada seorang user. Oleh karena itu diterapkanlah fasilitas lupa kata sandi dan ubah kata sandi pada aplikasi Simple-O.

e. Proses login dengan sistem terkunci

Berdasarkan fungsinya sebagai aplikasi untuk melakukan ujian esai, maka pada aplikasi Simple-O tidak boleh ada satu buah user id yang diakses pada waktu bersamaan dari dua tempat yang berbeda. Untuk menghindari kondisi ini, maka diterapkan algoritma untuk sistem terkunci, sehingga satu buah user id hanya bisa diakses apabila satu sesi telah selesai, dan kemudian dapat diakses kembali pada sesi yang lain.

f. CAPTCHA pada proses login

CAPTCHA dapat dikatakan sebagai suatu tantangan. Tantangan ini diberikan oleh *server* kepada siapa saja yang akan melakukan akses ke *server* itu. Tujuannya adalah untuk membedakan apakah yang melakukan akses ke *server* itu adalah manusia atau komputer. Pembedaan ini ditujukan untuk menghindari serangan terhadap sistem melalui celah autentikasi sistem, sehingga dengan penerapan CAPTCHA serangan semacam *brute force attack* (mencoba segala kemungkinan *password* atau id) dapat diminimalisir.

Sistem keamanan ini telah berhasil diimplementasikan dengan baik pada aplikasi Simple-O, namun masih perlu dianalisa kembali efektifitas dari sistem keamanan ini dan masih perlu dikembangkan lagi kedepannya.

2.6 Tools

2.6.1 Ubuntu 11.10 – Oneiric Ocelot [16]

Oneiric Ocelot adalah *name code* dari Ubuntu 11.10 yang dirilis pada 13 Oktober 2011. Ocelot Oneiric termasuk rilis terbaru dari semua aplikasi utama Ubuntu: *desktop, server, cloud*, Kubuntu, Xubuntu, Lubuntu, Edubuntu, Mythbuntu, dan Ubuntu Studio. Rilis ini memberikan Unity experience secara penuh, bahkan tanpa dukungan akselerasi hardware 3D, mempromosikan Unity 2D ke *shell fallback primer*. LightDM dikembangkan menjadi aplikasi login yang digunakan pada Ubuntu, Edubuntu, Xubuntu, Mythbuntu, dan Ubuntu Studio. Kubuntu menampilkan *Platform* KDE, Workspace Plasma, dan Aplikasi lain (termasuk Software Centre Muon) dengan tampilan dan sistem yang terbaru.

2.6.2 LAMP

LAMP adalah aplikasi yang merupakan kombinasi dari empat buah aplikasi open source, yaitu Linux, Apache, MySql, dan aplikasi *script*ing page yang dapat mengeksekusi bahasa pemrograman PHP, Perl, dan Python. Aplikasi ini biasa digunakan untuk membuat suatu jaringan *server*.

Apache merupakan suatu aplikasi web *server* yang biasa digunakan. Aplikasi ini digunakan untuk meng-*hosting* web untuk kemudian dapat diakses baik secara lokal ataupun *online*. Sedangkan MySql adalah suatu aplikasi yang bekerja dari sisi *back-end user*, digunakan untuk melakukan manajemen *database* dari suatu aplikasi yang berjalan diatasnya, termasuk web. Web yang dihosting menggunakan Apache, akan dihubungkan dengan *database*-nya yang diaktifkan di MySql.

2.6.3 Hashcat [17]

Hashcat adalah suatu aplikasi yang digunakan untuk melakukan recovery *password* yang telah mengalami perubahan bentuk karena *hash*. *Hash*cat merupakan salah satu aplikasi *password recovery* berbasis CPU tercepat saat ini. Meskipun tidak secepat versi GPU base-nya, seperti: oclHashcat, oclHashcat-plus, dan oclHashcat-lite, namun kinerja aplikasi CPU base ini sudah cukup baik bahkan untuk memgolah data dengan ukuran besar.

Software hashcat family adalah seperangkat aplikasi profesional yang disediakan tanpa biaya untuk dapat digunakan oleh siapa saja. Hashcat ditujukan untuk dapat digunakan secara legal sebagai alat untuk memulihkan string plain text untuk berbagai metode *hash* yang digunakan, yaitu:

- MD5 (dan variasinya)
- SHA1 (dan variasinya)
- MySQL
- SSHA-1
- MD4
- NTLM
- Domain Cach
- MSSQL
- SHA256
- SHA-512
- Oracle 11G
- DES(Unix)

BAB III
PERANCANGAN SISTEM
SOFTWARE DAN HARDWARE

Perbandingan hasil penerapan dua algoritma yang berbeda dalam suatu sistem, bisa didapat dengan dilakukannya suatu pengujian dalam penerapan kedua algoritma tersebut. Pengujian itu harus dilakukan dengan metode dan dalam suatu lingkungan pengujian yang tepat, sehingga dapat menghasilkan data yang valid.

Untuk itu dibuatlah suatu konfigurasi sistem untuk melakukan ujicoba terhadap penerapan algoritma MD5 dan SHA-1 pada aplikasi Simple-O. Pada bab ini akan dijelaskan mengenai spesifikasi hardware dan software yang digunakan untuk membentuk sistem ujicoba tersebut. Pada bab ini juga akan dijelaskan mengenai bagaimana konfigurasi dari sistem ujicoba serta skenario yang digunakan dalam proses ujicoba.

3.1. Spesifikasi Hardware

Pada sistem ujicoba ini digunakan dua buah PC desktop yang berfungsi sebagai *server* dan *client*. Kedua PC tersebut terhubung dalam suatu jaringan lokal. Berikut ini spesifikasi dari kedua PC tersebut :

- PC *Server*

Motherboards : MS-6712, vendor MSI, version 1.0

Processor : AMD Athlon, 3GHz, 32 bits, clock 100Mhz, vendor Hynix Semiconductor. L1 cache 1Mb, L2 cache 2Mb.

Memory : DDR1, PC2100, 1Gb

Harddrive : ATA Disk, Maxtor 6E040L0, Maxtor, 40Gb

Graphic : NV17 (GeForce4 MX440), nVidia Corporation, 64Mb

Network : Ethernet interface, RTL8139 Ethernet, D-Link Sistem Inc, capacity 100Mbit/s

- PC *Client*

Motherboards : MS-6712, vendor MSI, version 1.0

Processor : AMD Athlon, 3GHz, 32 bits, clock 100Mhz, vendor Hynix Semiconductor. L1 cache 1Mb, L2 cache 2Mb.

Memory : DDR1, PC2100, 1Gb

Harddrive : ATA Disk, ST340014A, Seagate, 40Gb

Graphic : NV17 (GeForce4 MX440), nVidia Corporation, 64Mb

Network : Ethernet interface, 3c905C-TX/TX-M (Tornado), 3Com Corporation, capacity 100Mbit/s

Selain kedua PC tersebut, digunakan juga satu buah PC *standalone* lain yang tidak terhubung dalam jaringan. PC ini digunakan untuk melakukan *brute force testing* terhadap *ciphertext* dari algoritma MD5 dan SHA-1 sehingga bisa menemukan *plaintext* asli dari *ciphertext* tersebut. Berikut ini spesifikasi dari PC tersebut :

- PC Standalone

Motherboards : P5KPLAMSE, vendor Intel, version 1.0

Processor : Intel® Core™2 Duo E7400, 2.8GHz.

Memory : DDR3, Kingston, 2Gb

Harddrive : SATA Disk, Seagate, 250Gb

3.2. Spesifikasi Software

Pada perancangan sistem ujicoba ini, ada beberapa jenis software yang digunakan, yaitu *operating system* dan aplikasi pendukung lain. *Operating system* dan aplikasi pendukung ini digunakan baik untuk *platform* dasar sistem operasi dari PC *server* dan *client*, dan juga untuk membentuk suatu jaringan *local server* dalam sistem ujicoba ini. Selain itu digunakan juga suatu *script* PHP khusus yang digunakan untuk pengujian Berikut ini adalah penjelasan dari beberapa software utama dan *script* PHP yang digunakan dalam perancangan sistem ujicoba ini.

a. Ubuntu 11.10 – Oneiric Ocelot

Pada perancangan sistem ujicoba ini, pada kedua PC *server* dan *client*, akan digunakan satu operating sistem yang sama, yaitu Ubuntu 11.10 – Oneiric Ocelot. Ubuntu 11.10 adalah versi terbaru dari Ubuntu yang dirilis pada 13 Oktober 2011. OS ini dipilih untuk digunakan pada PC *server* dan *client* karena Ubuntu, yang merupakan OS keluaran Linux, lebih mudah untuk dikustomisasi, sehingga bisa digunakan baik sebagai *server* maupun *client*. Selain itu karena jaringan yang akan digunakan dalam sistem ujicoba ini merupakan jarigan lokal yang sederhana, maka penggunaan OS yang memang untuk *server* (ex. Debian) tidaklah menjadi keharusan.

b. LAMP

Seperti yang telah dijelaskan pada bab sebelumnya, LAMP adalah aplikasi yang merupakan kombinasi dari empat buah aplikasi open source, yaitu Linux, Apache, MySql, dan aplikasi *script*ing page yang dapat mengeksekusi bahasa pemrograman PHP, Perl, dan Python. Aplikasi-aplikasi itulah yang akan digunakan untuk membentuk jaringan *local server* pada sistem ujicoba ini.

Apache merupakan suatu aplikasi *web server* yang biasa digunakan dalam berbagai OS ber-*platform* Linux. Sesuai dengan fungsinya tersebut, aplikasi ini digunakan sebagai *web server* pada sistem ujicoba ini, di-install pada PC *server*, dan digunakan untuk meng-*hosting* web Simple-O menggunakan *local network*. Apache yang digunakan dalam perancangan sistem ini adalah Apache 2.0 Handler (Apache/2.2.20).

MySql adalah suatu aplikasi yang bekerja dari sisi *back-end user*, digunakan untuk melakukan manajemen *database* dari suatu aplikasi yang berjalan diatasnya, termasuk web. Pada perancangan sistem ujicoba ini, *database* dari aplikasi Simple-O diaktifkan menggunakan aplikasi ini. Web yang dihosting menggunakan Apache kemudian akan dihubungkan dengan *database*nya yang diaktifkan di MySql.

Aplikasi LAMP merupakan aplikasi bekerja dinamis dalam hubungannya dengan berbagai *script*ing language yang ada, termasuk dengan PHP. Web Simple-O ini dikembangkang dengan menggunakan bahasa pemrograman PHP. Oleh karena itu penggunaan LAMP untuk menjalankan web ini merupakan pilihan yang tepat karena kompatibel dengan bahasa pemrograman yang digunakan di Simple-O.

c. *Hash*cat

*Hash*cat adalah suatu aplikasi yang digunakan untuk melakukan recovery *password* yang telah mengalami perubahan bentuk karena *hash*. Pada sistem ujicoba ini, software ini digunakan untuk menemukan *plaintext* dari *password* yang telah di-*hash* menggunakan algoritma MD5 dan SHA-1. Tujuannya adalah untuk menghitung waktu yang dibutuhkan untuk dapat menemukan suatu *plaintext* hasil *hash*dari suatu *ciphertext*. Suatu algoritma *hash* yang baik, akan membutuhkan waktu lebih lama saat akan di-*generate*. Dengan menggunakan software ini, maka waktu untuk men-*generate password* hasil *hash* MD5 dan SHA-1 dapat dihitung, kemudian membandingkan hasilnya untuk melihat mana diantara kedua algoritma ini yang lebih lama saat di-*generate*. Pada sistem ujicoba ini, digunakanlah aplikasi *Hash*cat versi 0.2.433.

d. LINUX Terminal Command – `Top`

Linux memiliki berbagai macam *terminal command*, salah satunya adalah `top`. `Top` merupukan terminal command Linux yang berfungsi untuk memantau seluruh aktifitas pada processor. Dengan menggunakan *command* ini, maka kita akan dapat memantau *uptime system* secara *real time*, proses-proses apa saja yang sedang berjalan, *CPU usage*, *memory usage, dan swap*.

e. *Microtime Script* pada PHP

Pada *script* PHP Simple-O, yaitu pada bagian cekuser.php, terdapat suatu *microtime script* yang digunakan untuk menghitung *processing time* saat dilakukan proses login. Setiap kali proses login dilakukan, maka waktu yang

diperlukan untuk memprosesnya akan secara otomatis dihitung dan disimpan di dalam *database*. Berikut ini adalah *script* yang terdapat pada cekuser.php :

```
<?php
        $mtime = microtime();
        $mtime = explode(" ",$mtime);
        $mtime = $mtime[1] + $mtime[0];
        $starttime = $mtime;
.
.
.
                $mtime = microtime();
                $mtime = explode(" ",$mtime);
                $mtime = $mtime[1] + $mtime[0];
                $endtime = $mtime;
                $totaltime = ($endtime - $starttime);

                $sqltime= "insert into delay_login (userid,delay)
                values ('$_POST[user]','$totaltime')";
                $rstime= $db->Execute($sqltime);
                header("Location: admin.php");
                }
.
.
.
?>
```

Gambar 3.1 *Script microtime* untuk proses ujicoba

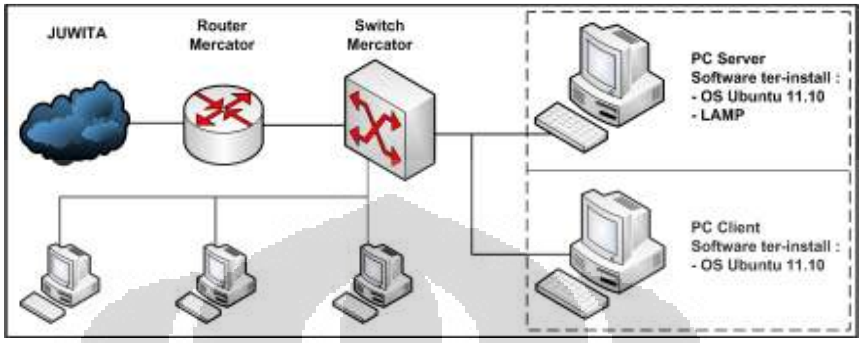
Bagian tulisan yang ditulis miring adalah *script microtime* untuk menghitung *processing time*. Sedangkan yang tidak ditulis miring merupakan perintah untuk menuliskan waktu *processing time* tersebut pada *database* delay_login.

3.3. Desain Sistem dan Skenario Ujicoba

a. Desain Sistem

Sistem ujicoba ini terdiri dari 2 buah PC, masing-masing berfungsi sebagai *server* dan *client*. *Server* ini terhubung dalam satu jaringan lokal di Mercator Office UI. Jaringan ini tersusun atas router yang terhubung dengan jaringan *public* UI, switch, dan komputer-komputer dalam jaringan. Dari banyak komputer yang terhubung dalam jaringan Mercator Office tersebut, digunakanlah dua buah PC

untuk dapat dijadikan sebagai lokal *server* dan *client*, yang terlibat dalam sistem ujicoba ini. Berikut ini desain sistem dari sistem ujicoba ini :



Gambar 3.2 Desain Sistem Ujicoba

Sedangkan satu buah PC lain digunakan untuk melakukan *brute force attack* terhadap *ciphertext* dari algoritma MD5 dan SHA-1, yang di dalamnya ter-*install* aplikasi *Hash*cat.

b. Skenario Ujicoba

Dalam sistem ujicoba ini, terdapat tiga buah skenario ujicoba yang akan digunakan untuk menguji kekuatan algoritma MD5 dan SHA-1 yang diterapkan pada aplikasi Simple-O, serta pengaruhnya pada kinerja aplikasi ini. Berikut ini penjelasan dari ketiga skenario tersebut :

1. *Brute force hash code*

Saat seorang hacker berada dalam *database* Simple-O, maka dia bisa mendapatkan seluruh data user pengguna aplikasi ini. Namun karena semua data *password* pengguna telah di *hash*, maka hacker tidak bisa menggunakan data tersebut, kecuali *password* yang di-*hash* tersebut di *generate* terlebih dahulu. Ujicoba ini bertujuan untuk menemukan *plaintext* asli dari *ciphertext* algoritma *hash* tersebut. Ujicoba akan dilakukan dengan menerapkan algoritma MD5 dan SHA-1, dan dilakukan dengan menggunakan tools *Hash*cat. Semakin baik algortima yang digunakan, maka akan semakin lama waktu yang dibutuhkan untuk mem-*brute force ciphertext* tersebut. *Brute force* akan dilakukan untuk *password* dengan panjang 6 sampai 9 karakter.

2. Mengukur *processing time* saat login

Lama proses autentikasi pada Simple-O terkait dengan penerapan algoritma *hash* pada sistem ini, karena saat melakukan autentikasi, *password* yang di-input oleh user akan di *hash* menggunakan algoritma yang diterapkan, untuk kemudian dibandingkan dengan hasil *hash* yang tersimpan dalam *database*. Oleh karena itu, apabila terdapat dua algoritma berbeda yang diterapkan dalam aplikasi Simple-O ini, maka *processing time* saat seorang user login ke dalam sistem juga akan mengalami perbedaan. Skenario ini bertujuan mengukur perubahan *processing time* saat proses autentikasi user, ketika diterapkan algoritma SHA-1 dibandingkan dengan saat diterapkan algoritma MD5. Pengukuran dilakukan dalam 4 kombinasi algoritma, yaitu MD5, SHA-1, MD5 dengan *salt*, dan SHA-1 dengan *salt*. Pengukurannya akan memanfaatkan fungsi perhitungan waktu *microtime* yang telah diimplementasikan pada aplikasi Simple-O saat melakukan proses cekuser.

3. Mengukur *CPU usage server* saat login

Pada skenario ini, akan dilakukan pengukuran atas penggunaan *resource* CPU pada *server* saat melakukan proses autentikasi. Dengan penggunaan algoritma yang berbeda pada proses autentikasi, maka alokasi CPU yang digunakan juga akan berbeda. Pengukuran dilakukan dalam 4 variasi penerapan algoritma, yaitu MD5, SHA-1, MD5 dengan *salt*, dan SHA-1 dengan *salt*. Pengukuran dilakukan secara menyeluruh, dalam arti tidak hanya memantau penggunaan CPU untuk autentikasi sistem saja, namun kondisi *CPU usage* yang di dalamnya juga terakumulasi dengan aplikasi lain yang berjalan pada *server*. Tujuannya adalah agar data yang didapatkan dapat sesuai dengan kondisi *server* yang sebenarnya, meski dilakukan dalam jaringan minimum *local server*.

Pada skenario pengukuran processing time dan CPU usage, pengunaan algoritma hash dalam program Simple-O harus diubah sesuai dengan variasi yang

akan diuji saat itu. Berikut ini *pseudocode* yang menampilkan perubahan pada program PHP Simple-O untuk tiap variasi algoritma yang diuji:

- MD5

```
$pass = $_POST['pass'];
$password = md5($pass);
```

- MD5+salt

```
$pass = $_POST['pass'];
$pengacak = "awdrgyjilpzdcvgnjm";
$password = md5($pengacak . md5($pass) . $pengacak );
```

- SHA-1

```
$pass = $_POST['pass'];
$password = sha1($pass);
```

- SHA-1+salt

```
$pass = $_POST['pass'];
$pengacak = "awdrgyjilpzdcvgnjm";
$password = sha1($pengacak . sha1($pass) . $pengacak );
```

Sedangkang fungsi pengacak atau *salt* yang digunakan pada proses pengujian, bersesuaian dengan persamaan (2) dibawah ini:

```
$salt = "awdrgyjilpzdcvgnjm";
$password = algoritma($salt . algoritma($pass) . $salt );
```
.....(2)

BAB IV
PENGUJIAN DAN ANALISIS SISTEM

Seperti yang telah dijelaskan pada bagian sebelumnya, pengujian akan dilakukan dalam tiga skenario utama yang berbeda, yaitu *brute force* terhadap *hash ciphertext*, pengukuran *processing time* saat login, dan pengukuran *CPU usage* saat login. Proses *brute force* dilakukan dengan menggunakan aplikasi *Hash*cat dengan tujuan untuk mendapatkan data berupa waktu yang dibutuhkan serta estimasi waktu yang dibutuhkan untuk berhasil menemukan *plaintext* asli dari suatu *hash ciphertext* yang diberikan, baik *ciphertext* yang didapat dari algoritma MD5 maupun SHA-1.

Untuk pengukuran *processing time* dilakukan dengan memanfaatkan string *microtime* yang terdapat pada proses cekuser Simple-O. Sedangkan pengukuran *CPU usage* dilakukan dengan memanfaatkan *command* top pada linux. Tujuan kedua pengukuran ini adalah untuk melihat hasil penerapan kedua algoritma MD5 dan SHA-1 terhadap kinerja Simple-O saat proses autentikasi. Proses pengetasan kedua skenario ini dilakukan dalam 4 kondisi penerapan algoritma yang berbeda untuk masing-masing skenario, yaitu penerapan algoritma MD5, SHA-1, MD5 dengan *salt*, dan SHA-1 dengan *salt*.

4.1 Skenario *Brute force Hash code*

Skenario ini dijalankan dengan melakukan proses *brute force attack* terhadap *hash ciphertext* dari *password* dengan panjang enam hingga sembilan karakter. Tujuannya untuk mengukur berapa lama waktu yang dibutuhkan untuk dapat menemukan *plaintext* dari suatu *hash ciphertext* saat dilakukan *brute force attack* terhadap *hash code* yang terdapat pada *database* Simple-O. Pengujian ini dilakukan dengan menggunakan software *Hash*cat yang memanfaatkan *processor* dalam proses pengujian. Pada proses pengujian digunakanlah suatu *character set (charset)* yang ditentukan sendiri kombinasinya. *Charset* ini digunakan sebagai referensi untuk membuat kombinasi huruf saat melakukan *brute force*. Kombinasi *charset* yang digunakan pada pengujian ini adalah semua huruf *alphabet uppercase dan lowercase* (ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghij-

klmnopqrstuvwxyz). Pengujian dilakukan pada penerapan algoritma MD5 dan SHA-1 dan menggunakan komputer *standalone* dengan *processor* Intel Core2Duo.

Ada dua macam hasil pengujian dari skenario ini, yaitu berupa waktu hasil pengujian dan waktu estimasi dari proses *brute force*. Waktu hasil pengujian adalah waktu yang didapat dalam proses *brute force* hingga didapat *plaintext* asli dari *ciphertext* yang di-*brute force* tersebut. Waktu hasil pengujian ini didapat pada proses *brute force* untuk *plaintext* dengan panjang 6 karakter, baik yang kemudian di *hash* menggunakan algoritma MD5 maupun SHA-1. Pengujian untuk mendapat waktu hasil pengujian dari *brute force* ini hanya dilakukan untuk *plaintext* dengan panjang 6 karakter karena waktu estimasi untuk *brute force* dengan panjang 6 karakter tidaklah terlalu lama, yaitu kurang lebih selama 30 menit. Oleh karena itu proses pengujian dilakukan hingga didapatkan *plaintext* asli dari *hash ciphertext* yang di-*brute force*.

Untuk *plaintext* dengan panjang lebih dari enam karakter, maka hanya diambil waktu estimasinya saja. Hal ini dikarenakan waktu estimasi proses *brute force* untuk *plaintext* yang lebih dari enam karakter membutuhkan waktu yang sangat lama. Untuk *plaintext* dengan tujuh karakter saja estimasi waktu *brute force*-nya bisa mencapai hingga lebih dari 20 jam karena kombinasi kata yang dapat terbentuk menjadi semakin banyak seiring bertambahnya jumlah karakter *password*. Itu pun hanya menggunakan *charset alphabet uppercase* dan *lowercase*. Oleh karena itu diputuskan bahwa untuk *plaintext* dengan panjang lebih dari enam karakter hanya akan diambil waktu estimasinya saja.

Berikut ini kombinasi plainext dari password yang digunakan pada pengujian:

- 6 karakter : ahmads
  - MD5 : 6745d750c41fec903e765b958f63748f
  - SHA-1 : 8f03c697c9ce1621e5c9386cef876677f3665d52
- 7 karakter : ashaugi
  - MD5 : 9b559c59f8a34592401c3793de31034d
  - SHA-1 : 9afb8e7a21d40b4dee4b6ac8388bca63128e6df2
- 8 karakter : farchank

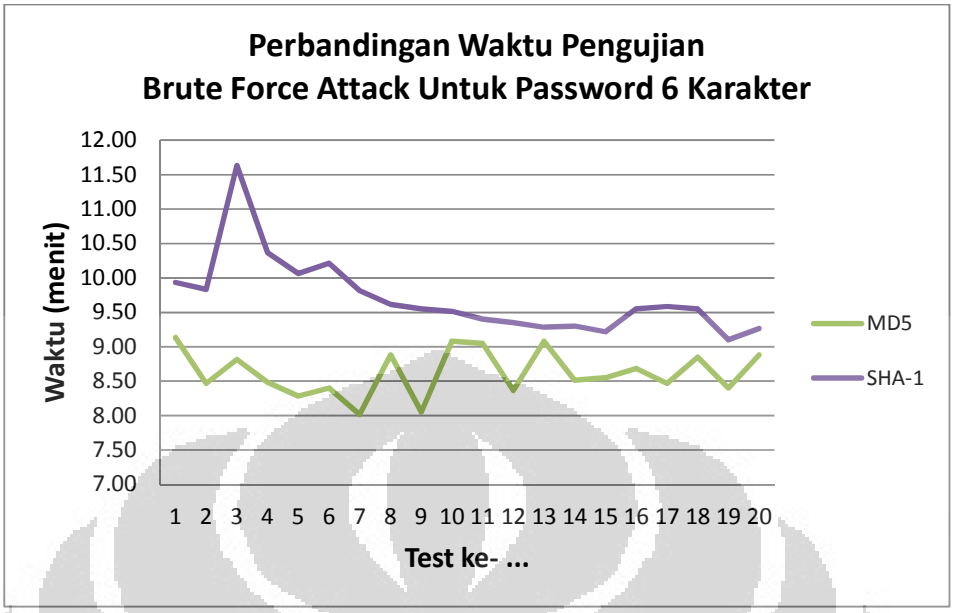
- MD5 : 093546f46590508a4171d3a63209065d
- SHA-1 : 80800dda37c54f359c48c67f86023c19a6e7a7b4

- 9 karakter : gregorius
  - MD5 : 655cfb5832d253a54677b9a9e2388373
  - SHA-1 : a8882d0448871f18abc918f375727dbd1cf0a59b

a. Hasil Pengukuran

Tabel 4.1 Perbandingan Waktu *Brute force Password* 6 Karakter

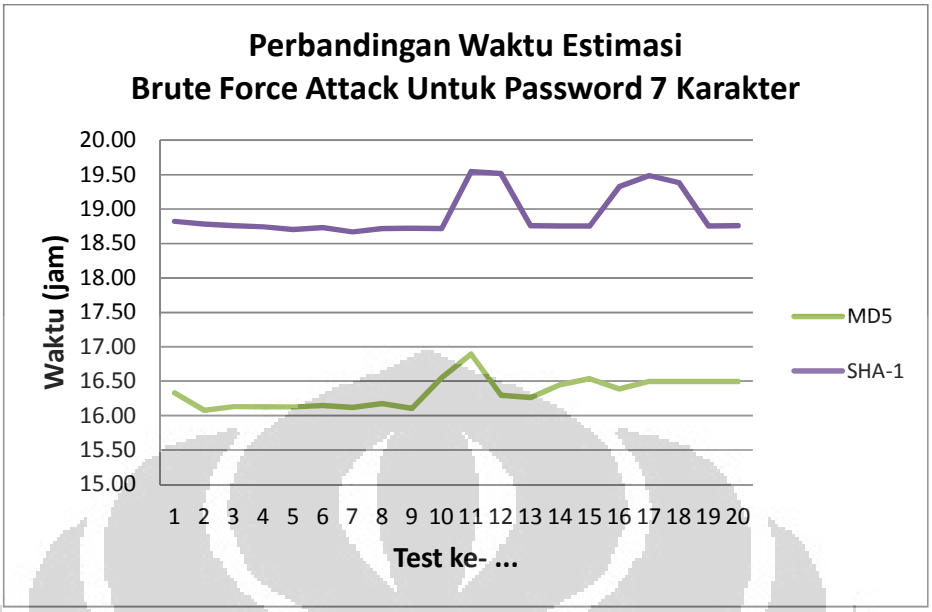
Testing ke-...	Waktu Pengujian *Brute force attack*			
	MD5		SHA-1	
	6 Karakter	6 Karakter (Dalam Menit)	6 Karakter	6 Karakter (Dalam Menit)
1	00:00:09:08	9.13	00:00:09:56	9.93
2	00:00:08:28	8.47	00:00:09:50	9.83
3	00:00:08:49	8.82	00:00:11:38	11.63
4	00:00:08:29	8.48	00:00:10:22	10.37
5	00:00:08:17	8.28	00:00:10:04	10.07
6	00:00:08:24	8.40	00:00:10:13	10.22
7	00:00:08:01	8.02	00:00:09:49	9.82
8	00:00:08:53	8.88	00:00:09:37	9.62
9	00:00:08:03	8.05	00:00:09:33	9.55
10	00:00:09:05	9.08	00:00:09:31	9.52
11	00:00:09:03	9.05	00:00:09:24	9.40
12	00:00:08:22	8.37	00:00:09:21	9.35
13	00:00:09:05	9.08	00:00:09:17	9.28
14	00:00:08:31	8.52	00:00:09:18	9.30
15	00:00:08:33	8.55	00:00:09:13	9.22
16	00:00:08:41	8.68	00:00:09:33	9.55
17	00:00:08:28	8.47	00:00:09:35	9.58
18	00:00:08:51	8.85	00:00:09:33	9.55
19	00:00:08:24	8.40	00:00:09:06	9.10
20	00:00:08:53	8.88	00;00:09:16	9.27
Rata-rata	**0:0:8:38.7**	**8.65**	**0:0:9:42.45**	**9.71**



Gambar 4.1 Perbandingan waktu pengujian *brute force attack* untuk *password* 6 karakter

Tabel 4.2 Perbandingan Waktu Estimasi *Brute force Password* 7 Karakter

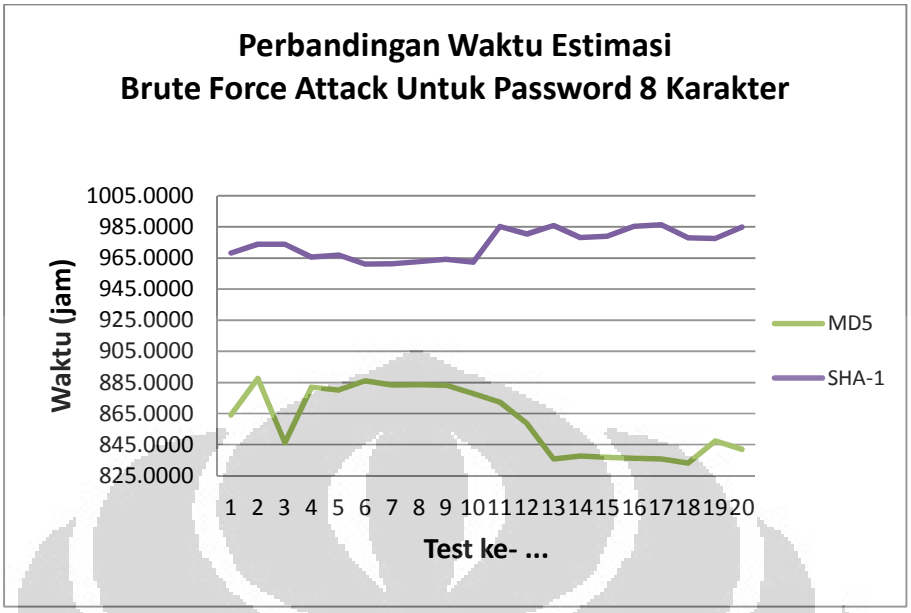
Testing ke-...	Waktu Estimasi *Brute force attack*			
	MD5		SHA-1	
	7 Karakter	7 Karakter (Dalam Jam)	7 Karakter	7 Karakter (Dalam Jam)
1	00:16:19:52	16.3311	00:18:49:16	18.8211
2	00:16:04:46	16.0794	00:18:47:04	18.7844
3	00:16:07:51	16.1308	00:18:45:45	18.7625
4	00:16:07:56	16.1322	00:18:44:35	18.7431
5	00:16:07:39	16.1275	00:18:42:10	18.7028
6	00:16:08:55	16.1486	00:18:43:46	18.7294
7	00:16:07:03	16.1175	00:18:40:10	18.6694
8	00:16:10:30	16.1750	00:18:42:56	18.7156
9	00:16:06:22	16.1061	00:18:43:18	18.7217
10	00:16:32:58	16.5494	00:18:42:55	18.7153
11	00:16:53:47	16.8964	00:19:32:55	19.5486
12	00:16:17:49	16.2969	00:19:31:05	19.5181
13	00:16:15:47	16.2631	00:18:45:33	18.7592
14	00:16:26:53	16.4481	00:18:45:29	18.7581
15	00:16:32:18	16.5383	00:18:45:22	18.7561
16	00:16:23:20	16.3889	00:19:19:39	19.3275
17	00:16:29:49	16.4969	00:19:29:09	19.4858
18	00:16:29:49	16.4969	00:19:23:08	19.3856
19	00:16:29:49	16.4969	00:18:45:15	18.7542
20	00:16:29:49	16.4969	00:18:45:42	18.7617
Rata-rata	**0:16:20:9.1**	**16.3359**	**0:18:55:15.6**	**18.9210**



Gambar 4.2 Perbandingan waktu estimasi *brute force attack* untuk *password* 7 karakter

Tabel 4.3 Perbandingan Waktu Estimasi *Brute force Password* 8 Karakter

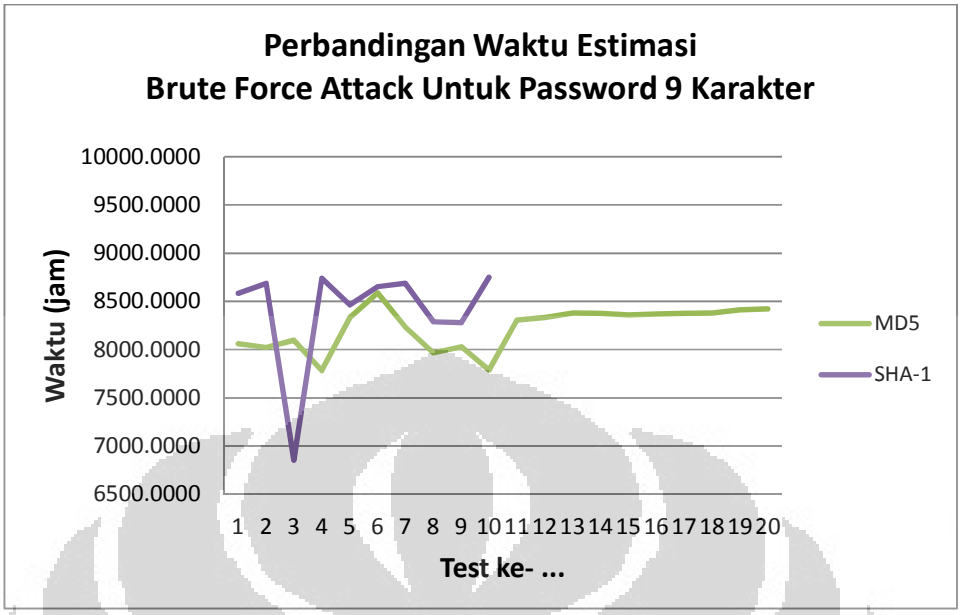
Testing ke-...	Waktu Estimasi *Brute force attack*			
	MD5		SHA-1	
	8 Karakter	8 Karakter (Dalam Jam)	8 Karakter	8 Karakter (Dalam Jam)
1	36:00:02:01	864.0336	40:08:10:32	968.1756
2	36:23:40:14	887.6706	40:13:46:59	973.7831
3	35:06:21:01	846.3503	40:13:49:31	973.8253
4	36:18:03:12	882.0533	40:05:37:18	965.6217
5	36:16:00:37	880.0103	40:06:56:04	966.9344
6	36:22:04:28	886.0744	40:01:04:07	961.0686
7	36:19:26:47	883.4464	40:01:20:21	961.3392
8	36:19:29:05	883.4847	40:02:45:06	962.7517
9	36:19:10:32	883.1756	40:04:08:22	964.1394
10	36:14:02:37	878.0436	40:02:20:22	962.3394
11	36:08:25:19	872.4219	41:01:15:41	985.2614
12	35:18:40:14	858.6706	40:20:23:42	980.3950
13	34:19:43:27	835.7242	41:01:48:48	985.8133
14	34:21:43:30	837.7250	40:18:13:07	978.2186
15	34:20:43:59	836.7331	40:19:04:41	979.0781
16	34:20:16:44	836.2789	41:01:33:18	985.5550
17	34:19:50:56	835.8489	41:02:24:42	986.4117
18	34:17:01:19	833.0219	40:17:55:53	977.9314
19	35:07:28:33	847.4758	40:17:30:33	977.5092
20	35:01:59:49	841.9969	41:00:58:27	984.9742
Rata-rata	**35:20:30:43.2**	**860.5120**	**40:14:3:22.7**	**974.0563**



Gambar 4.3 Perbandingan waktu estimasi *brute force attack* untuk *password* 8 karakter

Tabel 4.4 Perbandingan Waktu Estimasi *Brute force Password* 9 Karakter

Testing ke-...	Waktu Estimasi *Brute force attack*			
	MD5		SHA-1	
	9 Karakter	9 Karakter (Dalam Jam)	9 Karakter	9 Karakter (Dalam Jam)
1	335:18:25:50	8058.4306	357:17:25:05	8585.4181
2	334:06:09:40	8022.1611	361:22:03:09	8686.0525
3	337:09:08:50	8097.1472	285:12:54:06	6852.9017
4	324:04:53:43	7780.8953	364:02:44:38	8738.7439
5	347:02:06:11	8330.1031	352:15:15:03	8463.2508
6	357:18:51:50	8586.8639	360:11:09:16	8651.1544
7	342:22:57:31	8230.9586	361:21:34:37	8685.5769
8	331:23:16:19	7967.2719	345:07:04:59	8287.0831
9	334:13:37:59	8029.6331	344:23:32:31	8279.5419
10	324:11:35:18	7787.5883	364:12:18:59	8748.3164
11	346:02:22:33	8306.3758	-	-
12	347:02:02:10	8330.0361	-	-
13	349:03:11:21	8379.1892	-	-
14	348:22:23:39	8374.3942	-	-
15	348:08:44:47	8360.7464	-	-
16	348:16:13:56	8368.2322	-	-
17	348:22:38:00	8374.6333	-	-
18	349:03:23:28	8379.3911	-	-
19	350:12:51:43	8412.8619	-	-
20	350:22:02:49	8422.0469	-	-
Rata-rata	**342:21:56:52.85**	**8229.9480**	**349:21:47:74.3**	**8397.8040**



Gambar 4.4 Perbandingan waktu estimasi *brute force attack* untuk *password* 9 karakter

Tabel 4.5 Rata-rata dan Selisih Waktu dan Estimasi *Brute force*

Panjang Karakter	Waktu Rata-rata		Selisih	Keterangan
	MD5	SHA-1		
6 Karakter	8.62	9.71	1.08	Dalam menit
7 Karakter	16.34	18.92	2.59	Dalam jam
8 Karakter	860.51	974.06	113.54	
9 Karakter	8229.95	8397.80	167.86	

b. Analisis

Pada pengujian skenario ini didapatkan lima buah tabel hasil pengujian dan empat buah grafik hasi pengujian. Tabel 4.1 berisikan waktu yang dibutuhkan untuk mem-*brute force* suatu *ciphertext* dari algoritma *hash* MD5 dan SHA-1 hingga didapatkan *plaintext*-nya yang berupa *password* sepanjang 6 karakter. Tabel 4.2, 4.3, dan 4.4, menampilkan waktu estimasi yang dibutuhkan untuk mem*brute force ciphertext* dari MD5 dan SHA-1 dengan panjang 7, 8, dan 9 karakter. Penghitungan waktu estimasi ini didasarkan pada kondisi dimana waktu yang dibutuhkan untuk mem-*brute force password* dengan panjang lebih dari 6 karakter sangatlah lama. Pengujian dilakukan tanpa menggunakan *salt* dan menggunakan kombinasi *charset*

alphabet uppercase dan lower case. Format waktu pada kolom ke-2 dan ke-4 pada Tabel 4.1 hingga 4.4 adalah *hari:jam:menit:detik*.

Tabel 4.5 merupakan tabel rata-rata dan selisih waktu dan estimasi *brute force.* Untuk password 6 karakter selisih waktu *brute force*-nya dalam satuan menit. Sedangkan untuk password 7, 8, dan 9 karakter selisih waktu *brute force*-nya dalam satuan jam.

Pada Gambar 4.1 ditampilkan grafik perbandingan waktu yang dibutuhkan untuk mem-*brute force* suatu *ciphertext* dari algoritma MD5 dan SHA-1 hingga didapat *plaintext* aslinya. Pada Gambar 4.1 ini, dapat terlihat bahwa SHA-1 membutuhkan waktu yang lebih panjang bila dibandingkan dengan MD5 saat dicoba untuk di-*generate* dengan cara *brute force attack*. Terlihat pada Tabel 4.1, batas atas waktu *brute force* untuk SHA-1 adalah 11.63 menit, dan batas bawahnya 9.1 menit. Sedangkan pada MD5 batas atasnya 9.13 menit, dan batas bawahnya 8.02 menit. Rata-rata waktu untuk SHA-1 adalah 9.71 menit, dan MD5 8.62 menit. Dari nilai batas atas, batas bawah, dan rata-rata dari masing-masing algoritma ini, dapat terlihat bahwa perbedaan waktu yang tidak terlalu jauh saat dilakukan *brute force attack* terhadap *password* sepanjang 6 karakter yang di *hash* menggunakan MD5 dan SHA-1.

Pada Gambar 4.2 ditampilkan grafik perbandingan waktu estimasi *brute force* antara penerapan algoritma MD5 dan SHA-1. Sama seperti sebelumnya, algoritma SHA-1 membutuhkan waktu yang lebih lama ketika dilakukan *brute force attack*. Namun tidak seperti sebelumnya, selisih rata-rata waktu yang dibutuhkan menjadi lebih lama, yaitu 2.59 jam. Begitu juga dengan Gambar 4.3 yang menampilkan grafik perbandingan waktu estimasi *brute force* untuk *password* dengan panjang 8 karakter, SHA-1 membutuhkan waktu lebih panjang, dan selisih waktu estimasi dengan MD5-pun semakin besar. Selisih rata-rata waktu estimasi *brute force* untuk *password* dengan panjang 8 karakter mencapai 4.73 hari.

Saat proses pengujian untuk mendapatkan estimasi waktu *brute force password* sepanjang 9 karakter, ditemukan suatu masalah. Pada dasarnya, pengujian untuk *password* dengan panjang 9 karakter ini memang cenderung

tidak mudah, karena nilai waktu estimasi yang sangat fluktuatif saat proses pengujian. Diperlukan jeda waktu tertentu sehingga sistem *Hash*cat telah betul-betul stabil hingga bisa menentukan waktu estimasi. Hanya saja masalah muncul saat pengujian terhadap *password* yang di *hash* dengan algoritma SHA-1 dilakukan, yaitu sistem tidak kunjung stabil saat dilakukan pengujian untuk menentukan waktu estimasi setelah dilakukannya testing kesepuluh. Nilai yang didapat pada testing kesebelas sangat fluktuatif dan tidak kunjung stabil. Oleh karena itu diputuskan untuk hanya menggunakan 10 data pada pengujian SHA-1 dengan panjang 9 karakter ini. Meski begitu, terlihat pada Gambar 4.4, dari 10 data yang didapat pada proses pengujian menunjukkan SHA-1 tetap membutuhkan waktu estimasi yang lebih panjang, kecuali pada testing ketiga yang membutuhkan waktu estimasi dengan kisaran yang sama pada kisaran waktu estimasi MD5.

Pada dasarnya, kecepatan waktu *brute force* sangat berkaitan dengan *resource* software dan hardware yang digunakan. Semakin handal software yang dipakai, dan semain tinggi spesifikasi hardware yang digunakan, maka waktu yang dibutuhkan juga akan semakin cepat. Software *Hash*cat yang digunakan pada pengujian ini termasuk dalam software yang cukup handal dan umum digunakan untuk men-*generate ciphertext* dari suatu algoritma *hash* tertentu. *Processor* Core2Duo yang digunakan pada sistem pengujian ini juga terbilang cukup baik, meski masih ada *processor* lain yang lebih baik. Selain itu penggunaan GPU untuk *brute force* juga akan jauh lebih baik. Namun begitu, dengan penggunaan *processor* Core2Duo ini, didapatkan beberapa keuntungan dan juga kelemahan pada proses pengujian.

Jika dilihat pada proses pengujian 6 karakter, waktu yang dibutuhkan tidaklah terlalu lama, meskipun hasil tetap menunjukkan bahwa SHA-1 membutuhkan waktu *brute force* lebih panjang. Selisih waktu yang tidak terlalu lama ini didukung oleh kinerja software dan hardware yang baik, sehingga selisih waktu *brute force* terlihat tidak begitu signifikan. Namun saat digunakan untuk mem-*brute force password* sepanjang 7 dan 8 karakter, kemampuan *processor* Core2Duo bisa dibilang tidak sebaik sebelumnya, karena waktu *brute force* yang menjadi semakin panjang. Namun begitu CPU

masih bisa memberi waktu estimasi yang cukup stabil pada proses pengujian. Pada pengujian waktu estimasi *password* sepanjang 7 dan 8 karakter inilah dapat selisih waktu estimasi dari proses *brute force* algoritma SHA-1 dan MD5. Selisih waktu estimasi ini menjadi parameter yang menunjukkan kerumitan dan kekuatan algoritma SHA-1 yang lebih baik bila dibandingkan dengan MD5.

Kelemahan dari penggunaan *processor* Core2Duo pada proses pengujian ini terlihat pada pengujian *password* dengan panjang 9 karakter. Seperti yang dijelaskan sebelumnya, proses pengambilan data estimasi waktu *brute force* untuk *password* sepanjang 9 karakter harus terhenti pada testing kesepuluh algoritma SHA-1. Hal ini menunjukkan bahwa dengan kombinasi kata yang dapat terbentuk menjadi semakin banyak, procesor yang digunakan menjadi semakin kesulitan saat harus menentukan waktu estimasinya. Namun begitu, dari 10 data yang didapat tetap menunjukkan bahwa SHA-1 yang memiliki waktu estimasi yang lebih tinggi dibandingkan dengan MD5. Maka dapat disimpulkan bahwa SHA-1 membutuhkan waktu *brute force* yang lebih panjang dibandingkan MD5, yang berarti bersesuaian dengan teori yang ada.

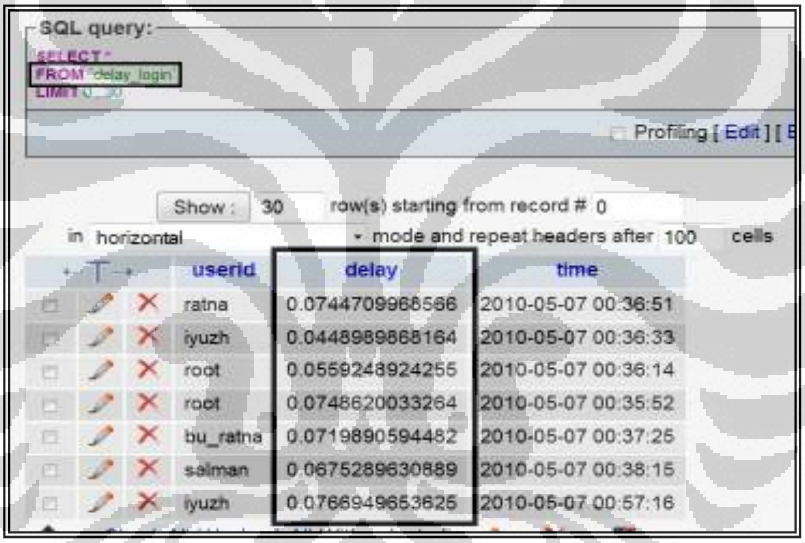
Satu hal yang harus tetap diperhatikan, proses *brute force* sendiri belum tentu sesuai dengan waktu estimasinya. Bisa saja waktu yang dibutuhkan lebih cepat atau bahkan lebih lama dari waktu estimasinya. Hanya saja, dengan mengukur waktu estimasi ini, maka dapat digunakan sebagai salah satu parameter untuk menunjukkan kekuatan dari masing-masing algoritma tersebut.

4.2 Skenario Pengukuran Processing Time

Skenario ini bertujuan untuk menghitung lama waktu dari proses cekuser pada aplikasi Simple-O saat dilakukan login. Perhitungan ini akan menunjukkan perbandingan hasil penerapan algoritma MD5 dan SHA-1 dalam pengaruhnya terhadap performa Simple-O.

Proses autentikasi Simple-O melibatkan 3 file php penyusun aplikasi ini, yaitu loginform.php, cekuser.php, dan admin.php. Loginform.php akan menampilkan halaman login dari aplikasi ini, dimana seorang user akan menginput *username* dan *password*-nya. Setelah user menekan submit, maka

proses pengecekan akan dilakukan oleh cekuser.php, dan kemudian akan masuk ke halaman admin.php apabila user tersebut terautentikasi. Pada bagian cekuser.php inilah terdapat keterlibatan algoritma *hash* untuk mengautentikasi user, dimana tiap *password* yang disubmit akan di-*hash* untuk kemudian dibandingkan dengan *password* yang tersimpan dalam *database* Simple-O. Pada *script* cekuser.php ini terdapat fungsi *microtime* yang berfungsi untuk menghitung lama proses yang terjadi dalam cekuser.php. Hasil perhitungan akan secara otomatis tersimpan dalam *database* delay_login pada aplikasi Simple-O. *Script microtime* inilah yang dimanfaatkan pada skenario ini untuk menghitung *processing time* pada saat proses cekuser.



Gambar 4.5 Waktu *processing time* (dalam kotak) di *database* delay_login pada aplikasi Simple-O

Pada skenario ini terdapat 4 macam kombinasi penerapan algoritma *hash*, yaitu MD5, SHA-1, MD5 dengan *salt*, dan SHA-1 dengan *salt*. Kombinasi ini diterapkan secara bergantian pada aplikasi Simple-O dengan cara memodifikasi 4 file php yang terdapat pada aplikasi Simple-O, yaitu cekuser.php, resetPass_save.php, changePass_update.php, dan changePass_update.save.php. Tiap file tersebut dimodifikasi pada bagian *hash password*-nya sesuai dengan penerapan algoritma yang akan diuji.

Pada tiap kombinasi dilakukan 60 kali proses login, dimana tiap proses login tersebut akan secara otomatis terhitung dan tersimpan lama waktu proses

cekusernya. Proses login dilakukan dalam 3 kombinasi *password*, yaitu *password* dengan panjang 8, 9, dan 10 karakter, masing-masing dilakukan 20 kali proses login. Setelah dilakukan 20 kali proses login untuk tiap panjang karakter, maka data di-*export* dari *database* Simple-O untuk kemudian diolah.

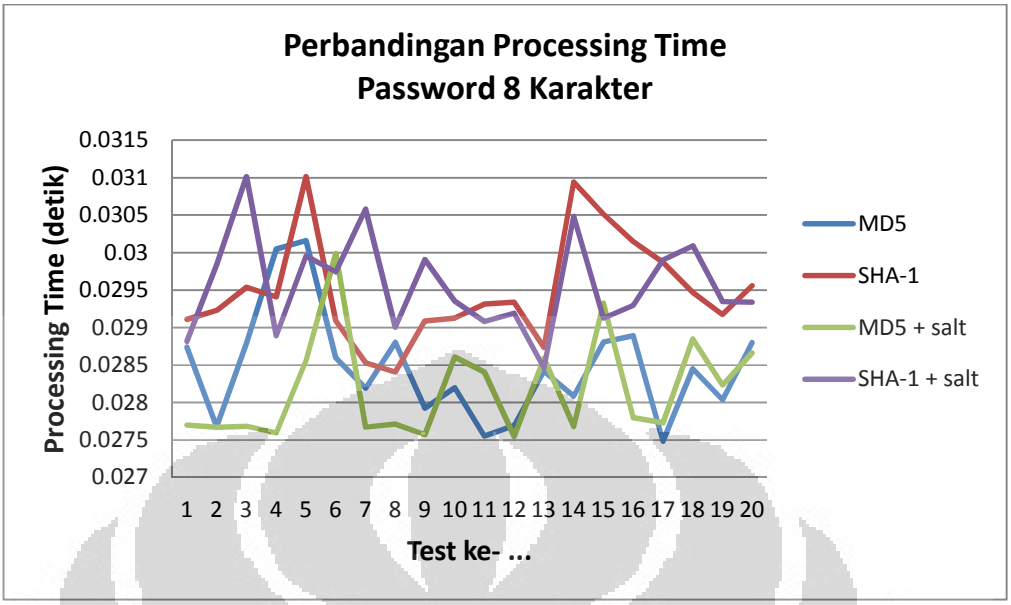
Berikut ini adalah *plaintext password* yang digunakan pada skenario ini:

- 8 karakter : ahmad100
- 9 karakter : shaugi100
- 10 karakter : farchan100

a. Hasil Pengukuran

Tabel 4.6 Perbandingan *Processing time* untuk *password* 8 karakter

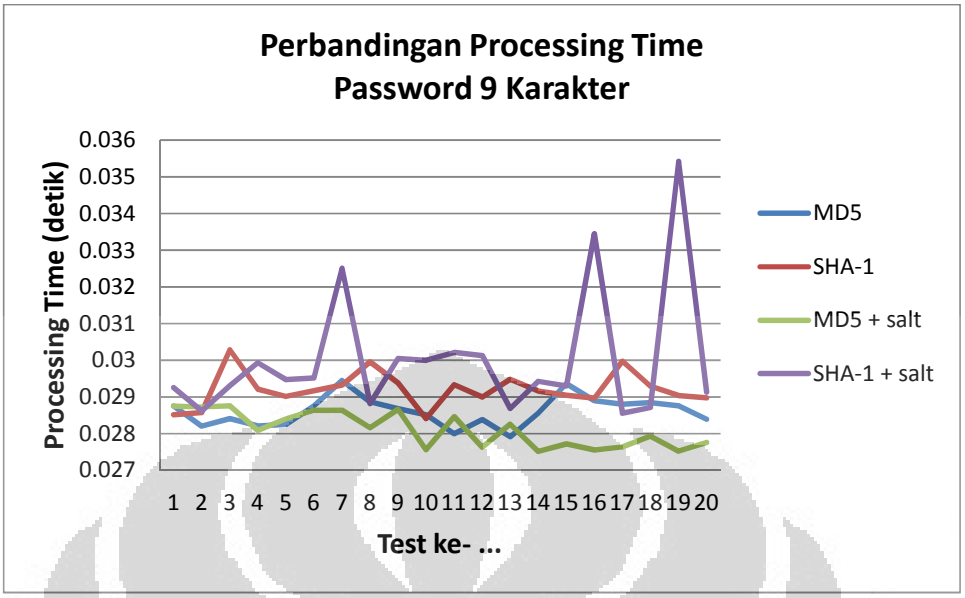
Test ke-	8 Karakter (dalam detik)			
	MD5	MD5 + salt	SHA-1	SHA-1 + salt
1	0.02873898	0.02769804	0.02910709	0.02881503
2	0.02767396	0.027668	0.02923012	0.02983618
3	0.0287869	0.02768493	0.02953887	0.03101611
4	0.03004789	0.02759719	0.02940917	0.02889204
5	0.03016186	0.02855897	0.03101683	0.0299561
6	0.02859402	0.02998805	0.02908707	0.02974486
7	0.02818799	0.02766991	0.02853203	0.03058386
8	0.02880287	0.02771211	0.0284071	0.029006
9	0.02792287	0.02757192	0.02908993	0.02990699
10	0.02819705	0.02860904	0.02912402	0.02935386
11	0.02755189	0.02840805	0.02931213	0.02908015
12	0.02769184	0.02754688	0.02933908	0.02919412
13	0.02841878	0.02859116	0.02873683	0.02846313
14	0.02808714	0.02768087	0.03094292	0.03048706
15	0.02880907	0.02932382	0.03051209	0.02912498
16	0.02889085	0.02779603	0.03014708	0.02929902
17	0.02748418	0.02772808	0.02987194	0.02990294
18	0.02845001	0.02884912	0.02947211	0.030092
19	0.02803802	0.0282321	0.02917385	0.02934599
20	0.02879882	0.02866411	0.02955914	0.02933884
Rata-rata	**0.02846675**	**0.02817892**	**0.02948047**	**0.02957196**



Gambar 4.6 Perbandingan *processing time* untuk *password* 8 karakter

Tabel 4.7 Perbandingan *Processing time* untuk *password* 9 karakter

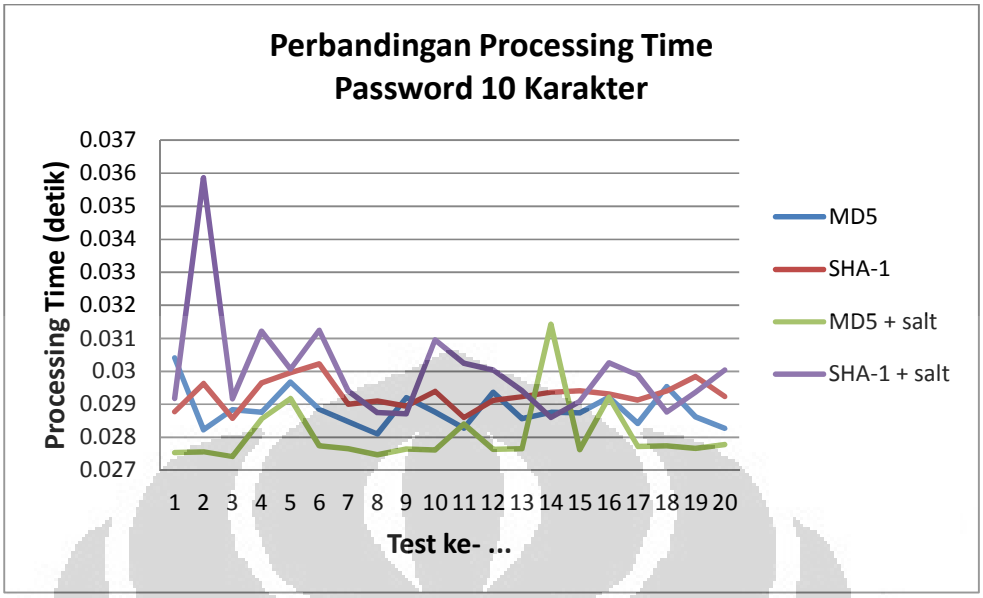
Test ke-	9 Karakter			
	MD5	MD5 + salt	SHA-1	SHA-1 + salt
1	0.02875113	0.02875209	0.02851295	0.02925491
2	0.02820301	0.02873206	0.02857113	0.02862191
3	0.0284121	0.02876496	0.03029084	0.02930999
4	0.02821398	0.02808881	0.029217	0.02993393
5	0.02825499	0.02839899	0.02901506	0.02947402
6	0.02875781	0.02864289	0.02917504	0.02952099
7	0.02945399	0.02863812	0.02933192	0.032511
8	0.02887297	0.0281682	0.02996206	0.02881694
9	0.02868009	0.02866697	0.02938795	0.03005099
10	0.02851987	0.027565	0.02840996	0.03000402
11	0.02800107	0.02846789	0.02933693	0.03021407
12	0.02838898	0.02762914	0.02900004	0.03013611
13	0.02792001	0.0282588	0.02948594	0.02868795
14	0.02855802	0.02751899	0.02916789	0.029423
15	0.02936006	0.02771902	0.02905083	0.02930403
16	0.02889919	0.02756119	0.02896786	0.03345585
17	0.02880001	0.02763796	0.02997303	0.0285542
18	0.02884698	0.02793312	0.02929592	0.02871203
19	0.02876687	0.02752614	0.02903891	0.03542805
20	0.02839303	0.02775908	0.02898002	0.02913022
Rata-rata	**0.02860271**	**0.02812147**	**0.02920856**	**0.03002721**



Gambar 4.7 Perbandingan *processing time* untuk *password* 9 karakter

Tabel 4.8 Perbandingan *Processing time* untuk *password* 10 karakter

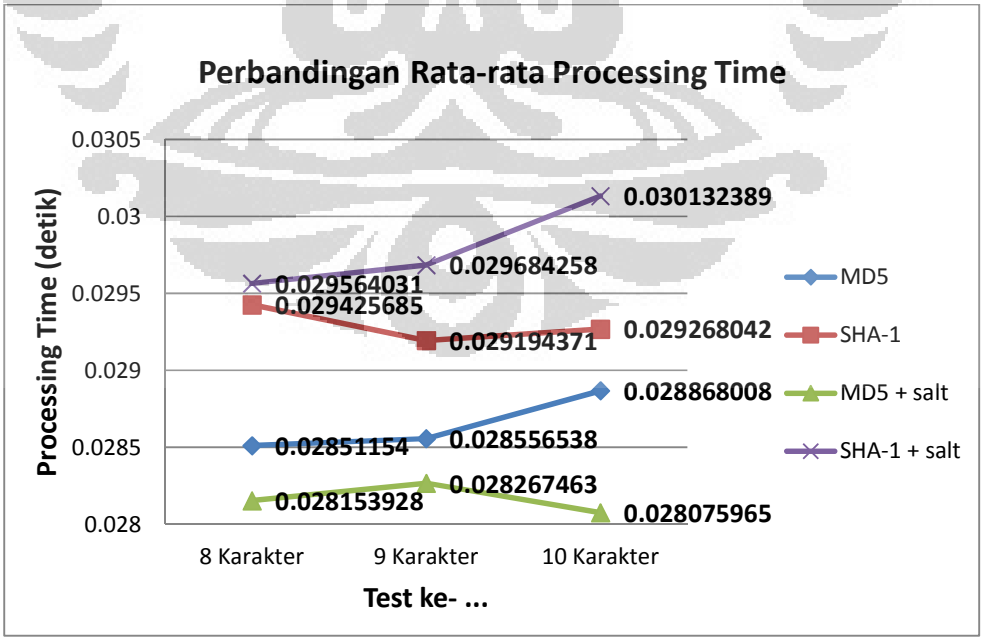
Test ke-	10 Karakter			
	MD5	MD5 + salt	SHA-1	SHA-1 + salt
1	0.03041005	0.02753615	0.02877688	0.029181
2	0.02822804	0.02756596	0.0296371	0.03586793
3	0.02883601	0.02741599	0.02857804	0.02917099
4	0.02875805	0.02855802	0.02965498	0.03121996
5	0.02968001	0.02917409	0.02996397	0.03007007
6	0.02884603	0.0277431	0.0302279	0.0312469
7	0.028476	0.02765799	0.02900815	0.02940893
8	0.02809811	0.02747798	0.02910089	0.02874899
9	0.02920485	0.02764606	0.02894807	0.02871704
10	0.02877092	0.02761507	0.02939487	0.03095293
11	0.02827811	0.02840209	0.02860403	0.03024507
12	0.02936792	0.027637	0.02911901	0.03004599
13	0.02856588	0.02765203	0.02922797	0.02941108
14	0.0287621	0.03142881	0.02936697	0.02860498
15	0.02873802	0.02762914	0.02941179	0.02909398
16	0.02921104	0.02921414	0.02931499	0.03026509
17	0.02841806	0.02772999	0.02912807	0.02989197
18	0.02953911	0.0277431	0.02940798	0.02876902
19	0.02862191	0.027668	0.02984691	0.02936697
20	0.02827311	0.02778506	0.02924109	0.03003883
Rata-rata	**0.02885417**	**0.02806399**	**0.02929798**	**0.03001589**



Gambar 4.8 Perbandingan *processing time* untuk *password* 10 karakter

Tabel 4.9 Rata-rata dan Selisih *Processing time*

Panjang Karakter	Waktu Rata-rata (detik)				Selisih SHA-1+salt dan MD5+salt
	MD5	MD5+salt	SHA-1	SHA-1+salt	
8 Karakter	0.02846675	0.02817892	0.02948047	0.02957196	0.00139304
9 Karakter	0.02860271	0.02812147	0.02920856	0.03002721	0.00190574
10 Karakter	0.02885417	0.02806399	0.02929798	0.03001589	0.00195190



Gambar 4.9 Perbandingan rata-rata *processing time*

b. Analisis

Pada skenario ini didapatkan empat buah tabel hasil pengujian. Masing-masing tabel merepresentasikan hasil pengujian dengan penerapan kombinasi panjang karakter *password* dengan algoritma *hash* yang berbeda serta selisih waktu *processing time* tersebut. Tabel 4.6, 4.7, dan 4.8 menampilkan data hasil pengujian untuk tiap variasi algoritma *hash*. Masing-masing variasi menghasilkan data berupa *processing time* selama proses cekuser. Sedangkan table 4.9 menampilkan selisih rata-rata *processing time*. Dari masing-masing tabel tersebut dibuatlah suatu grafik yang digunakan untuk membandingkan hasil pengujian yang dilakukan.

Gambar 4.6 menunjukkan grafik perbandingan *processing time* yang dibutuhkan untuk mengolah *password* sepanjang 8 karakter pada proses cekuser saat login pada aplikasi Simple-O. Pada Gambar 4.6 dapat terlihat bahwa proses yang dibutuhkan untuk pemrosesan saat login berkisar diantara 0.0275 detik hingga 0.031 detik untuk tiap variasi yang ada. Pada grafik terlihat bahwa ketika algoritma MD5 yang digunakan pada proses autentikasi, maka waktu yang dibutuhkan cenderung lebih cepat, sedangkan saat algoritma SHA-1 yang digabungkan dengan *salt* yang digunakan, maka akan membutuhkan waktu *processing time* yang lebih lama. Namun ada juga kondisi dimana algoritma MD5 memiliki waktu proses yang lebih lama bila dibandingkan dengan SHA-1 + *salt*, seperti pada testing MD5 ke-6 dengan SHA-1 + *salt* ke-13. Selain itu terdapat juga kondisi dimana puncak grafik SHA-1 sama dengan SHA-1 + *salt*, seperti pada testing SHA-1 ke-5 dan ke-14 dengan SHA-1 + *salt* ke-3.

Gambar 4.7 menampilkan grafik hasil pengujian terhadap variasi penerapan algoritma *hash* untuk *password* dengan panjang 9 karakter, hasil yang didapat terlihat agak berbeda dengan grafik sebelumnya. Dari Gambar 4.7 terlihat bahwa penerapan algoritma MD5 + *salt* cenderung membutuhkan waktu yang lebih cepat bila dibandingkan dengan variasi penerapan algoritma lain, bahkan lebih rendah bila dibandingkan dengan MD5. Padahal secara teori seharusnya algoritma MD5 yang divariasikan dengan *salt* akan membutuhkan waktu yang lebih panjang dibandingkan dengan MD5 tanpa *salt*, karena akan

membutuhkan proses *hash* yang lebih panjang. Untuk variasi algoritma lain cenderung sama dengan grafik sebelumnya dengan urutan teratas diisi oleh SHA-1 + *salt* dan diikuti oleh SHA-1.

Pada Gambar 4.8, anomali yang terjadi pada grafik sebelumnya kembali terulang, dimana *processing time* MD5 + *salt* justru lebih rendah daripada MD5, bahkan hampir diseluruh *sequence testing*. Hanya satu kali, yaitu pada testing ke-14, dimana algoritma MD5 + *salt* lebih memiliki *processing time* yang lebih tinggi dibandingkan MD5 bahkan lebih tinggi daripada variasi algoritma lain dalam urutan testing yang sama. Sedangkan untuk kondisi lain kecenderungannya sama dengan grafik sebelumnya dimana SHA-1 dengan *salt* membutuhkan waktu *processing time* paling tinggi yang kemudian diikuti oleh SHA-1.

Gambar 4.9 menunjukkan grafik rata-rata *processing time* yang dibutuhkan untuk tiap variasi panjang karakter *password* dengan variasi algoritma yang diterapkan. Pada grafik ini terlihat bahwa algoritma MD5 + *salt* memiliki rata-rata *processing time* paling rendah pada tiap variasi panjang karakter *password* bila dibandingkan dengan variasi algoritma lain, kemudian diikuti dengan MD5, SHA-1, dan SHA-1 + *salt* pada posisi teratas. Anomali yang terlihat pada dua grafik sebelumnya, dimana MD5 memiliki *processing time* lebih tingi dibandingkan dengan MD5 + *salt*, terlihat jelas pada grafik ini.

Pada skenario pengujian ini ditemukan beberapa anomali yang bertentangan dengan teori yang ada. Jika dianalisis lebih lanjut, anomali yang terjadi ini disebabkan oleh proses lain yang berjalan di *processor* saat dilakukan pengujian terhadap penerapan algoritma MD5. Karena pengujian sebanyak 60 kali, masing-masing 20 kali untuk tiap variasi panjang *password* ini dilakukan secara kontinyu dalam periode waktu yang tidak berjauhan, jadi bisa saja terdapat suatu proses lain yang berjalan pada *server*, yang ikut menggunakan alokasi CPU, sehingga *processing time* untuk MD5 lebih panjang dibandingkan dengan MD5 + *salt,* ataupun rata-rata *processing time* untuk *password* dengan 8 dan 9 karakter lebih tinggi dibandingkan *password* 10 karakter pada algoritma MD5+salt, dan rata-rata *processing time* untuk

password 8 karakter lebih tinggi dibandingkan dengan *password* 9 dan 10 karakter seperti pada SHA-1.

Bila dianalisis lebih lanjut, perbedaan waktu yang terjadi pada setiap variasi yang ada, baik variasi algoritma dan *password*, memiliki selisih yang berkisar pada angka 10^{-4} detik. Pada perhitungan matematika, angka perhitungan yang dapat dihitung minimal adalah hingga 2 angka dibelakang koma, sedangkan angka dibelakangnya cenderung dapat diabaikan, meskipun pada konteks pengujian tertentu tidak dapat diabaikan. Pada pengujian ini, pada dasarnya pembulatan dapat dilakukan hingga 2 angka dibelakang koma. Angka-angka yang terdapat pada tabel pengujian diatas merupakan nilai yang didapat berdasarkan perhitungan *microtime* pada PHP, sehingga angka yang didapatkan mencapai digit 10^{-9}, namun angka-angka setelah 2 digit dibelakang koma bisa saja diabaikan. Oleh karena itu dapat disimpulkan bahwa *processing time* yang dibutuhkan untuk penerapan algoritma MD5, MD5 + *salt*, SHA-1, dan SHA-1 + *salt* cenderung sama. Selisih yang ada sangatlah tipis bedanya, sehingga dapat dikatakan sama. Maka dapat disimpulkan bahwa penerapan algoritma SHA-1 + *salt* pada aplikasi Simple-O tidak akan membebani sistem yang ada.

Pengukuran *processing time* ini dilakukan pada jaringan lokal. Apabila aplikasi ini di-*hosting*, dan diakses melalui internet, maka *processing time* yang ada mungkin saja berbeda atau bahkan lebih tinggi. Hal tersebut bisa disebabkan banyak faktor lain yang terdapat dalam jaringan. Namun berdasarkan pengujian ini dapat juga disimpulkan bahwa kemungkinan *processing time* yang lebih tinggi saat aplikasi ini di-hosting tidaklah disebabkan oleh proses algoritma SHA-1 yang lebih rumit daripada MD5, namun bisa saja disebabkan oleh berbagai faktor lain yang terjadi pada jaringan.

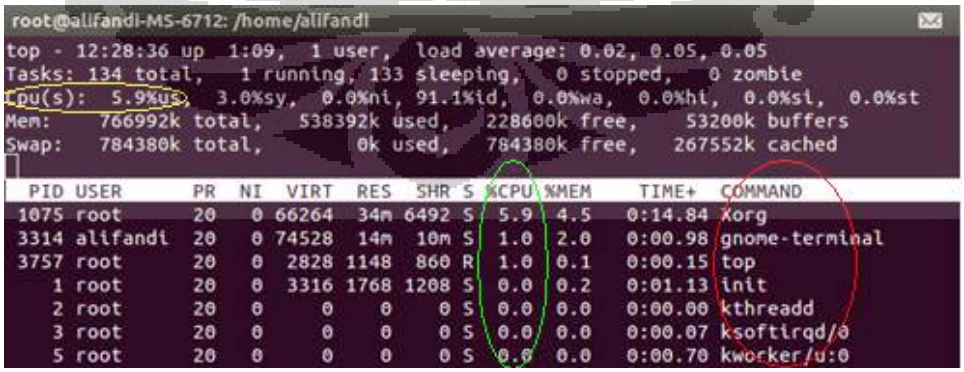
4.3 Skenario Pengukuran *CPU usage*

Pada skenario ini dilakukan perhitungan *CPU usage* pada proses login Simple-O. Perhitungan hasil pengujian pada skenario ini menunjukkan perbandingan hasil penerapan algoritma MD5 dan SHA-1 dalam pengaruhnya terhadap performa aplikasi dan *server* Simple-O.

Pada saat melakukan login, maka proses apache2 dan mysqld akan berjalan pada *server*, yang tentunya berpengaruh pada penggunaan *CPU usage*. Pengujian dilakukan dengan menggunakan *command* line `top` pada Linux. Dengan menggunakan `top`, maka berbagai aktivitas dari *server* dapat dipantau, seperti aplikasi yang sedang berjalan, bagaimana kondisi *CPU usage*, *memory usage*, dan *swap*. Pada pengujian ini *command line* `top` di-set untuk menampilkan perubahan kondisi *CPU usage* setiap 1 detik. Untuk menampilkan `top` pada tiap detik, maka digunakanlah *command line* sebagai berikut :

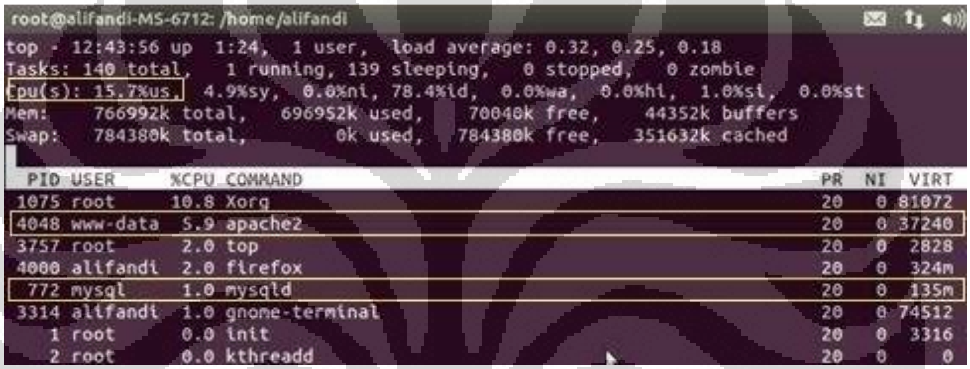
```
root@alifandi-MS-6712:/home/alifandi# top -d 1
```

Setelah perintah diatas dieksekusi, maka semua aktivitas yang berjalan pada *server* dapat dipantau perubahnnya pada setiap detik. Data yang diambil merupakan *CPU usage* total dari seluruh aktivitas yang berjalan pada tiap detik tersebut. Meskipun dapat dilakukan pengambilan data untuk *CPU usage* dari proses apache2 dan mysqld saat login saja, namun data *CPU usage* yang diambil sebagai data adalah *CPU usage* total, karena dengan menggunakan data *CPU usage* total maka kondisi *server* secara keseluruhan dapat dipantau. Karena bagaimanapun juga aktivitas yang berjalan pada *server* tidak hanya aplikasi Simple-O. Gambar 4.10 menampilkan kondisi saat *command* `top` dijalankan, *CPU usage* total dilingkari dengan lingkaran berwarna kuning, *CPU usage* peraktifitas dilingkari dengan lingkaran berwarna hijau, dan aktivitasnya dilingkari dengan lingkaran berwarna merah.



Gambar 4.10 Menjalankan *command* `top` pada Linux

Sama seperti skenario sebelumnya, skenario ini dijalankan dengan 4 macam kombinasi algoritma *hash* yang diterapkan secara bergantian pada aplikasi Simple-O, yaitu MD5, SHA-1, MD5 dengan *salt*, dan SHA-1 dengan *salt*. Untuk tiap-tiap kombinasi, dilakukan 60 kali proses login, dan diambil data *CPU usage*-nya pada saat proses apache2 dan mysqld berjalan pada *server*. Pada tiap penerapan kombinasi algoritma juga digunakan 3 kombinasi *password* dengan panjang 8, 9, dan 10 karakter seperti pada skenario sebelumnya. Gambar 4.11 menunjukkan berjalannya proses apache2 dan mysqld yang dipantau dengan *command* `top` :



Gambar 4.11 Proses apache2 dan mysqld yang berjalan saat dilakukan proses login

Berikut ini adalah *plaintext password* yang digunakan pada skenario ini:

- 8 karakter : ahmad100
- 9 karakter : shaugi100
- 10 karakter : farchan100

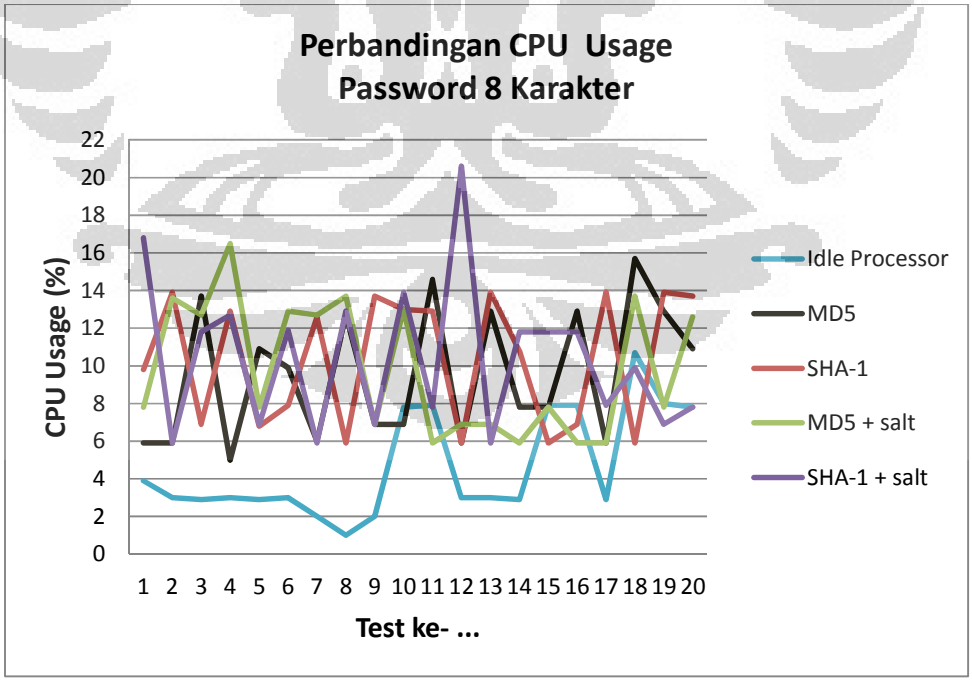
a. Hasil Pengukuran

Tabel 4.10 Kondisi *idle CPU* selama 20 detik

Detik ke-…	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Rata-rata**
CPU Usage (%)	3.9	3	2.9	3	2.9	3	2	1	2	7.8	3.15
Detik ke-…	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	
CPU Usage (%)	7.9	3	3	2.9	7.9	7.9	2.9	11	8	7.8	6.2
									Rata-rata		**4.68**

Tabel 4.11 Perbandingan *CPU usage* untuk *password* 8 karakter

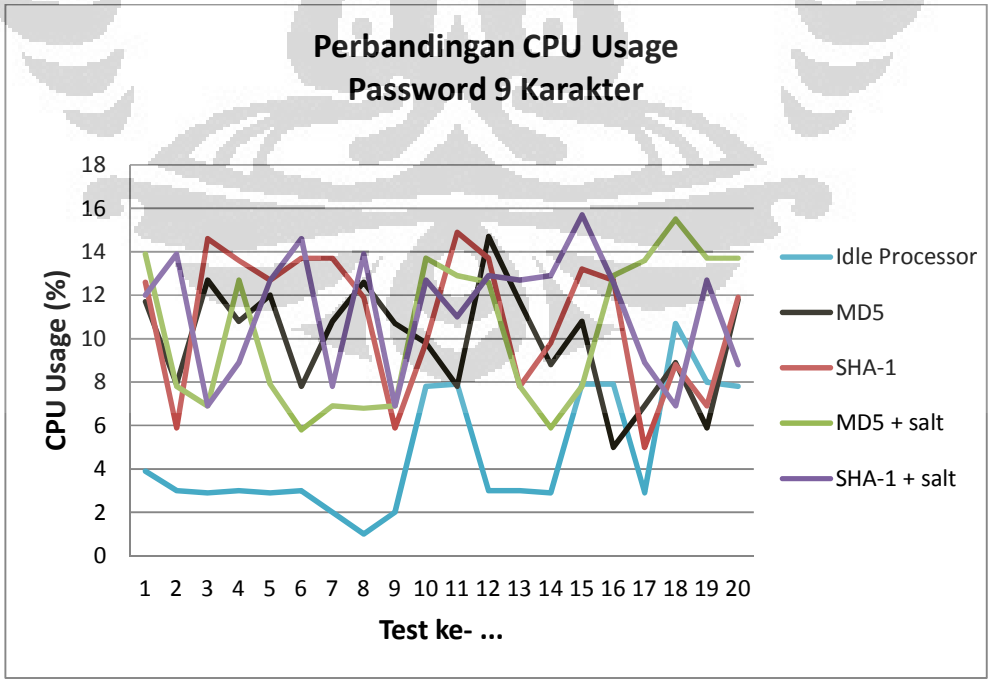
Test ke-	8 Karakter			
	MD5	MD5 + salt	SHA-1	SHA-1 + salt
1	5.9	7.8	9.8	16.8
2	5.9	13.6	13.9	5.9
3	13.7	12.7	6.9	11.8
4	5	16.5	12.9	12.7
5	10.9	7.8	6.8	6.9
6	9.9	12.9	7.9	11.9
7	6	12.7	12.6	5.9
8	12.7	13.7	5.9	12.9
9	6.9	6.9	13.7	6.9
10	6.9	13	13	13.9
11	14.6	5.9	12.9	7.8
12	5.9	6.9	5.9	20.6
13	12.9	6.9	13.9	5.9
14	7.8	5.9	10.8	11.8
15	7.8	7.8	5.9	11.8
16	12.9	5.9	6.9	11.8
17	5.9	5.9	13.9	7.9
18	15.7	13.7	5.9	9.9
19	12.9	7.8	13.9	6.9
20	10.9	12.6	13.7	7.8
Rata-rata	**9.555**	**9.845**	**10.355**	**10.39**



Gambar 4.12 Perbandingan *CPU usage* untuk *password* 8 karakter

Tabel 4.12 Perbandingan *CPU usage* untuk *password* 9 karakter

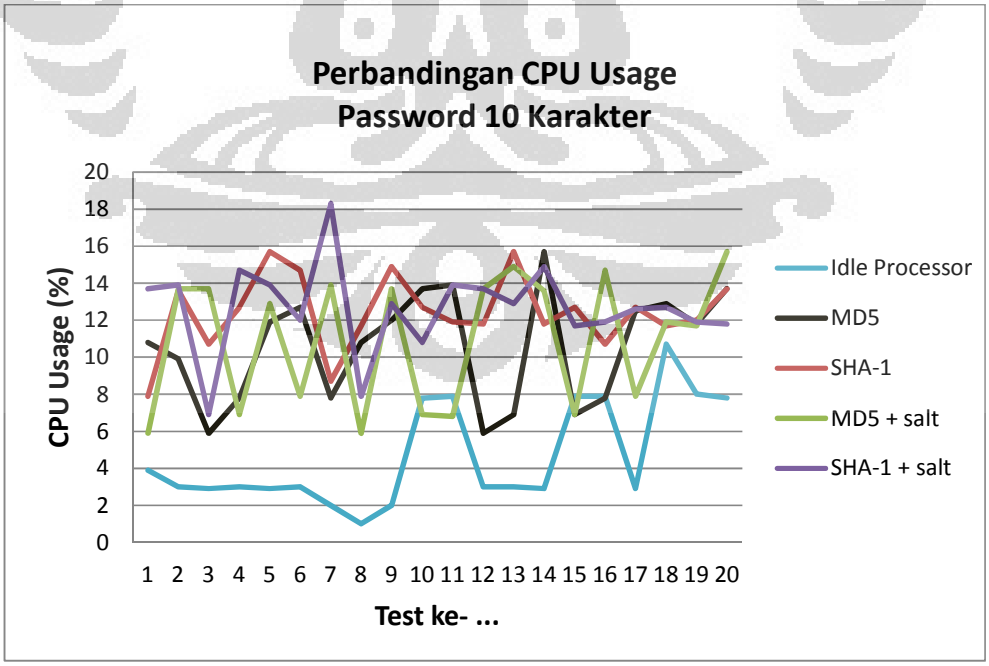
Test ke-	9 Karakter			
	MD5	MD5 + salt	SHA-1	SHA-1 + salt
1	11.7	13.9	12.6	12
2	7.9	7.8	5.9	13.9
3	12.7	6.9	14.6	6.9
4	10.8	12.7	13.6	8.9
5	12	7.9	12.7	12.7
6	7.8	5.8	13.7	14.6
7	10.8	6.9	13.7	7.8
8	12.6	6.8	11.9	13.9
9	10.7	6.9	5.9	6.9
10	9.8	13.7	9.8	12.7
11	7.8	12.9	14.9	11
12	14.7	12.6	13.7	12.9
13	11.7	7.8	7.8	12.7
14	8.8	5.9	9.8	12.9
15	10.8	7.8	13.2	15.7
16	5	12.9	12.7	12.6
17	6.9	13.6	5	8.9
18	8.9	15.5	8.8	6.9
19	5.9	13.7	6.9	12.7
20	11.8	13.7	11.9	8.8
Rata-rata	**9.955**	**10.285**	**10.955**	**11.27**



Gambar 4.13 Perbandingan *CPU usage* untuk *password* 9 karakter

Tabel 4.13 Perbandingan *CPU usage* untuk *password* 10 karakter

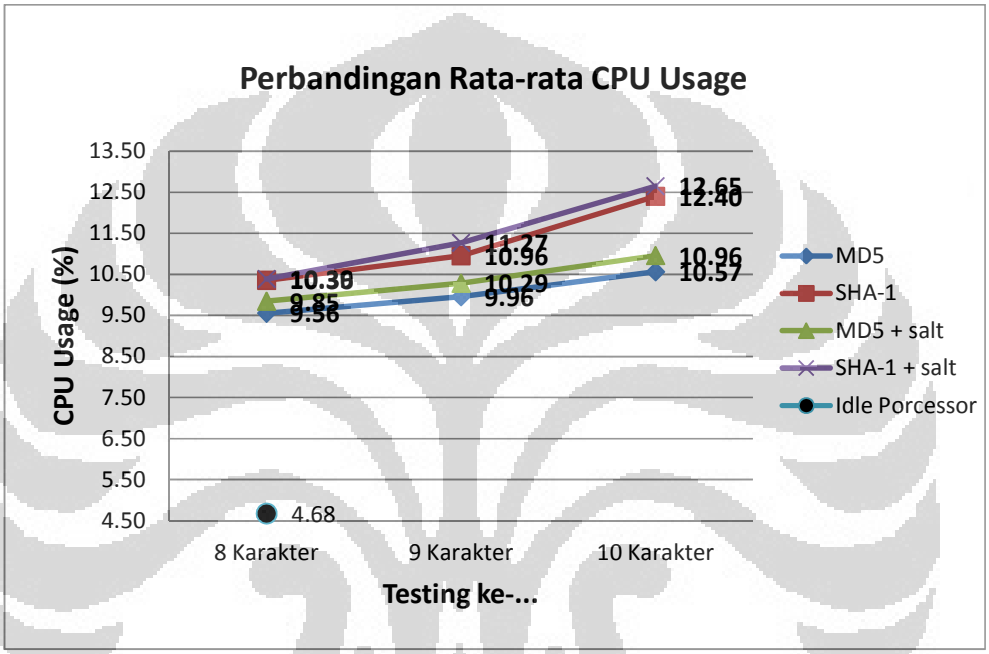
Test ke-	10 Karakter			
	MD5	MD5 + salt	SHA-1	SHA-1 + salt
1	10.8	5.9	7.9	13.7
2	9.9	13.7	13.6	13.9
3	5.9	13.7	10.7	6.9
4	7.8	6.9	12.7	14.7
5	11.9	12.9	15.7	13.9
6	12.7	7.9	14.7	12
7	7.8	13.9	8.7	18.3
8	10.8	5.9	11.7	7.9
9	12	13.7	14.9	12.9
10	13.7	6.9	12.7	10.8
11	13.9	6.8	11.9	13.9
12	5.9	13.7	11.8	13.7
13	6.9	14.9	15.7	12.9
14	15.7	13.6	11.8	14.9
15	6.9	6.9	12.7	11.7
16	7.8	14.7	10.7	11.9
17	12.5	7.9	12.7	12.6
18	12.9	11.9	11.7	12.7
19	11.8	11.7	12	11.9
20	13.7	15.7	13.7	11.8
Rata-rata	**10.565**	**10.96**	**12.4**	**12.65**



Gambar 4.14 Perbandingan *CPU usage* untuk *password* 10 karakter

Tabel 4.14 Rata-rata dan Selisih *CPU usage*

Panjang Karakter	*CPU Usage* Rata-rata (%)				Selisih SHA-1+salt dan MD5+salt
	MD5	MD5+salt	SHA-1	SHA-1+salt	
8 Karakter	9.555	9.845	10.355	10.39	0.545
9 Karakter	9.955	10.285	10.955	11.27	0.985
10 Karakter	10.565	10.96	12.4	12.65	1.69



Gambar 4.15 Perbandingan rata-rata *CPU usage*

b. Analisis

Pada skenario ini, hasil pengujian terdiri atas lima buah tabel hasil pengujian, yaitu Tabel 4.10 yang menampilkan kondisi idle dari prosesor yang diambil selama 20 detik. Tabel 4.11, 4.12, dan 4.13 menampilkan hasil pengujian processing time untuk tiap variasi algoritma dan variasi panjang karakter password masing-masing 8, 9, dan 10 karakter. Masing-masing variasi yang ada menghasilkan data berupa besaran *CPU usage* pada *server* yang digunakan selama proses login. Data *CPU usage* saat proses login diambil tiap kali proses login dilakukan atau saat proses apache2 dan mysqld berjalan.Terakhir yaitu Tabel 4.14 menampilkan rata-rata dan selisih *CPU usage* dari tiap variasi. Dari masing-masing tabel tersebut dibuatlah suatu grafik yang digunakan untuk membandingkan hasil pengujian yang dilakukan.

Gambar 4.12 menunjukkan grafik perbandingan penggunaan *resource* CPU dengan variasi penerapan algoritma *hash* pada aplikasi Simple-O untuk pemrosesan *password* dengan panjang 8 karakter. Pada grafik terlihat bahwa terdapat perubahan *CPU usage* saat *processor* berada pada kondis idle dan pada saat digunakan untuk proses login aplikasi Simple-O. Ada beberapa kondisi dimana *CPU usage* saat idle besarnya sama dengan saat proses login, seperti pada pengambilan data idle ke-10, 11, 15, 16, dan 18-20. Namun secara umum kondisi *CPU usage* saat idle lebih rendah dibandingkan saat menjalankan proses login. Sedangkan besarnya *CPU usage* saat proses login terlihat sangat fluktuatif pada Gambar 4.12. Perubahan *CPU usage* yang terjadi untuk setiap variasi algoritma berkisar diantara 6% hingga 14%, namun tetap ada yang berada dibawah ataupun diatas kisaran tersebut.

Pada Gambar 4.13, terlihat grafik perbandingan *CPU usage* untuk pemrosesan *password* sepanjang 9 karakter. Jarak antara *CPU usage* saat idle dengan saat proses login terlihat agak menjauh dibandingkan grafik sebelumya. Besar *CPU usage* saat proses login juga terlihat fluktuatif seperti pada grafik sebelumnya. Rentang perubahan *CPU usage* pada Gambar 4.13 ini juga berkisar antara 6% hingga 14%. Rentang yang ada sama dengan sebelumnya, namun tentunya ada beberapa kondisi dimana *CPU usage* lebih tinggi ataupun lebih rendah dari kisaran ini.

Gambar 4.14 menunjukkan grafik perbandingan *CPU usage* untuk pemrosesan *password* sepanjang 10 karakter. Jarak antara kondisi *CPU usage* saat idle dengan saat proses login-pun terlihat semakin menjauh dibandingkan 2 grafik sebelumnya, meskipun tidaklah signifikan. Sama seperti 2 grafik sebelumnya, kondisi *CPU usage* saat proses login juga terlihat fluktuatif perubahannya.

Tampilan grafik yang sangat fluktuatif pada Gambar 4.12, 4.13, dan 4.14, menunjukkan bahwa adanya proses lain yang berjalan bersamaan dengan proses *login*. Proses lain inilah yang menyebabkan besar *CPU usage* menjadi sangat fluktuatif seprti yang ditampilkan pada grafik, karena proses-proses lain yang berjalan ini menggunakan besar *resource CPU* yang berbeda satu sama lain.

Sedangkan Gambar 4.15 menunjukkan grafik perbandingan rata-rata perubahan *CPU usage* untuk setiap variasi panjang karakter *password* yang digunakan, pada setiap variasi penerapan algoritma *hash* untuk proses login. Pada grafik ini juga terdapat satu poin yang menunjukkan rata-rata *CPU usage* saat idle, yaitu sebesar 4.68%. Pada grafik terlihat bahwa algoritma MD5 menggunakan *resource* CPU paling rendah untuk setiap rata-rata variasi panjang karakter *password*, yang kemudian diikuti oleh MD5 + *salt*, SHA-1, dan SHA-1 + *salt*.

Berdasarkan Gambar 4.15, perubahan *CPU usage* yang terjadi pada proses login aplikasi Simple-O saat diterapkan algortima SHA-1 + *salt*, bila dibandingkan dengan proses login saat penerapan algoritma yang sebelumnya, yaitu MD5 + *salt*, maka selisih *CPU usage*-nya adalah 0.74% untuk *password* 8 karakter, 0.98% untuk *password* 9 karakter, dan 1.69% untuk *password* 10 karakter. Selisih yang ada terlihat meningkat seiring bertambahnya panjang karakter *password*. Hal ini menunjukkan bahwa penggunaan algoritma SHA-1 + *salt* akan meningktakan penggunaan *resource CPU* saat proses login. Namun perubahan yang terjadi tidaklah terlalu signifikan untuk panjang *password* hingga 10 karakter, yaitu 1.69%.

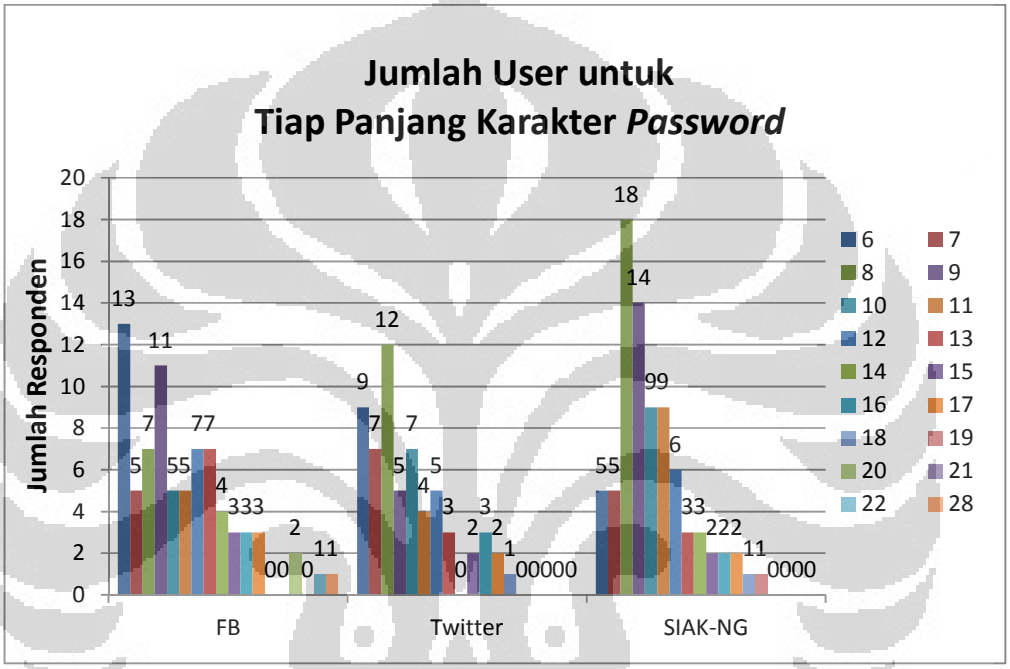
Untuk *password* dengan panjang karakter lebih dari 10, diperkirakan selisihnya juga akan bertambah. Hal ini mungkin memberi indikasi bahwa penggunaan algoritma SHA-1 + salt akan membebani aplikasi Simple-O, apalagi bila banyak user yang mengakses dalam waktu bersamaan. Namun beban yang diberikan kepada *server* diperkirakan tidak akan menjadi terlalu besar, mengingat panjang karakter *password* dari setiap user akan sangat bervariasi, karena tidak semua user menggunakan *password* dengan panjang 10 karakter atau lebih, ada juga user yang menggunakan *password* dengan panjang kurang dari 8 karakter. Berdasarkan hasil penyebaran kuesioner terhadap 80 responden, yang terdiri atas mahasiswa Teknik Elektro, untuk mengukur panjang karakter yang biasa digunakan orang-orang, didapatkan data sebagai seperti yang ditampilkan pada Tabel 4.15 berikut:

Tabel 4.15 Data Survey Panjang Karakter *Password* yang Biasa Digunakan User

No.	Nama	*Password*			*Good password*	Menerapkan	No.	Nama	*Password*			*Good password*	Menerapkan
		FB	Twitter	SIAK-NG					FB	Twitter	SIAK-NG		
1	Responden 1	9	-	10	-	-	41	Responden 41	6	7	8	✓	-
2	Responden 2	6	8	8	✓	✓	42	Responden 42	13	-	8	-	-
3	Responden 3	16	16	10	✓	✓	43	Responden 43	6	-	10	✓	-
4	Responden 4	6	9	7	✓	✓	44	Responden 44	20	-	19	✓	✓
5	Responden 5	6	7	8	✓	✓	45	Responden 45	8	6	8	✓	✓
6	Responden 6	14	9	9	✓	✓	46	Responden 46	17	17	17	✓	✓
7	Responden 7	8	7	16	✓	✓	47	Responden 47	13	6	8	✓	-
8	Responden 8	13	8	12	✓	✓	48	Responden 48	8	8	8	✓	✓
9	Responden 9	15	15	15	✓	✓	49	Responden 49	9	6	8	✓	✓
10	Responden 10	8	8	9	✓	✓	50	Responden 50	-	-	10	✓	✓
11	Responden 11	12	17	17	✓	✓	51	Responden 51	10	8	9	✓	-
12	Responden 12	6	-	13	✓	✓	52	Responden 52	9	-	8	✓	✓
13	Responden 13	16	8	10	✓	✓	53	Responden 53	9	8	12	✓	✓
14	Responden 14	6	-	7	✓	✓	54	Responden 54	9	7	9	✓	✓
15	Responden 15	9	9	9	✓	✓	55	Responden 55	8	8	8	✓	-
16	Responden 16	12	12	18	✓	✓	56	Responden 56	9	8	9	✓	✓
17	Responden 17	8	8	8	✓	✓	57	Responden 57	14	10	14	✓	✓
18	Responden 18	-	-	12	✓	✓	58	Responden 58	9	-	9	✓	-
19	Responden 19	6	6	6	✓	-	59	Responden 59	10	12	9	✓	✓
20	Responden 20	12	12	11	-	-	60	Responden 60	7	7	10	✓	✓
21	Responden 21	9	9	8	✓	✓	61	Responden 61	20	-	6	✓	✓
22	Responden 22	11	6	8	✓	✓	62	Responden 62	6	-	6	✓	-
23	Responden 23	15	15	8	-	-	63	Responden 63	6	10	11	✓	✓
24	Responden 24	6	6	9	✓	✓	64	Responden 64	12	-	10	✓	-
25	Responden 25	12	6	13	✓	✓	65	Responden 65	11	-	10	✓	-
26	Responden 26	14	10	14	✓	✓	66	Responden 66	7	7	9	✓	✓
27	Responden 27	11	11	9	✓	✓	67	Responden 67	7	11	12	✓	-
28	Responden 28	28	10	11	✓	✓	68	Responden 68	6	8	6	-	-
29	Responden 29	15	10	15	✓	✓	69	Responden 69	9	13	14	✓	✓
30	Responden 30	-	-	9	✓	-	70	Responden 70	10	10	11	✓	-
31	Responden 31	14	-	11	✓	-	71	Responden 71	7	-	8	✓	✓
32	Responden 32	7	-	7	✓	-	72	Responden 72	10	10	16	✓	-
33	Responden 33	11	-	12	✓	-	73	Responden 73	13	18	7	✓	-
34	Responden 34	13	12	6	✓	-	74	Responden 74	22	16	13	✓	✓
35	Responden 35	6	6	7	✓	-	75	Responden 75	16	16	12	✓	✓
36	Responden 36	10	12	9	✓	-	76	Responden 76	11	-	11	✓	-
37	Responden 37	17	11	11	✓	-	77	Responden 77	8	8	8	✓	-
38	Responden 38	12	11	8	✓	-	78	Responden 78	9	9	11	✓	-
39	Responden 39	13	7	10	✓	-	79	Responden 79	12	13	8	✓	✓
40	Responden 40	17	6	9	✓	-	80	Responden 80	13	13	11	✓	✓

Tabel 4.16 Statistik Panjang Karakter *Password* Yang Biasa Digunakan

Variabel Pengukuran	Panjang Karakter *Password*		
	FB	Twitter	SIAK-NG
Modus	6	8	8
Maksimum	28	18	19
Minimum	6	6	6



Gambar 4.16 Jumlah user untuk tiap panjang karakter *password*

Tabel 4.15 dan 4.16, serta Gambar 4.16 dapat dijadikan acuan panjang *password* yang biasa digunakan user untuk akun Facebook, Twitter, dan SIAK-NG. Dengan modus 6, 8, dan 8 karakter untuk *password* Facebook, Twitter, dan SIAK-NG, dan rentang panjang *password* yang paling banyak digunakan berkisar antara 6 hingga 9 karakter seperti terlihat pada Gambar 4.16, maka penerapan SHA-1 tidak akan membebani sistem yang ada apabila dilihat dari sisi panjang *password* yang biasa digunakan, apalagi didukung oleh *processing time* saat proses login yang sangat cepat. Karena proses login sangat cepat, maka *resource CPU* yang digunakan untuk proses autentikasi menggunakan SHA-1 + *salt* juga tidak akan terlalu lama.

Selain itu, apabila PC *server* yang digunakan pada sistem pengujian benar-benar menggunakan PC dengan spesifikasi *server* yang sebenarnya, maka beasr *CPU usage* saat login juga akan lebih rendah dari penggunaan PC spesifikasi standar seperti ini.

Namun untuk dapat betul-betul mendapatkan hasil apakah penerapan algoritma ini membebani sistem atau tidak, masih diperlukan pengujian untuk mengukur beban saat web dihosting dan ada lebih dari satu user yang login dalam satu waktu. Hanya saja pengujian tersebut tidak tercakup dalam skenario skripsi ini. Kemampuan PC *server* pada pengujian ini yang notabene tidak berspesifikasi PC *server* pada umumnya juga berpengaruh pada penggunaan *resource CPU* pada tiap kali proses login.

Analisa terhadap grafik rata-rata variasi perbandingan *CPU usage* juga memunculkan suatu fakta yang menarik. Berdasarkan teori yang ada, *CPU usage* pada saat penggunaan algoritma SHA-1 akan lebih tinggi dibandingkan dengan MD5, dan hal tersebut terlihat pada grafik rata-rata. Namun hal yang menarik adalah, bahwa meskipun algoritma MD5 yang digunakan sudah digabungkan dengan *salt* sehingga menambahkan kerumitan prosesnya, namun ternyata *CPU usage* untuk SHA-1 tetaplah lebih tinggi dibandingkan dibandingkan MD5 + *salt*. Hal ini dapat dijadikan indikator bahwa algoritma SHA-1 tetap lebih baik dibandingkan MD5 + *salt*, apalagi bila dikombinasikan dengan penggunaan *salt*.

Selain itu, munculnya data yang fluktuatif pada Gambar 4.12, 4.13, dan 4.14 menunjukkan bahwa ada proses lain yang berjalan pada *processor* selain untuk pemrosesan login Simple-O, sehingga nilai *CPU usage* yang muncul menjadi fluktuatif. Hal ini menjawab anomali yang terjadi pada skenario sebelumnya, dimana *processing time* MD5 lebih tinggi dibandingkan MD5 + *salt*. Hal ini menguatkan argumen bahwa ada proses lain yang bekerja selama pengambilan data *processing time* MD5 sehingga menghasilkan grafik MD5 yang lebih tinggi dibandingkan MD5 + *salt*.

BAB V
KESIMPULAN

1. Hasil pengujian dan perbandingan *brute force attack* untuk *password* dengan panjang 6 karakter, menunjukkan bahwa algoritma SHA-1 lebih kuat dari algoritma MD5, dimana SHA-1 memiliki rata-rata waktu sebesar 9.71 menit, sedangkan MD5 8.65 menit. Begitu juga pada waktu estimasi *brute force* terhadap *password* sepanjang 7 hingga 9 karakter, dimana waktu estimasi *brute force* SHA-1 untuk *password* sepanjang 7, 8, dan 9 karakter masing-masing 18.92 jam, 974.06 jam, dan 8397.8 jam. Sedangkan MD5 masing-masing selama 16.34 jam, 860.51 jam, dan 8229.95 jam.
2. Algoritma SHA-1 berhasil diterapkan pada sistem autentikasi aplikasi Simple-O.
3. Pada pengujian dan perbandingan perhitungan *processing time* saat login, didapatkan hasil yang menunjukkan bahwa algoritma SHA-1 memiliki *processing time* yang tidak berbeda jauh dari MD5 bahkan terbilang sama. Algoritma MD5 memiliki *processing time* 0.029 detik untuk tiap variasi panjang *password* (8, 9, 10 karakter), MD5+*salt* sebesar 0.028 detik, SHA-1 sebesar 0.029 detik, dan SHA-1+*salt* 0.03 detik.
4. Pada pengukuran dan perbandingan *CPU usage* saat proses login, didaptkan hasil bahwa SHA-1 menggunakan *resource CPU* yang lebih besar dari MD5, namun perbedaan keduanya tidaklah terlalu jauh. Algoritma MD5 menggunakan *resource CPU* untuk *password* dengan panjang 8, 9, dan 10 karakter sebesar masing-masing 9.56%, 9.96%, dan 10.57%. Sedangkan MD5+*salt* masing-masing 9.85%, 10.29%, dan 10.96%, SHA-1 masing-masing 10.36%, 10.96%, dan 12.4%, dan SHA-1+*salt* masing-masing 10.39%, 11.27%, dan 12.65%.

5. Survey terhadap 80 responden untuk mengukur panjang karakter *password* yang biasa digunakan menunjukkan bahwa sebagian besar user menggunakan *password* dengan panjang 6 hingga 9 karakter. Oleh karena itu penerapan algoritma SHA-1 + *salt* pada aplikasi Simple-O tidak akan membebani *server* dan aplikasi yang berjalan.
6. Kompleksitas algoritma SHA-1 juga terlihat pada pengujian ketiga, dimana algoritma SHA-1 menggunakan *resource* CPU yang lebih besar bahkan dari algoritma MD5+*salt*, yang sudah meningkat kompleksitasnya karena penambahan *salt*. Selisih penggunaan *resource* CPU antara algoritma SHA-1 dan MD5 juga kian meningkat seiring bertambahnya panjang karakter dari *plaintext* yang mengalami proses *hash*.

DAFTAR REFERENSI

[1] http://www.pcmag.com/encyclopedia_term/0,2542,t=Web+application&i=54272,00.asp#fbid=IIytiqWAyQZ. Diakses pada 30 Oktober 2011.

[2] "Web-Based Application?". 18 September 2009. http://bilcyber.com/2009/09/web-based-application/, diakses tanggal 30 Oktober 2011.

[3] Islam, Md. M. & Hoque, A. S. M. L. (2010). Automated Essay Scoring Using Generalized Latent Semantic Analysis. *13th International Conference on Komputer and Information Technology*.

[4] Putri Ratna, Anak Agung; Budiardjo, Bagio; & Hartanto, Djoko. (2007). "SIMPLE: Sistem Penilaian Esei Otomatis untuk Menilai Ujian dalam Bahasa Indonesia". Jurnal Makara Seri Teknologi, volume 11, April 2007, ISSN : 1693-6698.

[5] Stalling, William. "Network Security Essentials : Applications and Standards, 4th Ed". New Jersey : Prentice Hall, 2011.

[6] Saptono, Henry. (2007) "Web Security". Depok : LP3T Nurul Fikri. *overflow.**web**.id/source/**Web-Security**.pdf, diakses tanggal 30 Oktober 2011.*

[7] Alwi, Najwa H. M. & Fan, Ip-Shing. (2009). Information Security Management in E-Learning. *International Conference for Internet Technology and Secured Transaction*.

[8] Stalling, William. "Cryptography and Network Principles and Practices, 4th Ed". New Jersey : Prentice Hall, 2005.

[9] Alim Sutanto, Candra. (2011). Algoritma Fungsi *Hash* Baru dengan Menggabungkan MD5, SHA-1 dan Penyertaan Panjang Pesan Asli. Sekolah Tinggi Elektro dan Informatika, Institut Teknologi Bandung.

[10] Pamungkas, Angger Ardyanto, dkk. (2006). Implementasi Algoritma Sistem Kriptografi MD5, SHA1, dan RC4 Pada Aplikasi Mobile Internet Berbasis Java. *Jurnal Penelitian dan Pengembangan TELEKOMUNIKASI, Juni 2006, Vol. 11, No. 1.*

[11] Wicaksono, Rizki. 2009. "MD5 itu Berbahaya, Titik!". (http://www.ilmuhacking.com/cryptography/md5-itu-berbahaya/, diakses tanggal 19 Desember 2011).

[12] Ardi . 2011. "Perbedaan Antara MD5 dan SHA". (http://dark-holi.blogspot.com/2011/03/perbedaan-antara-md5-dan-sha.html, diakses tanggal 13 April 2012).

[13] Duan, Bing et al. (2008). "An Easy-to-deploy *Penetration testing Platform*". *The 9th International Confrence for Young Komputer Scientists.*

[14] "*Brute force attack*". http://www.techopedia.com/definition/18091/brute-force-attack, diakses tanggal 13 April 2012.

[15] Handoyo, G. "PENGEMBANGAN SISTEM KEAMANAN SIMPLE-O", Skripsi. : Departemen Teknik Elektro, Universitas Indonesia Depok. 2010.

[16] Oneiric Ocelot/Technical Overview. https://wiki.ubuntu.com/OneiricOcelot/ReleaseNotes?action=show&redirect=OneiricOcelot%2FTechnicalOverview, diakses tanggal 20 April 2012.

[17] "*Hash*cat : User Manual v1.2". (2011). Compiled by radix.