Aplikasi Hash untuk Keamanan Penyimpanan Password

Nixon Andhika 13517059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517059@std.stei.itb.ac.id

***Abstract*—Keamanan dari sebuah database yang menyimpan daftar password pengguna merupakan hal yang sangat penting saat ini akibat frekuensi serangan siber yang semakin meningkat. Salah satu metode paling aman untuk menyimpan password pengguna adalah menggunakan fungsi hash yang dapat melakukan konversi password asli menjadi bentuk yang tidak bisa dibaca langsung oleh manusia. Makalah ini akan membahas penerapan dari fungsi hash untuk meningkatkan keamanan dalam penyimpanan password pengguna.**

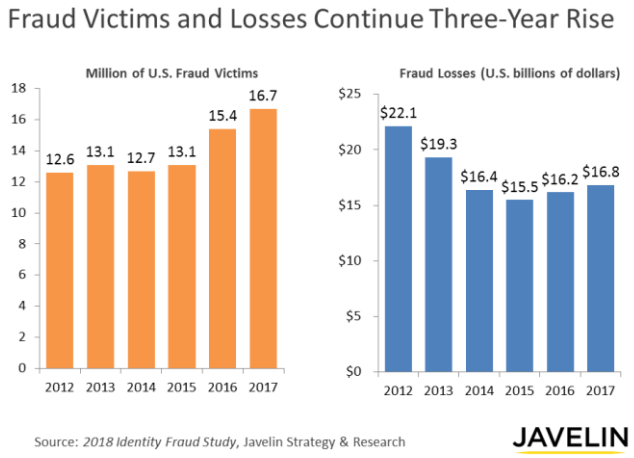
***Keywords*—Hash, keamanan, metode penyimpanan, password.**

I. PENDAHULUAN

Dengan berkembangnya internet yang penggunaannya semakin meningkat, muncul berbagai *platform website* dan aplikasi yang menampung jutaan *account* penggunanya yang masing-masing memiliki informasi pribadi yang harus dijaga kerahasiaannya. Setiap *account* dapat diakses menggunakan *username* dan *password* yang seharusnya hanya diketahui oleh pemiliknya. Jika orang lain mengetahui *password* untuk suatu *account*, maka informasi pribadi dari pemilik aslinya dapat bocor.

Password atau kata sandi merupakan sebuah *string* yang merupakan gabungan dari huruf dan angka yang digunakan untuk memberi seseorang akses kepada suatu perangkat, aplikasi, atau *website*. *Password* dari setiap pengguna sebuah situs atau aplikasi disimpan dalam sebuah *database* dari sebuah *server*. Seorang *hacker* dapat menyerang *database* ini untuk mendapatkan data authentikasi yang dibutuhkan untuk mengakses *account* yang berisi informasi pribadi dari jutaan pengguna. Oleh karena itu, faktor keamanan dari metode penyimpanan data authentikasi, salah satunya *password*, pada sebuah *database* menjadi sangat penting untuk melindungi privasi penggunanya.

Salah satu akibat dari penyimpanan data yang tidak aman adalah *identity fraud*, yaitu pemalsuan identitas dengan menggunakan *account* milik orang lain untuk mendapat keuntungan pribadi. *Identity fraud* menimbulkan kerugian yang sangat besar karena sebagian besar sasarannya adalah *account* finansial. Peningkatan kasus *identity fraud* semakin meningkat dari tahun ke tahun seperti yang terlihat pada statistik oleh New Javelin Strategy & Research Study pada korban *identity fraud* di Amerika Serikat dari tahun 2012 hingga 2017.



Gambar 1. Statistik *Identity Fraud.*
Sumber: New Javelin Strategy & Research Study [1].

Tindakan kriminal seperti *identity fraud* tersebut dapat dicegah dengan meningkatkan faktor keamanan dari penyimpanan data authentikasi. Dengan metode-metode tertentu, keamanan dari penyimpanan data dapat ditingkatkan sehingga lebih *resistant* terhadap serangan siber. Terdapat berbagai metode yang digunakan untuk menyimpan *password* pada sebuah *database* dengan tingkat keamanan yang berbeda-beda. Salah satu metode penyimpanan *password* paling aman adalah dengan menggunakan fungsi *hash* yang dapat mengubah *password* menjadi bentuk lain yang bersifat satu-arah. Satu-arah berarti hasil konversi dari *password* asli tidak dapat didekripsi sehingga walaupun seorang *hacker* mendapat hasil konversi ini, ia tidak akan dapat menggunakannya untuk mengakses informasi dari suatu *account*. Pada makalah ini akan dibahas penggunaan fungsi *hash* dalam metode penyimpanan suatu *password* dan mekanismenya.

II. FUNGSI HASH

A. *Definisi Hash*

Fungsi *hash* atau *hash function* adalah fungsi yang menerima masukan *string* dengan panjang bebas, kemudian mengubahnya menjadi sebuah *string* keluaran dengan

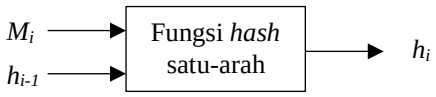
panjang tetap yang umumnya berukuruan lebih kecil daripada *string* semula. Hasil dari fungsi *hash* disebut sebagai *hash value*. Persamaan fungsi *hash* memiliki bentuk di bawah ini.

$$h = H(M)$$

dengan *h* merupakan nilai *hash* atau *hash value* yang didapat dari hasil konversi *string* masukan *M* melalui suatu fungsi *hash H* [2].

Definisi matematis dan skema dari fungsi *hash:*

$$h_i = H(M_i, h_{i-1})$$



Gambar 2. Skema Fungsi Hash.
Sumber: Diktat IF4020 Kriptografi [2].

Karena panjang nilai hasil yang tetap dan ukurannya yang lebih kecil daripada *string* semula, terdapat kemungkinan terjadi kolisi, yaitu kondisi dua *string* sembarang menghasilkan nilai *hash* yang sama. Kolisi dapat dicegah dengan menggunakan beberapa fungsi *hash*, membuat table *overflow* ketika nilai duplikat ditemui, atau menggunakan fungsi *hash* yang menghasilkan ukuran *hash value* yang lebih besar [3].

B. Sifat-sifat Hash

Sifat-sifat fungsi *hash* adalah sebagai berikut [2]:

1. Fungsi *hash* bersifat satu-arah (*one-way function*) yang berarti hasil keluaran tidak dapat dikembalikan menjadi semula (*irreversible*).
2. Fungsi *H* dapat diterapkan pada blok data tanpa batas ukuran.
3. Panjang nilai keluaran dari fungsi *hash* tetap (*fixed length*).
4. Hasil fungsi *hash* mudah dihitung untuk setiap nilai yang dimasukkan.
5. Untuk setiap *h* yang dihasilkan, tidak mungkin dikembalikan nilai *x* sedemikian sehingga *H(x) = h*.
6. Tidak mungkin mencari *x* sedemikian sehingga *H(y) = H(x)*.
7. Tidak mungkin mencari pasangan *x* dan *y* sedemikin sehingga *H(x) = H(y)*.

C. Kegunaan Hash

Terdapat banyak kegunaan dari *hash*. Beberapa aplikasinya adalah sebagai berikut [4].

1. *Message Digest*

Salah satu kegunaan dari fungsi *hash* adalah untuk menciptakan *hash value* dengan panjang dan nilai tetap untuk masukan yang sama. *Hash value* yang dihasilkan oleh suatu *file* sangat peka terhadap perubahan bit pada *file*. Hal ini membuat *hash* dapat mengecek apakah suatu data telah dimodifikasi atau tidak dan juga mengecek integritas suatu data.

2. *Password Verification*

Password disimpan pada server dalam bentuk *hash value* sehingga saat *password* dimasukkan untuk mengakses suatu *platform*, *password* yang dimasukkan diubah terlebih dahulu menjadi *hash value* dan kemudian dibandingkan dengan *hash value* yang ada di *server*. Hal ini dilakukan untuk menjaga keamanan dari *password* dan memastikan *password* dikirim dari *client* ke *server* (tidak terjadi *tampering* oleh pihak ketiga).

3. *Data Structure*

Berbagai Bahasa pemrograman memiliki struktur data yang menggunakan *hash table*. Struktur data ini membentuk pasangan *key-value* dengan *key* memiliki suatu nilai yang unik dan *value* dapat sama untuk *key* yang berbeda. Contoh implementasinya terlihat pada unordered_set & unordered_map di C++, HashSet & HashMap di Java, dict, di Python, dan pada bahasa lainnya.

4. *Compiler Operation*

Keywords dari sebuah bahasa pemrograman diproses dengan cara yang berbeda dari *identifier* lainnya. Saat melakukan kompilasi, *compiler* menggunakan *hash table* untuk membedakan *keywords* tertentu dari sebuah bahasa, seperti if, else, for, return, dan yang lainnya, dengan *identifier* lain.

5. *Rabin-Karp Algorithm*

Algoritma ini merupakan algoritma pencarian *string* yang menggunakan fungsi *hash* untuk menemukan suatu pola tertentu dari *string*. Algoritma ini digunakan untuk mendeteksi plagiarsime.

6. *Linking File name and path*

Dalam pemindahan suatu *file* pada sistem, terdapat dua komponen yaitu nama *file* (*file name*) dan lokasi tujuan file (*file path*). Untuk menyimpan hubungan antara *file name* dan *file path*, sistem menggunakan pemetaan antara keduanya yang diimplementasi menggunakan *hash table*.

D. Algortima Hash Umum

Terdapat dua algoritma *hash* yang umum digunakan, yaitu *Message Digest 5* (MD5) dan *Secure Hash Algorithm* (SHA). Kedua algoritma tersebut umumnya digunakan untuk

memverifikasi *digital signature* dan integritas data dari suatu *file*. Berikut penjelasan untuk kedua algoritma *hash* tersebut [5].

1. *Message Digest 5* (MD5)

MD5 merupakan fungsi *hash* yang menghasilkan 128-bit (16 byte) *hash value*. MD5 digunakan dalam berbagai aplikasi keamanan, dan biasanya digunakan untuk mengecek integritas data. Terdapat kemungkinan terjadinya kolisi saat melakukan *hashing* dengan menggunakan algortima ini sehingga MD5 disebut sebagai algoritma yang tidak *collision resistant*.

Dalam proses untuk menghasilkan *hash value*, pesan masukan dipecah menjadi pecahan berukuran 512-bit, kemudian ditambahkan *padding bits* agar panjangnya dapat dibagi oleh 512. Setelah itu, sebuah blok 128-bit, yang dibagi menjadi 4 pecahan berukuruan 32-bit (A, B, C, dan D), diinisialisasi oleh suatu konstanta 32-bit. Algoritma utama dari MD5 kemudian melakukan operasi kepada masing-masing pecahan 512-bit, yang setiap pecahan tersebut akan memodifikasi kondisi dari algoritmanya. Pemrosesan dari sebuah pesan terdiri dari 4 tahap yang mirip (disebut *rounds*) dan setiap *round* terdiri dari 16 operasi, yang didasari oleh sebuah fungsi non-linear F yang berbeda setiap *round*, penambahan modular, dan *left rotation*.

Terdapat empat kemungkinan fungsi F yang digunakan dalam satu *round*, yaitu fungsi-fungsi yang berada di bawah ini.

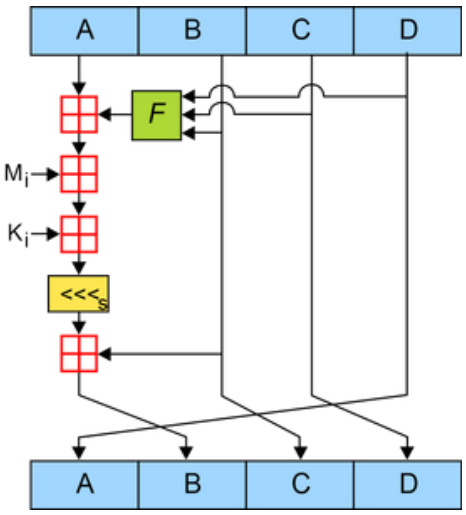
$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z)$$

Pemrosesan satu ronde operasi MD5 dapat dilihat pada gambar di samping.



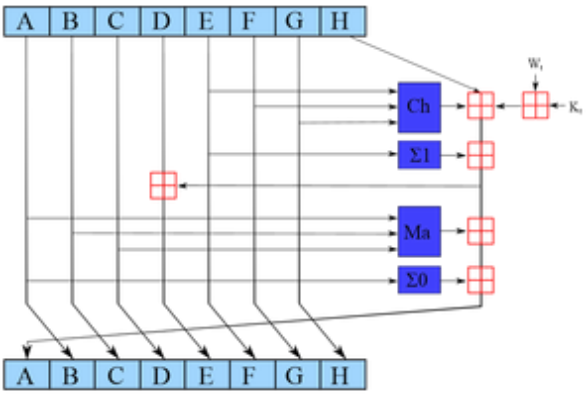
Gambar 3. Skema Satu Ronde Operasi MD5.
Sumber: https://kl2217.wordpress.com/2011/07/21/common-hashing-algorithms/ [5]

2. *Secure Hash Algorithm* (SHA)

Terdapat tiga algoritma SHA yang masing-masing strukturnya berbeda, yaitu *SHA-0*, *SHA-1*, dan *SHA-2*. *SHA-0* dan *SHA-1* tidak *collision resistant* dalam arti terdapat kemungkinan kolisi pada kedua algoritma tersebut. Oleh karena itu, *SHA-2* lebih umum digunakan dan lebih aman karena kemungkinan kolisi yang sangat kecil. *SHA-2* terdiri dari *SHA-224*, *SHA-256*, *SHA-385*, dan *SHA-512* yang menghasilkan *hash value* sebesar nomornya.

Cara kerja SHA-2 mirip dengan MD5, dengan perbedaannya terletak pada kondisi 256-bit (yang dipecah menjadi 8 pecahan 32-bit, yaitu A, B, C, D, E, F, G, dan H), fungsi yang digunakan, dan dilakukan dalam lima *rounds*.

Pemrosesan satu ronde operasi SHA-2 dapat dilihat di bawah ini.



Gambar 4. Skema Satu Ronde Operasi SHA-2.
Sumber: https://kl2217.wordpress.com/2011/07/21/common-hashing-algorithms/ [5]

Komponen biru pada skema tersebut merupakan fungsi-fungsi di bawah ini.

$$Ch\ (E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$Ma\ (A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\sum_0 (A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\sum_1 (E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

III. PASSWORD

A. *Definisi Password*

Password adalah sebuah *string* yang terbentuk dari gabungan berbagai karakter yang digunakan untuk memverifikasi identitas dari pengguna saat proses authentikasi. *Password* biasanya digunakan bersamaan dengan *username* yang keduanya hanya diketahui oleh satu pengguna dan memungkinkan akses pengguna tersebut ke sebuah perangkat, aplikasi, atau *website*. *Password* dapat memiliki panjang yang bervariasi dan dapat mengandung huruf, angka, atau karakter spesial [6].

B. *Strong Password*

Strong password diartikan sebagai *password* yang menolak akses melalui *trial-and-error*. Sebagian besar dari sistem IT mempunyai syarat yang perlu dipenuhi untuk membuat *strong password*, yang membuat pengguna perlu memasukkan *password* yang lebih kompleks dalam upaya mencegah akses yang tidak sah oleh orang lain [7].

Menurut *Administrative Information Technology Services* (AITS) dari *University of Illinois System*, syarat dari sebuah *strong password* adalah sebagai berikut [8].

1. *Password* harus mengandung minimal satu huruf kapital.
2. *Password* harus mengandung minimal satu huruf kecil.
3. *Password* harus mengandung minimal satu angka atau tanda baca.
4. *Password* tidak boleh mengandung kata-kata umum atau nama yang terbentuk dari lima atau lebih karakter.
5. *Password* tidak boleh mengandung karakter *invalid* (spasi, *tabs*, dan lainnya).
6. Panjang *password* harus antara 8-15 karakter.
7. *Password* tidak boleh mengandung pecahan atau kebalikan dari lima atau lebih karakter dari nama depan, tengah, atau belakang.
8. *Password* tidak boleh mengandung pecahan atau kebalikan dari lima atau lebih karakter dari nama perusahaan.
9. *Password* tidak boleh mengandung sekuens maju atau mundur dari lima atau lebih karakter.
10. *Password* tidak boleh mengandung sekuens maju atau mundur dari lima atau lebih angka.
11. *Password* tidak boleh diganti menjadi salah satu dari 12 *password* sebelumnya.
12. *Password* tidak boleh diganti lebih dari satu kali dalam periode 24 jam.
13. *Password* harus diganti secara berkala.

C. *Metode Penyimpanan Password*

Password disimpan pada suatu server dengan berbagai metode dengan beberapa di antaranya lebih aman dibandingkan yang lainnya. Beberapa metode penyimpanan *password* yang paling terkenal adalah sebagai berikut [9].

1. *Plaintext Passwords*

Metode paling sederhana dalam penyimpanan *password* adalah dalam bentuk *plaintext*, yaitu *password* disimpan dalam bentuk yang bisa dibaca oleh orang umum. Penyimpanan *plaintext* tidak menghubah bentuk *password* yang dimasukkan sama sekali. Hal ini berarti terdapat *database* yang berisi *username* dan *password* yang bisa langsung dibaca oleh siapa pun yang mendapat akses ke *database* tersebut. Saat pengguna memasukkan *password*, dilakukan pengecekan langsung terhadap data yang ada di *database* tanpa dilakukan konversi terlebih dahulu.

Metode ini merupakan metode terburuk karena jika seorang *hacker* mendapatkan daftar *username* dan *password* tersebut dari *database*, semua *password* dari pengguna akan langsung bocor.

2. *Basic Password Encryption*

Metode ini menggunakan suatu *encryption key* untuk mengubah *string password* menjadi *string* yang *random* yang tidak bisa dibaca langsung oleh seseorang. Hal ini dilakukan untuk mencegah *hacker* yang mungkin mendapat mendapat daftar *password* dari suatu *database* untuk mengetahui nilai dari *password* yang sebenarnya. Untuk dapat melakukan dekripsi *password* tersebut, ia harus mendapat *encryption key* yang digunakan.

Permasalahan pada metode ini adalah *encryption key* yang digunakan biasanya terletak pada *server* yang sama tempat disimpannya *password*. Sehingga, jika *server* tersebut diserang, kemungkinan seorang *hacker* mendapatkan *encryption key* yang digunakan sangat tinggi. Jika *key* tersebut didapatkan oleh seorang *hacker*, ia hanya perlu melakukan dekripsi dengan membalikkan proses enkripsi.

3. *Hashed Passwords*

Hash berfungsi mirip seperti enkripsi dalam pengertian bahwa *hash* mengubah suatu *string password* menjadi bentuk lain. Perbedaan *hash* dan *encryption*

adalah *hash* bersifat satu-arah sehingga tidak dapat didekripsi walaupun seorang *hacker* mendapatkan fungsi *hash* yang digunakan oleh suatu sistemm.

Metode ini membuat *hacker* tidak dapat melakukan konversi dari *hash value* menjadi *password* asli, tetapi mereka dapat mencoba berbagai *password* hingga mendapat *hash value* yang sama (*brute-force attack*). Komputer dapat melakukan hal ini dengan sangat cepat dan dengan bantuan *rainbow tables* (tabel yang berisi daftar triliunan *hash value* dan *password* yang cocok), mereka dapat mencocokkannya dengan *password* yang telah ditemukan dan menemukan *password* aslinya (*dictionary attack*).

4. *Hashed Passwords with a Dash of Salt*

Salt dalam hal ini diartikan sebagai penambahan sebuah *string random* pada awal atau akhir dari suatu *password* sebelum dimasukkan ke dalam fungsi *hash*. Metode ini menggunakan *salt* yang berbeda untuk setiap *password* sehingga menjadi sangat sulit untuk di-*brute force* dan mempersulit serangan dengan *rainbow table* karena penambahan kompleksitas dan tidak ada *password* yang dapat dicocokkan.

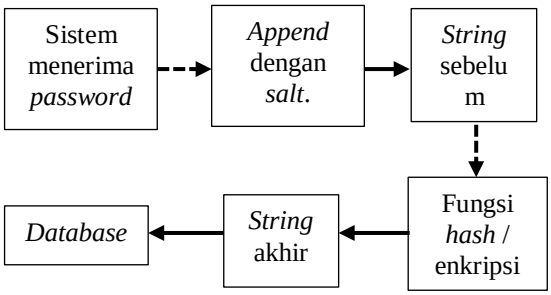
Panjang dari *salt* yang digunakan menentukan tingkat keamanan dari metode ini. *Salt* yang terlalu pendek tidak akan berperan besar dalam mencegah serangan.

IV. MEKANISME PENYIMPANAN PASSWORD DENGAN HASHING

Mekanisme atau alur dari penyimpanan sebuah *password* ke sebuah *database* pada *server* terdiri dari beberapa tahap, yaitu:

1. Sistem menerima *input string password*.
2. Jika diterapkan metode *hash with salt*, dilakukan penambahan *salt*. Penambahan dilakukan dengan *append salt* ke depan atau belakang *string password*.
3. Melakukan enkripsi/*hashing* kepada *password* tergantung metode penyimpanan (jika plaintext, langkah ini tidak dilakukan).
4. Menyimpan *password* ke *database*.

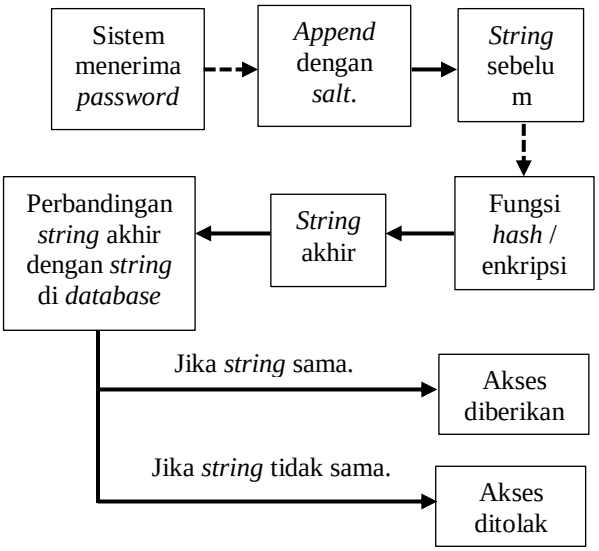
Alur dari penyimpan *password* dapat dilihat pada skema di samping (panah dengan garis putus-putus menandakan tahap tersebut dapat dilewati tergantung metode penyimpanan yang digunakan oleh suatu sistem).



Gambar 5. Skema alur penyimpanan *password*.

Proses authentikasi pengguna mirip dengan mekanisme penyimpanannya, dengan perbedaan terletak di langkah terakhir, yaitu membandingkan dengan nilai yang disimpan pada *database*. Jika nilai akhir dari *password* masukan sama dengan yang disimpan di *database*, maka sistem akan memberi akses kepada pengguna dan jika tidak, maka sistem tidak akan memberi akses.

Proses authentikasi dapat dilihat pada skema di bawah ini.



Gambar 6. Skema alur authentikasi pengguna

Sebagai contoh kasus, akan digunakan sebuah *strong password*, yaitu f4T-p1Nk_gUM. Dengan metode *Hashed Passwords with Salt*, yang digunakan sebagian besar sistem saat ini, *password* tersebut akan di-*append* dengan sebuah *salt* acak, misalnya A2F5WVC2HP62KB0. Hasil dari *string* yang telah ditambahkan *salt* dan sebelum dimasukkan ke fungsi *hash* adalah f4T-p1Nk_gUMA2F5WVC2HP62KB0. *String* tersebut kemudian dimasukkan ke suatu fungsi *hash*, misalnya SHA-512. *Hash value* yang dihasilkan dari fungsi tersebut adalah ECB5E54F009EC0FF374E9C0D2E47D4D300D9EE325C2D0 C6DFA7B90949940B9AB5C7BFFC37A7B18EB02C942886 DDF263233C8B4EAC437C1BCBB1C36D3F596BC2F. *Hash value* ini yang akan disimpan dalam *database.*

Saat seorang pengguna ingin mengakses *account* miliknya, *password* yang dimasukkan akan melalui tahapan-tahapan

proses authentikasi dan kemudian sistem akan membandingkan hasilnya dengan *hash value* yang disimpan di *database*.

Dengan menyimpan *password* dalam bentuk *hash values*, seorang *hacker* akan kesulitan dalam mencoba untuk mendapatkan *password* aslinya. Untuk menemukan *password* asli dari sebuah *hash value* SHA-512 tanpa *salt* secara *brute-force*, dibutuhkan waktu 2^240 * 2^-2 = 2^238 ~ 10^72s ~ 3,17 * 10^64 tahun [10]. Dengan penambahan *salt*, waktu yang dibutuhkan akan bertambah secara eksponensial. Serangan melalui *rainbow table* pun tidak akan mudah karena penambahan *salt* membuat *hash value* menjadi lebih panjang, kompleks, dan unik.

V. KESIMPULAN

Keamanan dari metode penyimpanan *password* sangat penting untuk mencegah serangan siber dan menjaga informasi pribadi dari pengguna. Salah satu metode penyimpanan teraman adalah dengan menggunakan fungsi *hash* yang akan mengubah *string password* masukan menjadi sebuah *hash value*. Fungsi *hash* bersifat satu-arah sehingga *hash value* tidak dapat didekripsi. *Hash value* sangat sulit untuk di *brute-force* dan dengan penambahan *salt*, menjadi sangat sulit untuk diserang walaupun menggunakan *dictionary attack.* Oleh karena itu, penggunaan *hash* untuk menyimpan *password* dan data authentikasi lainnya akan meningkatkan keamanan dari sistem penyimpannya.

VII. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Ibu Dra. Harlili S., M.Sc. selaku dosen pengajar mata kuliah Matematika Diskrit kelas 2 yang telah memberikan ilmu yang digunakan dalam penulisan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada semua penulis dari berbagai referensi yang digunakan pada penulisan makalah ini.

REFERENSI

[1] Javelin. *Identity Fraud Hits All Time High With 16.7 Million U.S. Victims in 2017, According to New Javelin Strategy & Research Study.* Diakses pada 9 Desember 2018 dari https://www.javelinstrategy.com/press-release/identity-fraud-hits-all-time-high-167-million-us-victims-2017-according-new-javelin.

[2] Munir, Rinaldi. 2004. Diktat IF5054 Kriptografi. Bandung: Departemen Teknik Informatika.

[3] Techterms. *Hash*. Diakses pada 8 Desember 2018 dari https://techterms.com/definition/hash.

[4] Thakral, Kushagra. *Applications of Hashing*. Diakses pada 8 Desember 2018 dari https://www.geeksforgeeks.org/applications-of-hashing/.

[5] Anonymous. *Common Hashing Algorithms*. Diakses pada 8 Desember 2018 dari https://kl2217.wordpress.com/2011/07/21/common-hashing-algorithms/.

[6] Rouse, Margaret. *Password.* Diakses pada 8 Desember 2018 dari https://searchsecurity.techtarget.com/definition/password.

[7] Technopedia. *Strong Password*. Diakses pada 8 Desember 2018 dari https://www.techopedia.com/definition/4132/strong-password .

[8] Anonymous. *Strong Password Definition.* Diakses pada 8 Desember 2018 darihttps://www.aits.uillinois.edu/reference_library/i_t_policies/strong_password_definition.

[9] Gordon, Whitson. *How Your Passwords Are Stored on the Internet (and When Your Password Strength Doesn't Matter).* Diakses pada 7 Desember 2018 dari https://lifehacker.com/5919918/how-your-passwords-are-stored-on-the-internet-and-when-your-password-strength-doesnt-matter.

[10] Stack Overflow. *How long to brute force a salted SHA-512 hash? (salt provided).* Diakses pada 9 Desember 2018 dari https://stackoverflow.com/questions/6776050/how-long-to-brute-force-a-salted-sha-512-hash-salt-provided.

PERNYATAAN

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

Bandung, 8 Desember 2018

Nixon Andhika 13517059