Analisis Penggunaan Fungsi Hash BCrypt untuk Keamanan Kata Sandi

Hilmi Naufal Yafie, 13517035
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517035@std.stei.itb.ac.id

Abstrak—Dengan pesatnya perkembangan teknologi saat ini, pertukaran informasi dan data menjadi lebih mudah dilakukan. Untuk menjaga data yang bersifat rahasia, maka digunakan autentikasi pengguna untuk mengetahui pemilik sah dari informasi atau data tersebut. Salah satu cara termudah yang dapat dilakukan adalah dengan menggunakan kata sandi. Namun terjadi permasalahan baru, yaitu pihak yang menyimpan kata sandi harus dapat memastikan kata sandi para pengguna tidak dapat diketahui oleh orang lain. Salah satu cara menjaga keamanan dari kata sandi adalah dengan menggunakan fungsi hash. Dengan fungsi hash, meskipun penyerang berhasil mengambil data yang tersimpan di dalam basis data, maka penyerang akan tetap kesulitan untuk menemukan kata sandi yang sebenarnya, karena sifat dari fungsi hash yang digunakan. Namun, karena ukuran kata sandi yang terbatas dan didukung kecepatan fungsi hash dalam men-*digest* pesan, penyerang dapat memperkirakan jumlah kombinasi yang bisa dilakukan pada kata sandi dan menyerang kata sandi tersebut secara *brute force* hingga didapat kata sandi yang benar. Sehingga, dibutuhkan fungsi hash yang melakukan proses *hashing* selama mungkin untuk mempersulit penyerangan dengan menggunakan *brute force*. Salah satu dari fungsi hash tersebut adalah BCrypt, yang memang diperuntukan untuk melakukan hashing pada kata sandi dan memakan waktu proses *hashing* lebih lama dibanding fungsi hash pada umumnya.

Kata Kunci—Kata Sandi, Fungsi Hash, BCrypt, *Brute Force*.

I. PENDAHULUAN

Di era dunia digital saat ini, pertukaran informasi dan data sangat mudah untuk dilakukan melalui berbagai *platform* di internet. Namun, terdapat beberapa informasi yang cukup penting dan krusial bagi beberapa orang yan juga tersimpan di dalam *platform* tersebut, seperti contohnya informasi terkait keuangan atau data pribadi. Jika informasi ini dapat diketahui oleh orang lain, maka mungkin saja informasi tersebut disalahgunakan untuk kepentingan pribadi. Oleh karena itu, agar hanya si pemilik informasi yang dapat mengakses informasi dan data yang terdapat di suatu *platform* di internet, dilakukan autentikasi untuk memastikan pemilik sah yang dapat mengakses informasi yang disediakan oleh *platform*.

Autentikasi saat ini umumnya dapat dilakukan cukup dengan menggunakan nama atau identitas pengguna beserta kata sandi yang hanya diketahui oleh pengguna. Dalam melakukan autentikasi tersebut, nantinya masukkan nama pengguna dan kata sandi akan dicocokkan dengan pasangan nama pengguna dan kata sandi yang telah tersimpan di dalam basis data *platform*. Namun dengan metode tersebut, pihak penyedia *platform* harus memastikan bahwa pasangan nama pengguna dan kata sandi tersebut dijaga dengan keamanan sehingga tidak dapat diserang oleh orang asing.

Namun, menjaga agar basis data tidak dapat diakses oleh orang asing cukuplah sulit. Maka dari itu, penyimpanan kata sandi pada basis data suatu *platform* tidak disarankan hanya dalam bentuk plainteks, namun perlu dilakukan enkripsi. Salah satu cara pengenkripsian pada kata sandi adalah dengan melakukan *digest* kata sandi dengan fungsi hash. Keuntungan dari penggunaan fungsi hash pada kata sandi adalah meskipun kata sandi yang tersimpan di dalam basis data *platform* berhasil diketahui, kata sandi tersebut tidak mungkin bisa didekripsi oleh orang lain. Sementara itu, untuk melakukan autentikasi, nantinya kata sandi dari masukan pengguna akan di-*digest* dengan fungsi hash, lalu dicocokkan dengan hasil *digest* kata sandi yang telah tersimpan di basis data.

Saat ini, umumnya fungsi hash, seperti MD5 atau SHA256, memiliki kemampuan untuk melakukan *digest* pesan dengan sangat cepat karena sering digunakan sebagai tanda tangan digital pada pesan dengan ukuran yang besar. Untuk ukuran pesan yang besar, *digest* pesan yang cepat bukanlah masalah. Namun, hal ini berbeda untuk kata sandi, dimana panjang kata sandi umumnya singkat dan terbatas, serta jumlah maksimal kombinasi yang dapat dilakukan pada kata sandi bisa diperkirakan. Dengan begitu, kecepatan *digest* pesan dari fungsi hash justru menjadi kelemahan tersendiri, yaitu jika penyerang menggunakan *brute force* untuk melakukan autentikasi, maka dalam waktu singkat autentikasi dapat ditembus berkat kecepatan fungsi hash dalam men-*digest* kata sandi.

Maka dari itu, digunakan fungsi hash khusus yang diperuntukan untuk men-*digest* kata sandi, dimana fungsi hash ini memakan waktu untuk melakukan *digest* kata sandi dan perkembangan perangkat keras tidak akan cukup membantu untuk mempercepat proses digest kata sandi. Fungsi hash ini disebut juga dengan *password hash*. Salah satu jenis dari *password hash* adalah BCrypt.

II. LANDASAN TEORI

A. Keamanan Kata Sandi

Dalam melakukan autentikasi pengguna, terdapat berbagai macam cara yang dapat dilakukan, salah satu cara termudah dan teraman adalah dengan menggunakan kata sandi. Keamanan dari penggunaan kata sandi dari sisi pengguna cukup terjamin karena kata sandi disimpan dalam ingatan pengguna, kecuali jika pengguna secara sengaja maupun tidak membeberkan kata sandinya ke publik. Sehingga, biasanya pihak yang rawan untuk diserang agar kata sandi tersebut bisa didapat untuk disalahgunakan adalah para penyedia layanan *platform* yang menyimpan kata sandir tersebut. Oleh karena itu, pihak penyedia layanan *platform* di internet juga perlu memastikan bahwa mereka telah menerapkan suatu sistem keamanan untuk penyimpanan kata sandi dari para penggunanya.

B. Kriptografi

Kriptografi merupakan sebuah ilmu yang mempelajari bagaimana informasi dapat dirahasiakan dengan baik dari orang yang tidak memiliki hak terhadap informasi tersebut dan tetap dapat memenuhi tujuannya. Dalam kriptografi, umumnya suatu informasi atau pesan akan dibuat menjadi informasi bersifat rahasia (enkripsi) dengan memanfaatkan metode enkripsi yang tersedia. Lalu, nantinya pesan rahasia tersebut akan dikirimkan kepada penerima yang berhak untuk membaca isi pesan tersebut dan si penerima dapat membaca pesan dengan mendekripsi pesan menggunakan metode dekripsi yang sesuai dengan metode enkripsi yang digunakan pada pesan. Dengan mengenkripsi pesan yang dikirim, jika terdapat penyerangan dari pihak luar, seperti menyadap pesan yang dikirim, maka tetap akan mempersulit pihak luar tersebut untuk mengetahui isi sebenarnya dari pesan yang dikirim.

Namun, penggunaan kriptografi juga semakin meluas, tidak hanya untuk merahasiakan suatu pesan yang dikirm, tapi juga dapat untuk hal-hal lain. Salah satunya adalah merahasiakan informasi yang ingin disimpan, dimana isi dari informasi tersebut tidak boleh diketahui oleh orang lain. Berbeda dengan enkripsi pada pesan yang dikirimkan kepada orang lain, pada kasus ini informasi yang dienkripsi justru tidak boleh dapat didekripsi. Salah satu contoh kasus ini adalah *digital signature* dan keamanan kata sandi. Pada *digital signature*, pemilik tanda tangan akan mengirimkan tanda tangannya bersama pesan, namun penerima pesan tidak melakukan dekripsi pada pesan, tetapi mencocokkan *digital signature* yang diterima dengan isi pesan menggunakan metode tertentu. Selain itu, contoh kasus lainnya adalah mengamankan kata sandi dengan menyimpan kata sandi yang telah dienkripsi. Dalam mencocokkan kata sandi, masukan kata sandi dari pengguna akan dienkripsi dengan algoritma yang sama, lalu dicocokkan dengan kata sandi terenkripsi yang telah tersimpan. Hal ini dapat dilakukan dengan memanfaatkan fungsi hash.

C. Fungsi Hash

Fungsi hash merupakan salah satu jenis dari kriptografi dimana suatu pesan dilakukan enkripsi dalam ukuran tertentu. Keunikan dari fungsi hash adalah suatu pesan yang telah dienkripsi dengan fungsi hash tidak mungkin dilakukan dekripsi agar dapat dikembalikan isi dari pesan tersebut. Selain itu, ukuran dari pesan yang dihasilkan oleh fungsi hash akan selalu sama, berapapun ukuran pesan yang dimasukkan. Keunikan dari fungsi hash ini dapat digunakan untuk melakukan autentikasi atau verfikasi suatu informasi yang diterima dengan informasi yang telah tersimpan sebelumnya. Pencocokkan dilakukan dalam bentuk hasil dari fungsi hash.

Fungsi hash sendiri memiliki beberapa sifat tertentu yang mendefinisikan bahwa fungsi hash tersebut bekerja dengan baik yaitu:

- *Collision Resistance*: sifat ini diartikan bahwa akan sulit bagi fungsi hash untuk menghasilkan sebuah nilai yang sama dari dua buah pesan yang berbeda. Sebagai contoh, jika terdapat suatu fungsi hash H(x) dan dua buah pesan a dan b dimana $a \neq b$, maka sangat kecil kemungkinan untuk mendapatkan $H(a)=H(b)$.
- *Preimage Resistance*: sifat ini diartikan bahwa jika terdapat sebuah hasil fungsi hash, akan sangat sulit mencari pesan yang memiliki hasil fungsi hash yang sama. Sebagai contoh, semisal telah terdapat hasil fungsi hash H(x) berupa y, akan sulit untuk menemukan sembarang pesan a dimana $H(a) = y$.
- *Second Preimage Resistance*: sifat ini diartikan bahwa jika terdapat pasangan pesan dan hasil fungsi hash, akan sulit untuk menemukan hasil fungsi hash yang sama dengan pasangan pesan dan hasil fungsi hash yang telah diketahui dari pesan yang berbeda. Sebagai contoh, jika terdapat pasangan pesan a dan hasil fungsi hash H(x) berupa $H(a) = y$, maka akan sulit untuk mencari pesan b dimana hasil dari fungsi hash pada pesan b $H(b) = y = H(a)$.

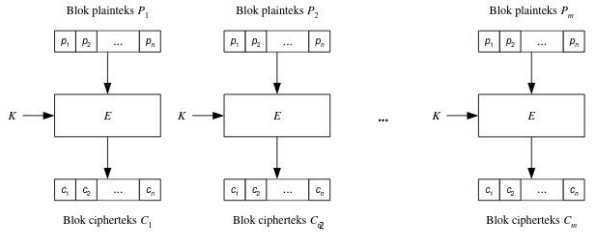
Dengan sifat-sifat tersebut, maka tingkat keamanan dari suatu fungsi hash dapat diketahui berdasarkan apakah sifat tersebut sudah berhasil dipecahkan atau belum.

D. Cipher Blok

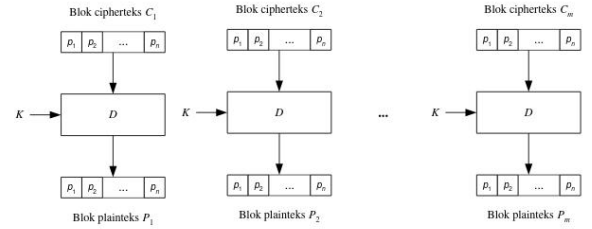
Cipher blok merupakan salah satu bentuk dari kripotgrafi yang menggunakan kunci simetri. Ciri khas dari cipher blok adalah pesan yang akan dienkripsi dibagi menjadi sekumpulan blok dengan panjang bit tertentu. Jika ukuran pesan bukan kelipatan dari ukuran blok yang digunakan, maka setelah pesan dibagi ke dalam blok, pada blok terakhir akan ditambahkan padding untuk menyesuaikan ukuran blok tambah mengubah isi sebenarnya dari isi pesan. Hasil enkripsi menggunakan cipher blok akan memiliki jumlah ukuran yang sama dengan jumlah ukuran blok yang dimiliki. Untuk meningkatkan kompleksitas dari cipher blok, terdapat lima mode yang dapat digunakan pada tahap enkripsi, yaitu ECB (*Electronic Code Book*), CBC (*Cipher Block Chaining*), CFB (*Cipher-Feedback*), OFB (*Output-Feedback*), dan mode Counter.

E. Mode ECB

ECB (*Electronic Code Book*) merupakan salah satu dari mode enkripsi pada cipher blok dimana setiap blok pesan dilakukan enkripsi secara individual dan terpisah, sehingga enkripsi pada satu blok pesan tidak akan mempengaruhi enkripsi yang dilakukan pada blok pesan lainnya.



Gambar 1. Skema enkripsi menggunakan mode ECB



Gambar 2. Skema dekripsi menggunakan mode ECB

Keuntungan dari penggunaan mode ECB adalah setiap blok dienkripsi secara independen, sehingga enkripsi dapat dilakukan tanpa secara sekuensial atau secara acak. Selain itu, penggunaan mode ECB membuat kesalahan enkripsi pada satu blok tidak akan berpengaruh pada blok-blok lainnya.

F. Blowfish Cipher

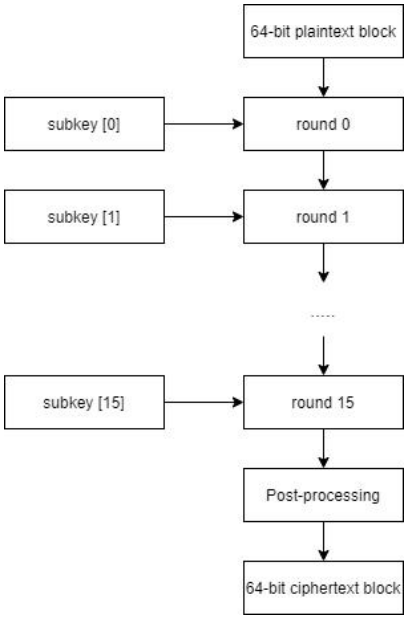
Blowfish cipher merupakan salah satu jenis dari algoritma cipher blok yang dikembangkan oleh Bruce Schneier pada tahun 1993 sebagai alternatif algoritma cipher blok DES. Blowfish cipher menggunakan blok dengan ukuran 64-bit, kunci dengan ukuran bervariasi antara 32-bit hingga 448-bit, subkunci sebanyak 18 buah, putaran sebanyak 16 kali, dan 4 buah substitution box (S-Box).

Untuk tahap pembuatan subkunci, pada awalnya terdapat 18 buah subkey yang nilainya telah diinisialisasi dan disimpan pada sebuah array sebagai berikut.

```
Subkey[0] :243f6a88  Subkey[9] :38d01377
Subkey[1] :85a308d3  Subkey[10]:be5466cf
Subkey[2] :13198a2e  Subkey[11]:34e90c6c
Subkey[3] :03707344  Subkey[12]:c0ac29b7
Subkey[4] :a4093822  Subkey[13]:c97c50dd
Subkey[5] :299f31d0  Subkey[14]:3f84d5b5
Subkey[6] :082efa98  Subkey[15]:b5470917
Subkey[7] :ec4e6c89  Subkey[16]:9216d5d9
Subkey[8] :452821e6  Subkey[17]:8979fb1b
```

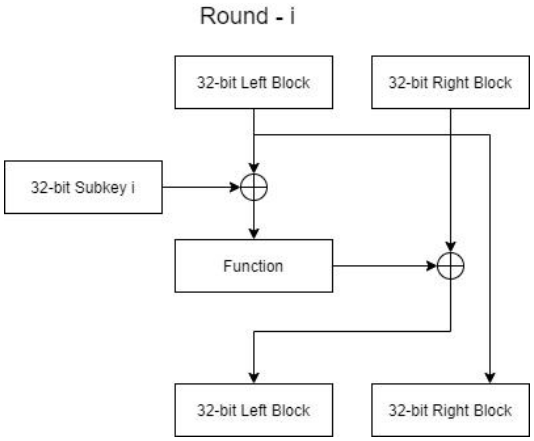
Lalu, setiap subkunci tersebut dilakukan operasi XOR dengan kunci dari input yang dibagi per 32-bit secara sekuensial. Sebagai contoh, subkunci pertama di XOR dengan blok kunci pertama, subkunci kedua di XOR dengan blok kunci kedua, dan seterusnya. Jika jumlah blok kunci kurang dari 18, maka setelah melakukan XOR pada blok kunci terakhir, untuk subkunci berikutnya dilakukan XOR dengan blok kunci pertama. Sebagai contoh, jika kunci sepanjang 64-bit, maka akan didapat dua buah blok kunci. Untuk proses pembentukan kunci, pada subkunci pertama di XOR dengan blok kunci pertama, lalu pada subkunci kedua di XOR dengan blok kunci kedua. Untuk subkunci ketiga, karena jumlah blok kunci hanya dua, maka urutan blok kunci yang dipilih dimulai dari awal lagi, sehingga subkey ketiga di XOR dengan blok kunci pertama. Setelah semua subkunci telah diproses, barulah masuk ke dalam proses enkripsi Blowfish cipher.

Secara keseluruhan, alur dari enkripsi pada blowfish cipher adalah sebagai berikut.



Gambar 3. Skema enkripsi pada Blowfish Cipher

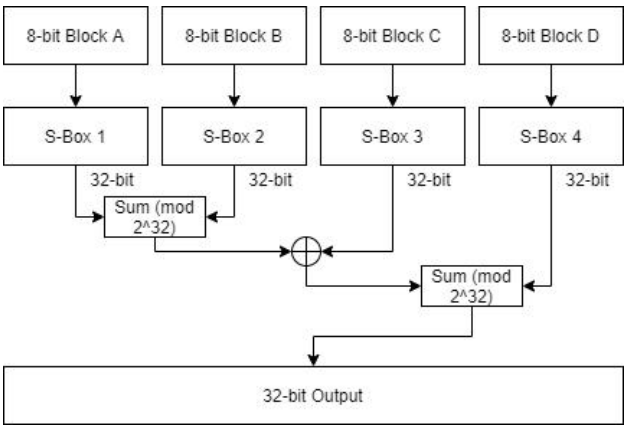
Pada setiap rondenya, enkripsi dilakukan dengan menggunakan jaringan feitsel sebagai berikut.



Gambar 4. Jaringan Feitsel pada Blowfish Cipher

Pertama-tama blok pesan 64-bit dibagi menjadi 32-bit blok kiri dan 32-bit blok kanan. Lalu, pada blok kiri dilakukan operasi XOR dengan subkunci untuk putaran ke-i. setelah itu, hasil XOR dimasukkan ke dalam fungsi. Hasil dari fungsi tersebut lalu di XOR dengan blok kanan dan disimpan sebagai blok kiri yang baru. Sementara untuk blok kanan yang baru menggunakan nilai blok kiri sebelum dilakukan operasi.

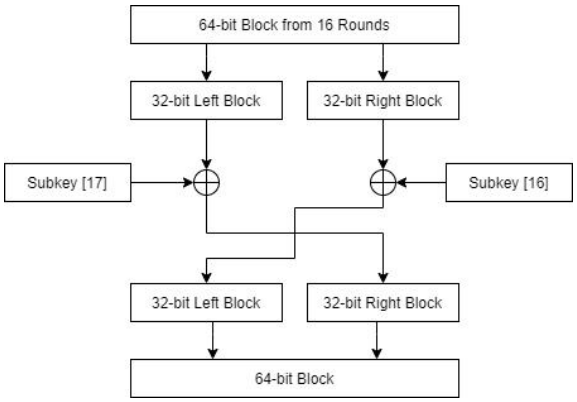
Fungsi yang dijelaskan pada jaringan feitsel dijelaskan sebagai berikut.



Gambar 5. Fungsi pada Jaringan Feitsel

Di dalam fungsi, input blok dengan ukuran 32-bit dibagi secara sekuensial menjadi 4 buah blok yang masing-masing berukuran 8-bit. Lalu, dari dilakukan proses substitusi pada tiap blok dengan S-Box yang telah didefinisikan sebelumnya. Blok 8-bit pertama disubstitusi dengan S-Box pertama, blok 8-bit kedua dengan S-Box kedua, dan seterusnya. Lalu, hasil dari substitusi pada blok pertama dan kedua dijumlahkan dan di modulo dengan 2^{32}. Hasil dari penjumlahan tersebut kemudia di XOR dengan hasil substitusi pada blok ketiga, lalu diteruskan dengan menjumlahkan hasil dengan hasil substitusi blok keempat dan di modulo 2^{32}. Hasil ini kemudian dijadikan output dari fungsi tersebut.

Setelah melalui seluruh ronde, terdapat post-processing pada cipherteks sebelum dikembalikan sebagai output. Post-processing tersebut dilakukan sebagai berikut.



Gambar 6. Post-processing Blowfish Cipher

Hasil blok 64-bit yang telah melewati 16 ronde dibagi menjadi dua blok dengan ukuran masing-masing 32-bit. Pada blok kiri dilakukan operasi XOR dengan subkey ke 17, sementara pada blok kanan dilakukan operasi XOR dengan subkey ke 16. Lalu, hasil XOR pada blok kiri disimpan sebagai blok kanan yang baru, sementara hasil XOR pada blok kanan disimpan sebagai blok kiri yang baru, lalu kedua blok digabungkan kembali menjadi output akhir dari enkripsi.

G. BCrypt

BCrypt merupakan sebuah fungsi hash yang dibuat oleh [3] dengan berdasarkan Blowfish cipher. Penamaan BCrypt terdiri dari B untuk Blowfish dan Crypt yang merupakan nama fungsi hash yang digunakan pada sistem kata sandi di UNIX. Crypt pada awal pengembangannya di tahun 1976 hanya dapat melakukan hash maksima untuk empat buah kata sandi setiap detiknya, hal ini yang membuat fungsi hash Crypt cukup kuat pada waktunya. Namun, seiring dengan pesatnya perkembangan teknologi, sebuah komputer saat ini dengan software dan hardware yang telah dioptimalisasi dapat melakukan hashing dengan Crypt sebanyak 200.000 kata sandi per detik. Maka dari itu, fungsi hash Crypt sendiri saat ini sudah tidak cocok jika digunakan untuk menjaga keamanan kata sandi.

Pada BCrypt, fungsi hash Crypt dikombinasikan dengan Blowfish cipher agar waktu yang dibutuhkan untuk melakukan hashing menjadi lebih lama. Blowfish cipher sendiri sebenarnya merupakan block cipher yang cukup cepat, kecuali saat melakukan pergantian kunci. Pada Blowfish cipher, setiap kali pembuatan kunci yang baru membutuhkan waktu pemrosesan awal yang setara dengan waktu yang dibutuhkan untuk melakukan enkripsi pada teks berukuran empat kilobytes. Lamanya waktu proses yang dibutuhkan untuk membuat kunci baru pada Blowfish cipher inilah yang dimanfaatkan sebagai kebutuhan komputasi tambahan dalam BCrypt. Sehingga, serangan brute force pada kata sandi yang menggunakan fungsi hash BCrypt dapat diperlambat.

Pada prosesnya, BCrypt memiliki tahap inisialisasi kunci yang dikembangkan dari tahap inisialisasi kunci pada Blowfish cipher, yang diberi nama “eksblowfish”, kepanjangan dari “expensive key schedule Blowfish”. Selain itu, BCrypt juga secara default menggunakan 128-bit salt pada proses hashingnya untuk mencegah serangan rainbow table. Proses BCrypt sendiri terbagi menjadi dua tahap. Tahap pertama adalah menjalankan inisialisasi kunci eksblowfish dengan parameter berupa cost yang diinginkan, nilai salt, dan kata sandi yang akan di hashing. Pada tahap ini, BCrypt akan melakukan melakukan penurunan kunci, dimana sekumpulan subkunci diturunkan dari sebuah kunci utama, dengan kunci utama diisi oleh kata sandi. Jika kata sandi yang digunakan terlalu pendek, pada tahap ini nantinya akan dibuat menjadi kunci yang lebih panjang. Sehingga, tahap pertama juga dapat dikatakan melakukan penguatan pada kunci. Lalu, pada tahap kedua dilakukan enkripsi pada sebuah magic value dengan ukuran 192-bit “OrpheanBeholderScryDoubt” menggunakan kunci yang telah dibuat pada tahap pertama dan dilakukan iterasi sebanyak 64 kali. Hasil akhir dari tahap ini adalah hasil enkripsi menggunakan mode ECB yang digabungkan dengan cost serta nilai salt yang berukuran 128-bit.

Implementasi dari BCrypt digambarkan pada pseudo-code sebagai berikut:

```
BCrypt (cost, salt, password)
 key ← EksblowfishSetup(cost, salt, password)
 ciphertext ← “OrpheanBeholderScryDoubt” #magic
value
 repeat(64)
   ciphertext ← EncryptECB(ciphertext, key)
 return Concatenate(cost, salt, ciphertext)
```

Hasil dari BCrypt sendiri memiliki prefix seperti “$2a$” atau “$2y$” yang berguna untuk memberitahu fungsi hash yang BCrypt beserta versinya yang digunakan untuk mendapatkan hasil tersebut.

III. Eksperimen Fungsi Hash

A. Langkah Eksperimen

Eksperimen pengujian fungsi hash dilakukan dengan melakukan hashing pada beberapa fungsi hash dengan menggunakan sebuah kata sandi yang sama. Nantinya dilakukan pengukuran untuk melihat seberapa banyak hashing dapat dilakukan oleh fungsi hash dalam satuan detik. Pada kasus ini, semakin sedikit jumlah hashing yang dapat dilakukan oleh fungsi hash per detik berarti fungsi hash tersebut juga semakin aman dari serangan brute force. Pada eksperimen ini tidak dilakukan tahap implementasi untuk setiap fungsi hash maupun pengukurannya, namun menggunakan website pengujian yang telah menyediakan implementasi tersebut. Website tersebut dapat diakses pada laman berikut (https://asecuritysite.com/encryption/htest). Sementara itu, fungsi hash yang digunakan pada eksperimen adalah sebagai berikut:

1. SHA-1
2. SHA-256
3. SHA-512
4. MD5
5. DES
6. BCrypt
7. APR1
8. PBKDF2 (Password-Based Key Derivation Function 2) dengan SHA-1
9. PBKDF2 (Password-Based Key Derivation Function 2) dengan SHA-256
10. LM (LAN Manager) Hash
11. NTLM (New Technology Lan Manager) Hash
12. MS DCC
13. Oracle 10
14. Cisco PIX
15. Cisco Type 7
16. MS SQL 2000
17. MySQL
18. Postgres (MD5)
19. LDAP (MD5)
20. LDAP (SHA1)

B. Tabel Hasil Eksperimen

Berikut adalah tabel hasil ekperimen terhadap 20 fungsi hash yang menampilkan jumlah hash per detik dan hasil nilai fungsi hashnya. Kata yang digunakan oleh setiap fungsi hash untuk melakukan hashing adalah “password123”.

Tabel 1. Hasil Pengujian Fungsi Hash

Metode	Hash / sec	Hash value
SHA-1	465116	cbfdac6008f9cab4083784cbd1874f76618d2a97
SHA-256	418410	ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
SHA-512	223214	bed4efa1d4fdbd954bd3705d6a2a78270ec9a52ecfbfb010c61862af5c76af1761ffeb1aef6aca1bf5d02b3781aa854fabd2b69c790de74e17ecfec3cb6ac4bf
MD5	478468	482c811da5d5b4bc6d497ffa98491e38
DES	329	ZD3yxA4N/XZVg
BCrypt	**265**	$2a$05$.OSesZjz0y92ZQKbWpivHuEUQGSBuJ.03oE3RnCjKm.IsqBeBf.Tm
APR1	557	$apr1$ZDzPE45C$uOxTdFn0zTGjCWvED0cb./
PBKDF2 SHA-1	4810	$pbkdf2$5$WkR6UEU0NUM$HHvu.pow2ps8KP2JlrIfeWW0yYY
PBKDF2 SHA-256	12030	$pbkdf2-sha256$5$WkR6UEU0NUM$K5gUHZtX5hvNJ6hsQJmMnehBZjwfU/la18KWoyLiURc
LM Hash	1708	e52cac67419a9a22664345140a852f61
NTLM Hash	29312	a9fdfa038c4b75ebc76dc855dd74f0da
MS DCC	33057	cd0197ff7857f762b82ff63ecb0e4d7f
Oracle 10	390	1F289F5811BAC195
Cisco PIX	14347	WmRamiZ6ldepwhIN
Cisco Type 7	23571	01030717481C091D251D1C5A
MS SQL 2000	15598	0x0100AE758E7182CD65C7D119B28FC12992CF50581D50DD2B50E3C21D631E39F060B9C1A0545A3F9252DCFD152BA8
MySQL	28628	*A0F874BC7F54EE086FCE60A37CE7887D8B31086B
Postgres MD5	25859	md5db4724c23b37dd94d8dec611e0bac701
LDAP MD5	15260	{MD5}482c811da5d5b4bc6d497ffa98491e38
LDAP SHA-1	49912	{SHA}cbfdac6008f9cab4083784cbd1874f76618d2a97

IV. Analisis Hasil Eksperimen

Untuk menentukan apakah suatu fungsi hash cukup baik untuk digunakan sebagai hashing pada kata sandi, maka jumlah hash per detik yang dilakukan oleh fungsi hash haruslah sekecil mungkin. Semakin kecil nilai hash per detik, maka akan semakin lama waktu yang dibutuhkan untuk melakukan brute force pada fungsi hash tersebut. Berdasarkan hasil dari eksperimen yang dilakukan pada Bab III untuk 20 fungsi hash, didapatkan bahwa BCrypt mendapatkan nilai terendah, yaitu 265 hash per detik. Selain itu, terdapat juga beberapa fungsi hash lainnya yang juga memiliki nilai yang rendah, seperti DES dengan 329 hash per detik, Oracle 10 dengan 390 hash per detik, dan APR1 dengan 557 hash per detik. Beberapa fungsi hash tersebut bisa dikatakan cocok untuk dijadikan fungsi hash untuk kata sandi karena sulit diserang dari sisi brute force. Sementara untuk penyerangan dari sisi lain perlu dilakukan pengujian lebih mendalam.

Sementara itu, kebanyakan fungsi hash yang diuji kurang cocok jika ingin dijadikan fungsi hash untuk menjaga keamanan kata sandi dilihat dari sisi penyerangan brute force, karena nilai yang dihasilkan lebih dari 10000, seperti PBKDF2

(SHA-256) dengan nilai 12030 hash per detik, NTLM Hash dengan nilai 29312 hash per detik, MS DCC dengan nilai 33057 hash per detik, Cisco PIX dengan nilai 14347 hash per detik, Cisco Type 7 dengan nilai 23571 hash per detik, MS SQL 2020 dengan nilai 15598 hash per detik, MySQL dengan nilai 28628 hash per detik, Postgres (MD5) dengan nilai 25859 hash per detik, LDAP (MD5) dengan nilai 15260 hash per detik, dan LDAP (SHA-1) dengan nilai 49912 hash per detik. Bahkan, terdapat beberapa fungsi hash yang memiliki nilai diatas 100000, yaitu SHA-1 dengan nilai 465116 hash per detik, SHA-256 dengan nilai 418410 hash per detik, SHA-512 dengan nilai 223214 hash per detik, dan MD5 dengan nilai 478468 hash per detik.

Meskipun kurang cocok untuk digunakan pada hashing kata sandi, fungsi hash yang memiliki nilai hash per detik yang tinggi akan sangat cocok digunakan pada pesan yang ukurannya cukup besar, karena dengan begitu waktu hashing yang dibutuhkan untuk mendigest pesan tersebut menjadi lebih cepat.

V. KESIMPULAN

Dari hasil eksperimen yang telah dilakukan, didapatkan bahwa BCrypt mendapatkan nilai hash per detik terkecil diantara 20 fungsi hash yang diuji, yaitu 265 hash per detik. Maka dari itu, dapat dikatakan bahwa BCrypt memiliki keamanan terbaik diantara 20 fungsi hash yang duji dalam mencegah penyerangan brute force pada kata sandi yang di hashing. Selain BCrypt, terdapat pula beberapa fungsi hash lain yang dapat digunakan sebagai fungsi hash alternatif untuk menjaga keamanan kata sandi dari penyerangan brute force, seperti DES, Oracle 10, dan APR1.

VI. KATA PENGANTAR

Pertama-tama saya ingin mengucapkan rasa puji dan syukur kehadirat Allah SWT. karena berkat rahmat dan karunia-Nya makalah ini dapat diselesaikan. Lalu, saya juga ingin mengucapkan rasa terima kasih kepada Pak Rinaldi Munir sebagai dosen mata kuliah IF4020 Kriptografi atas ilmu yang telah diajarkan sehingga dapat diterapkan dalam pembuatan makalah ini. Dan juga, saya ingin mengucapkan rasa terima kasih kepada kedua orang tua saya atas dukungan dan do'anya selama pengerjaan makalah ini.

REFERENSI

[1] Munir, Rinaldi. (2020). Kriptografi Modern (Bagian 3: Block Cipher). Slide Bahan Kuliah IF4020 Kriptografi.

[2] Munir, Rinaldi. (2020). Fungsi Hash. Slide Bahan Kuliah IF4020 Kriptografi.

[3] Boterhoven, Daniel. (2016). *Why you should use BCrypt to hash password.* https://danboterhoven.medium.com/why-you-should-use-bcrypt-to-hash-passwords-af330100b861

[4] Provos, Niels & Mazières, David. (1999). *A future-adaptive password scheme*. 32-32.

[5] Malvoni, Katja & Designer, Solar & Knezovic, Josip. (2014). *Are Your Passwords Safe: Energy-Efficient Bcrypt Cracking with Low-Cost Parallel Hardware*. 10.13140/2.1.3267.0081.

[6] LuisPa. (2019). *How To Safely Store A Password.* https://dev.to/luispa/how-to-safely-store-a-password-5fao

[7] Bhat, Abhay. (2019). *Blowfish Algorithm with Examples*. https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/

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.

Balikpapan, 20 Desember 2020

Hilmi Naufal Yafie, 13517035