Implementasi dan Pengujian Protokol Autentikasi Pengguna pada Flask dengan Enkripsi STANDARD, Keccak dan Garam

Muhammad Rizki Duwinanto - 13515006
Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515006@std.stei.itb.ac.id

***Abstract*—Protokol Autentikasi saat ini rentan dengan serangan *bruteforce* berupa serangan pelangi untuk melakukan hashing, maka saat ini dibutuhkan penyimpanan data yang baik untuk melakukan manajemen pengguna. Dibuat protokol untuk lebih merumitkan proses untuk serangan pelangi tersebut dengan menggunakan Hash Keccak, Algoritma STANDARD dan garam yang mengenkripsi password. Protokol tersebut terdiri protokol *login* dan *registrasi*. Protokol registrasi melakukan enkripsi dua kali dan menyimpan *hash* dari *password* ditambah garam dengan kunci garam. Protokol *login* melakukan dekripsi dari *ciphertext* kemudian disamakan dengan hash dari *password* dan garam. Hasil dari penelitian ini adalah sebuah protokol yang tidak rentan terhadap *bruteforce*, pengubahan *password* dan memiliki efisiensi yang tinggi.**

Keywords-Protokol, Keccak, STANDARD, Salt

I. PENDAHULUAN

Seiring perkembangan zaman, hal yang juga sangat terasa perkembangannya adalah teknologi informasi. Kebutuhan akan pertukaran informasi semakin hari semakin bertambah. Tentunya pertukaran informasi harus didukung juga dengan keamanan akan autentikasi di dalamnya. Sebenarnya sejak dahulu kala, sudah bermunculan berbagai macam metode untuk mengamankan data yang dipertukarkan, hal ini yang nantinya menjadi cikal bakal dari kriptografi.

Kriptografi memiliki sifat yaitu untuk mencapai kerahasiaan, integritas data, autentikasi, dan *Non-Repudiation*. Autentikasi sangat dibutuhkan dalam kebutuhan sehari-hari untuk *login* dan kebutuhan lain. Autentikasi merupakan dasar dari aplikasi untuk dapat digunakan secara nyaman dan aman tanpa gangguan dari pihak yang merugikan. Autentikasi yang baik adalah awal dari penjaminan informasi terhadap pengguna.

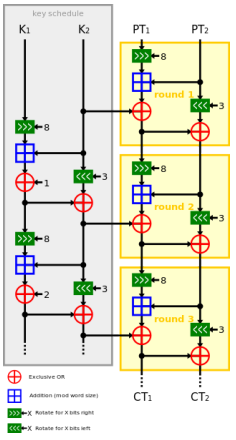
Saat ini Autentikasi sangat lemah dan dapat dipecahkan dengan cepat oleh proses *bruteforce* dikarenakan kekuatan komputasi yang semakin cepat. Kemudian terdapat banyak penyalahgunaan data yang disimpan dalam basis data yang sangat signifikan.

Dibutuhkan protokol yang dapat mengenkripsi *password* dengan garam yang tidak rentan terhadap serangan *bruteforce,* pengubahan dan tetap efisien. Maka dari itu dibutuhkan penelitian apakah protokol dengan algoritma STANDARD, *hashing* dengan garam dapat memperbaiki protokol yang ada.

II. DASAR TEORI

A. Algoritma Block Cipher STANDARD

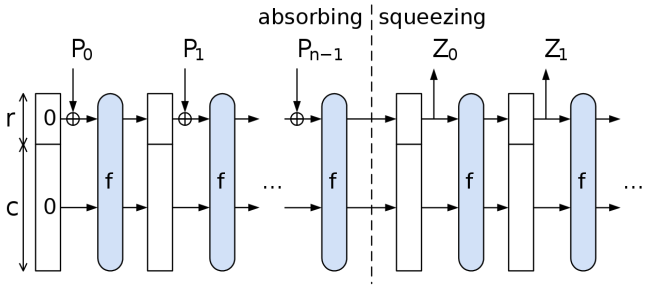
Algoritma STANDARD adalah algoritma *block-cipher* yang dibuat menerima *block cipher* sepanjang 256 bit atau 32 byte dan kunci dengan ukuran yang sama. Dipilih jumlah bit tersebut dikarenakan semakin banyak bit yang diproses oleh algoritma, maka akan semakin rumit proses kriptografi yang ada sehingga mempersulit serangan terhadap algoritma. Algoritma ini juga diulang sebanyak 20 kali untuk meningkatkan kompleksitas. Algoritma ini menerapkan prinsip *diffusion* dan *confusion* dengan memproses algoritma tersebut dengan permutasi dengan kotak permutasi dan substitusi dengan kotak substitusi pada fungsi *round*-nya. Fungsi *round* tersebut kemudian juga diproses dengan operasi bit agar semakin kompleks. Algoritma ini juga menerapkan jaringan Feistel dengan membagi kedua *plaintext* menjadi 128 bit di keduanya.



Gambar 1. Contoh *Block Cipher* (Sumber : https://en.wikipedia.org/wiki/File:Speck_block_cipher.svg)

B. Keccak

Fungsi Hash merupakan suatu fungsi yang merubah suatu pesan dengan panjang sembarang menjadi suatu pesan ringkas yang panjangnya selalu tetap meskipun panjang pesan aslinya berbeda-beda. Hash merupakan salah satu bentuk teknik kriptografi dan dikategorikan sebagai kriptografi tanpa kunci. Fungsi hash merupakan fungsi yang bekerja dalam satu arah, sehingga pesan yang diubah menjadi message-digest tidak dapat dikembalikan menjadi pesan semula (irreversible).



Gambar 2. Konstruksi Spons *Hash Keccak* (Sumber : https://commons.wikimedia.org/wiki/File:SpongeConstruction.svg)

Fungsi Hash Keccak merupakan pemenang dari kompetisi terbuka untuk SHA-3 yang diadakan oleh NIST. Keccak berbeda dari finalis SHA-3 lainnya dalam hal menggunakan konstruksi 'spons' (sponge construction). Jika desain lainnya bergantung pada 'fungsi kompresi, Keccak menggunakan fungsi non-kompresi untuk menyerap dan kemudian membuat digest. Fungsi ini dapat mengeluarkan byte dari 224 hingga 512.

C. Protokol Kriptografi

Protokol adalah aturan yang berisi rangkaian langkah-langkah, yang melibatkan dua atau lebih orang, yang dibuat untuk menyelesaikan suatu kegiatan. Protokol kriptografi adalah protokol yang menggunakan kriptografi untuk kerahasiaan, integritas, otentikasi, maupun nirpenyangkalan. Setiap protokol kriptografi pasti melibatkan dua atau lebih pihak yang dapat berupa orang, mesin, sistem, dll. Protokol ini dibangun dengan melibatkan beberapa algoritma atau skema kriptografi. Fungsi protokol bisa bermacam-macam. Berbagi pesan rahasia, pertukaran kunci, otentikasi identitas adalah contoh-contoh protokol yang membutuhkan kriptografi.

III. Rancangan Solusi

Dibuat sebuah aplikasi *web* yang menerima *username* dengan *password,* harapannya *username* dapat dikembangkan menjadi info-info yang lain. Dibuat dua algoritma untuk registrasi dan *login*. Algoritma tersebut dibuat agar memperbaiki proses enkripsi password yang hanya mengandalkan fungsi *hash*.

Rancangan algoritma yang dibuat adalah, saat proses registrasi, agar dapat memperkuat proses autentikasi digunakan algoritma *block-cipher* STANDARD untuk mengenkripsi *hash* Keccak dari *username* dan sepuluh huruf acak dari pembangkit bilangan acak dengan kunci yaitu hash dengan mode *electronic code book.* Hash *keccak* atau SHA-3 yang dihasilkan dari *ciphertext block-cipher* tersebut menjadi garam.

Kemudian *hash* dari *password* dan salt dilakukan enkripsi dengan kunci *hash* dari *salt* yang dihasilkan dengan algoritma *block cipher* yang sama menghasilkan *ciphertext* dari *hash* dari *password* dan *salt*. *Username*, *ciphertext* yang baru saja dihasilkan dari dan garam akan disimpan pada basis data.

Berikut adalah protokol *registrasi* yang akan dilakukan.

```
Protokol Registrasi:
   1. Username dan Password dimasukkan
      oleh user.
   2. Password menjadi key dan username
      ditambah dengan hasil pembangkitan
      acak dari alphabet menjadi
      plaintext.
   3. Dilakukan enkripsi block-cipher
      dari key dan plaintext.
   4. Garam dihasilkan dari Hash dari
      ciphertext.
   5. Hash dari garam menjadi key dan
      hash dari password dan garam
      menjadi plaintext.
   6. Dilakukan enkripsi block-cipher
      dari key dan plaintext.
   7. Username, Ciphertext yang
      dihasilkan pada proses 7 dan garam
      dimasukkan kedalam basis data.
```

Proses dalam melakukan *login* dilakukan dengan melakukan dekripsi dari kolom *password* basis data dengan kunci yaitu garam. Kemudian hasil dari *ciphertext* dilakukan penyamaan apakah sama dengan *hash* dari *password* ditambah garam. Hal ini berguna untuk menyembunyikan informasi *password* jika basis data bocor.

Berikut adalah protokol *login* yang akan dilakukan.

```
Protokol Login:
   1. User memasukan username dan
      password.
   2. Mendapatkan informasi dengan
      username dari basis data yang ada.
   3. Mendapatkan informasi salt dan
      password.
```

4. Password menjadi ciphertext, garam menjadi kunci.
5. Dilakukan dekripsi block-cipher dari key dan ciphertext.
6. Melakukan verifikasi pengguna dari hasil plaintext dengan hash dari pengguna dan garam

IV. Hasil Implementasi

Hasil implementasi adalah dua buah *endpoint* API yang dibuat dengan bahasa Python. *Endpoint* API tersebut bernama /login dan /register. Berikut adalah hasil implementasi dari endpoint register yang sudah dibuat sistem manajemen pengguna yang sudah dibuat.

```
@app.route("/register",
methods=['GET','POST'])
def register():
  if request.method == 'POST':
    username = request.form['username']
    password = request.form['password']

    mode =
Modes.ECB(str.encode(hash(password)))
    bytes_processed =
mode.encrypt(bytearray(str.encode(hash(usern
ame + randomword(len(password),
len(password))))))
    salt = hash(bytes_processed.hex())

    mode = Modes.ECB(str.encode(hash(salt)))
    password_processed =
mode.encrypt(bytearray(str.encode(hash(passw
ord+salt))))

    user_dict = { "username" : username,
                  "password" :
password_processed.hex(),
                  "salt"     : salt}

    user_id =
user.insert_one(user_dict).inserted_id

    return
jsonify(status=200,message='Registered',user
_id=str(user_id))
```

Berikut adalah hasil implementasi dari endpoint login yang sudah dibuat sistem manajemen pengguna yang sudah dibuat.

```
@app.route("/login", methods=['GET','POST'])
def login():
  if request.method == 'POST':
    username = request.form['username']
    password = request.form['password']

    query = { "username" : username }

    user_information = user.find(query)

    infos_user = []
    for infos in user_information:
      id = str(infos.pop('_id'))
      infos['id'] = id
      infos_user.append(infos)

    mode =
Modes.ECB(str.encode(hash(infos_user[0]['sal
t'])))
    password_processed =
mode.decrypt(bytearray.fromhex(infos_user[0]
['password']))

    str_password =
password_processed.decode('utf-8')
    hash_p = hash(password +
infos_user[0]['salt'])

    if str_password[:len(hash_p)] == hash_p:
      return
jsonify(status=200,message='login')
    else:
      return jsonify(status=500,message='Not
matched')
```

Basis data yang digunakan dalam penelitian ini adalah MongoDB. Basis data tersebut bersifat NoSQL. Berikut adalah hasil penyimpanan data dari basis data tersebut.

```
{
      "_id" :
ObjectId("5cd51551d23480552fe1201c"),
      "username" : "rizkiduwinanto",
      "password" :
"367a317143243b4b0731312a74d158f0333f3844
42447c2682af60af368543c34339453e305157a02
12540c832e72e033841197b4533c32ca08d248041
e20857",
      "salt" :
"8816EE048A0B6D6616D9852F31EB8E4443A56974
935E621D29897EF1"
}
```

V. Pengujian dan Analisis

A. Kecepatan dan Efisiensi

Kecepatan dari algoritma untuk melakukan enkripsi dan *hashing* untuk fitur registrasi adalah sebagai berikut.

No.	Ukuran Password	Waktu
1.	5 byte	13 ms
2.	10 byte	14 ms
3.	20 byte	13 ms
4.	50 byte	13 ms
5.	100 byte	13 ms

Kecepatan dari algoritma untuk melakukan enkripsi dan *hashing* untuk fitur *login* adalah sebagai berikut.

No.	Ukuran Password	Waktu
1.	5 byte	9 ms
2.	10 byte	6 ms
3.	20 byte	7 ms
4.	50 byte	7 ms
5.	100 byte	7 ms

Dapat disimpulkan bahwa kecepatan algoritma *login* dan registrasi tidak berbanding lurus dengan waktu algoritma dan sangat efisien. Hal ini dapat diakibatkan fungsi Keccak mengubah data *password* dll menjadi lebih kecil dan seragam yaitu 224 byte. Sehingga dapat disimpulkan bahwa efisiensi algoritma ini adalah O(1).

B. Pengubahan Password

Jika dilakukan pengubahan pada *password* misal:

```
5250c7b3725432f7c4272553582531754043c6139a3019
30c12c282b2721330f15a4c317462138b00b5b42115543
014902a7b742e3353730800a010920000004
```

menjadi *password* dengan *string* lain.

```
5250c7b3725432f7c4272553582531754043c6139a3019
30c12c282b2721330f15a4c317462138b00b5b42115543
014902a7b742e3353730800a010920000005
```

Menunjukan bahwa *password* jika diubah akan membuat pengguna tersebut tidak bisa melakukan *login* sehingga menunjukan keamanan dari *algoritma block cipher* dan *hash.*

C. *Bruteforce*

Ukuran password yang digunakan juga sama yaitu 224 byte. Dibutuhkan 2 ^ 224 atau 2.7x10^67 percobaan dan karena terdapat dua algoritma dibutuhkan hampir dua kali untuk mencapai password yang diinginkan atau 5,4X10^67. Kemungkinan sebanyak itu apabila dilakukan brute force oleh super komputer tercepat saat ini saja, Sunway TaihuLight milik Tiongkok yang berkecepatan 93 petaflops (93x10 15 operasi per detik), masih membutuhkan waktu sangat lama untuk memecahkan algoritma ini dengan brute force.

VI. Simpulan dan Saran

Simpulan dari makalah ini adalah dihasilkan protokol *login* dan *user* yang dapat melakukan enkripsi dan *hashing*. Protokol ini melakukan improvisasi dari protokol sebelumnya. Dari protokol yang sudah dibuat, dapat dipastikan bahwa protokol ini tidak rentan terhadap serangan *bruteforce,* memiliki efisiensi yang baik yaitu O(1), tidak dapat dilakukan pengubahan menunjukan keamanan protokol ini sangat baik.

Saran untuk selanjutnya adalah lebih melakukan operasi yang kompleks dalam melakukan autentikasi *password, username*, menambahkan kolom lain dan menggunakan protokol *secret sharing*.

REFERENSI

[1] http://www.wikiwand.com/en/Block_cipher_mode_of_operation

[2] Munir, Rinaldi, 2005. Diktat Kuliah Kriptografi. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung.

[3] Menezes, Alfred J, 2001, Handbook of Applied Cryptography (Fifth ed.), Waterloo: CRC Press. hal. 251

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, 10 Mei 2019

Muhammad Rizki Duwinanto
(13515006)