Implementasi serta Analisis Fungsi *Hash* sebagai *Authenticator* dalam Alat IoT

T. Antra Oksidian Tafly 13517020
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517020@std.stei.itb.ac.id

***Abstrak*—Teknologi saat ini tidak hanya berada di dalam komputer atau telepon genggam saja. Teknologi juga berada dalam alat kehidupan sehari-hari seperti lampu, tirai dan lainnya yang disebut sebagai Internet of Things (IoT). Alat IoT ini di desain untuk berkoneksi dengan cepat ke alat lain tanpa adanya prosedur autentikasi. Oleh karena itu, perlu dibuat sebuah algoritma autentikasi pengguna yang dapat mendukung kapabilitas alat IoT yang kecil dan lemah. Pada makalah ini, beberapa fungsi *hash* SHA-1, SHA-2, dan SHA-3 akan diimplementasikan ke sebuah simulasi alat IoT sebagai *authenticator*. Kinerja autentikasi dari semua fungsi *hash* kemudian akan dibandingkan.**

***Kata Kunci—Internet of Things*, fungsi *hash*, SHA-1, SHA-2, SHA-3, autentikasi.**

I. Pendahuluan

Teknologi saat ini sudah berkembang pesat sehingga teknologi hadir tak hanya di komputer atau telepon genggam, namun juga hadir di alat kehidupan sehari-hari seperti lampu, tirai, kamera CCTV, *ear buds*, dsb. Karena sifatnya, alat-alat tersebut haruslah tidak memakan banyak tempat, tidak membutuhkan banyak energi, dan juga cepat terutama dalam hal koneksi baik ke jaringan maupun ke perangkat lain. Maka dari itu, sering kali dalam pembuatan alat IoT algoritma yang digunakan dalam koneksi ke perangkat lain tidak memiliki prosedur yang memastikan perangkat berhak untuk melakukan koneksi ke alat IoT. Hal ini terutama sangat penting jika alat bersifat mudah dicuri (seperti *ear buds*) atau mengandung informasi sensitif (seperti kamera CCTV).

Oleh karena itu, makalah ini akan membahas sebuah usulan solusi yang berupa algoritma autentikasi dalam bentuk fungsi *hash*. Solusi menggunakan fungsi *hash* dikarenakan proses verifikasi pengguna tidak memerlukan proses dekripsi sebuah pesan sehingga fungsi kriptografi dua arah tidak diperlukan. Selain itu, solusi memerlukan sebuah fungsi yang bersifat ringan mengingat terbatasnya kapabilitas yang dimiliki oleh sebuah alat IoT.

Makalah ini juga tidak hanya mengimplementasikan satu buah fungsi *hash*. Namun, ketiga versi *Secure Hash Algorithm* (SHA-1, SHA-2, SHA-3) akan diimplementasikan untuk diuji kinerjanya dalam sebuah alat IoT. Pengujian akan berfokus pada kecepatan dalam melakukan *hashing* yang diperlukan pada saat *hashing*.

II. Dasar Teori

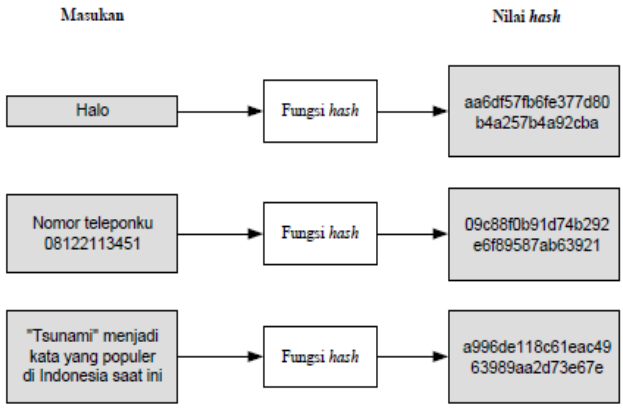
A. Password Hashing

Dalam sebuah proses verifikasi seorang pengguna, biasanya pengguna akan memberi dua hal: sebuah ID yang akan memberitahu siapa pengguna tersebut (seperti username) dan sebuah kata kunci yang memastikan bahwa pihak yang berkomunikasi memang benar adalah pengguna tersebut. Tentunya dalam pengiriman kata kunci data tersebut harus di enkripsi agar tidak terlihat jika terjadi penyadapan komunikasi atau pembobolan penyimpanan data. Perlu diingat bahwa hasil enkripsi tersebut tidak perlu bisa dienkripsi namun hanya perlu dicocokkan kesamaannya dengan yang ada di penyimpanan. Hal ini bukan hanya bersifat tidak perlu namun juga berbahaya, karena penyimpanan hasil enkripsi kata kunci yang bisa di dekripsi akan menimpbukan sebuah bahaya pembobolan data kata kunci asli pengguna.

Oleh karena itu, kebanyakan sistem yang memerlukan kata kunci hanya menyimpan hasil *hash* dari kata kunci. Dimana sistem akan melakukan verifikasi dengan melakukan *hashing input* pengguna dan mencocokkannya dengan hasil *hash* yang ada di simpanan. Proses inilah yang disebut dengan *password hashing*.

B. Fungsi Hash

Fungsi *hash* adalah sebuah fungsi yang mengubah sebuah pesan berukuran sembarang menjadi *string* berukuran tetap yang dinamakan pesan ringkas atau *message-digest*. Hasil pengubahan ini bersifat *irreversible* (tidak bisa dikembalikan seperti semula).



Gambar II.1 Ilustrasi penggunaan fungsi *hash*.

Sebuah fungsi *hash* memiliki sifat-sifat sebagai berikut:

1. *Collision resistance*: Sangat sukar menemukan dua *input* a dan b sehingga Hash(a) = Hash(b).
2. *Preimage resistance*: Untuk sembarang *output* y, sukar menemukan *input* a sehingga Hash(a) = y.
3. *Second preimage resistance*: Jika diketahui *output y* dan *input* a dengan Hash(a) = y. Sukar menemukan sebuah *input* b sehingga Hash(b) = y.

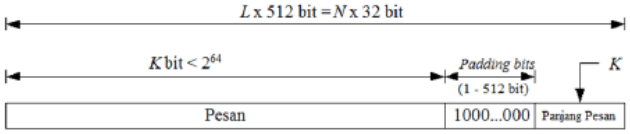
Sifat-sifat tersebut adalah mengapa *password hashing* dipakai. Sang penyerang tidak akan bisa menemukan kata kunci asli dari *hash*. Jika *brute force attack* dilakukan, sangat sukar ditemukan sebuah kata kunci yang berbeda namun terjadi *collision* sehingga penyerang terverifikasi.

C. Fungsi Hash SHA-1

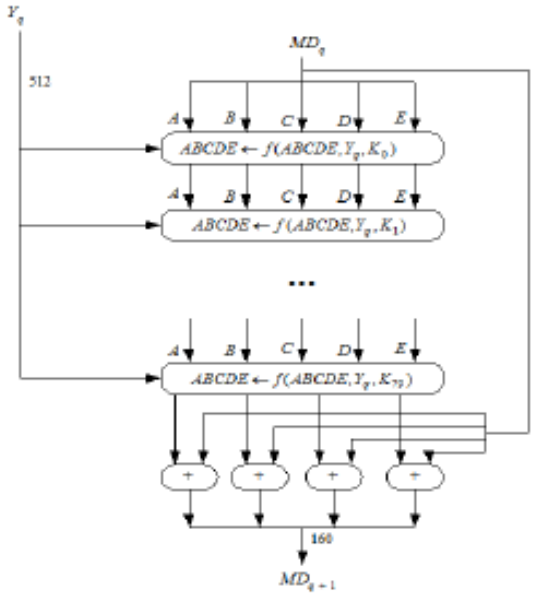
SHA (*Secure Hash Algorithm*) adalah sebuah fungsi enkripsi satu arah yang dikembangkan oleh *National Institute of Standards and Technology* (NIST) yang didasari oleh algoritma MD4 yang dikembangkan oleh Ronald L. Rivest dari MIT. SHA-1 adalah SHA-0 (atau SHA biasa) yang ditingkatkan dengan menambahkan sebuah *bitwise rotation* dalam kompresi pesan [1]. Oleh karena peningkatan yang sedikit serta karena tidak adanya rilis resmi dari SHA-0, banyak orang beranggapan bahwa SHA-1 adalah SHA pertama.

Fungsi SHA-1 akan digunakan dalam makalah ini sebagai salah satu alternatif dari *password hashing*. Cara kerja SHA-1 secara umum adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (*padding bits*).
2. Penambahan nilai panjang pesan.
3. Inisialisasi penyangga (*buffer*) MD
4. Pengolahan pesan dalam blok berukuran 512 bit.

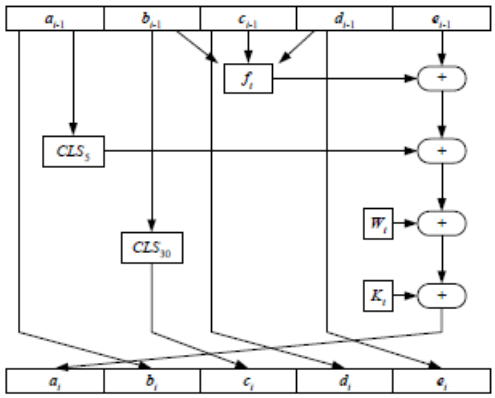


Gambar II.2 Ilustrasi hasil penambahan bit-bit pengganjal serta nilai panjang pesan



Gambar II.3 Ilustrasi pengolahan pesan dalam blok berukuran 512 bit.

Dilihat pada Gambar II.3 bahwa fungsi SHA menggunakan lima *buffer* MD5 yang dihasilkan pada tahap kedua dengan sebuah fungsi kompresi yang menghasilkan 5 *buffer* baru. *Buffer* baru ini juga akan dimasukkan ke fungsi kompresi. Hal ini akan diulang sebanyak 80 kali.

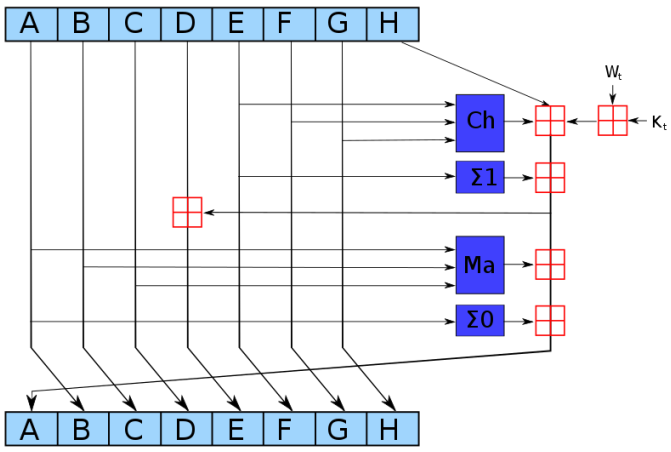


Gambar II.4 Fungsi kompresi yang digunakan setiap putaran.

Algoritma SHA-1 ini dianggap cukup aman hingga tahun 2011, dimana NIST sendiri menganggap SHA-1 telah *deprecated* dan melarang penggunaannya sebagai *digital signature* pada tahun 2013 [2]. Oleh karena alasan ini, SHA-1 tidak akan dipakai dalam implementasi sebenarnya namun hanya sebagai pembanding dalam kinerja *hashing*.

D. Fungsi-fungsi Hash SHA-2

SHA-2 adalah sebuah kumpulan fungsi enkripsi satu arah yang dikembangkan oleh *National Security Agency* (NSA) pada tahun 2001 yang didasarkan oleh SHA-1. SHA-2 dibuat sebagai pengganti SHA-1 yang sudah dianggap tidak aman. Tidak seperti SHA-1 yang terbatas pada ukuran 512 bits, SHA-2 mendukung operasi dalam 224, 256, 384, dan 512 bit dengan tiap *bit-level* membuat sebuah algoritma baru dalam kumpulan SHA-2. Cara kerja utama SHA-2 sebenarnya mirip dengan SHA-1 dengan perubahan yang dilakukan adalah mengganti penuh fungsi kompresi yang dilakukan tiap putaran.



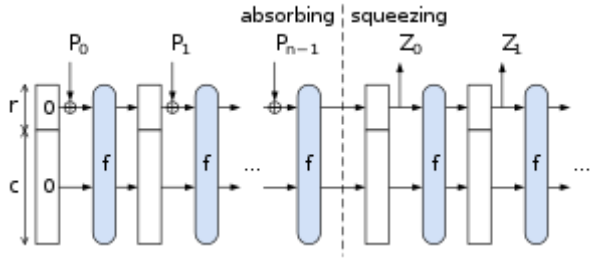
Gambar II.5 Fungsi kompresi yang digunakan setiap putaran SHA-2

Hingga saat ini belum ditemukan sebuah *collision* diantara dua hasil *hash* sehingga untuk sekarang dianggap cukup aman untuk melakukan *password verification*. Walau begitu, telah diketahui sebuah serangan yang efektif terhadap SHA-2 yaitu

length-extension attacks (Penjelasan mengenai *length extension attacks* berada pada Subbab G). Oleh karena itu, sebuah HMAC SHA-2 akan dipakai dalam implementasi dengan SHA-2 yang dimaksud adalah SHA-256 mengingat kecilnya kapasitas memori yang ada pada alat IoT dan sifat SHA-256 yang lebih tahan *extension attacks* daripada SHA-224.

E. Fungsi-fungsi Hash SHA-3

SHA-3 adalah sekumpulan fungsi enkripsi satu arah yang dirilis oleh NIST pada tanggal 5 Agustus 2015. SHA-3 merupakan bagian kecil dari sebuah kelompok fungsi *hash* yang dibuat oleh Guido Bertoni, Joan Daemen, Michael Peeters, dan Gilles Van Assche [3] sebagai bagian dari kompetisi yang diumumkan NIST pada tahun 2007 dengan nama Keccak. Berbeda dengan SHA-2, SHA-3 tidak memiliki hubungan sama sekali dengan SHA-1. Berbeda dari fungsi *hash* lainnya yang menggunakan fungsi kompresi yang dilakukan berulang-ulang, Struktur utama dari SHA-3 menggunakan konsep *sponge construction* untuk 'menyerap dan memeras' *message-digest*.



Gambar II.6 Ilustrasi *sponge construction* SHA-3

Terlihat dari Gambar II.6 bahwa proses *hashing* dilakukan dengan dua tahap yaitu *absorbing* dan *squuezing*. Detail proses *hashing* adalah sebagai berikut:

1. Tentukan nilai *rate* r.
2. Tentukan nilai panjang blok b yang digunakan fungsi permutasi f.
3. Tentukan nilai panjang *output* d.
4. Tambahkan pengganjal (*padding*) pada *plaintext* M agar M dapat dibagi rata menjadi bagian-bagian yang berukuran r.
5. Buat sebuah *state* S yang berukuran b dan diisi nilai 0.
6. Untuk setiap bagian dari M, lakukan fase penyerapan sebagai berikut:
    a. XOR dengan r bit pertama dari S.
    b. Masukkan S yang baru ke dalam fungsi permutasi f.
7. Buat *string* kosong Z.
8. Selama panjang Z kurang dari d, lakukan fase pemerasan sebagai berikut:
    a. Tambahkan r bit pertama dari S ke dalam Z.
    b. Masukkan S ke dalam fungsi permutasi f, jadikan hasil sebagai S yang baru.
9. Potong panjang Z jika lebih besar dari d.

Sampai saat ini belum ada *collision* yang ditemukan dari SHA-3 yang berarti SHA-3 masih aman untuk digunakan dalam *password verification*. Terlebih lagi, SHA-3 aman terhadap *extension attack* fungsi *hash* bisa dikelabui dengan menambahkan panjang pesan. Hal ini berarti SHA-3 tidak memerlukan kunci dalam proses *hash*. Oleh karena itu akan digunakan SHA3-224 sebagai fungsi *hash*. Penggunaan SHA3-256 adalah karena *output size* nya yang sangat kecil hingga dapat mendukung kapabilitas IoT yang kecil pula.

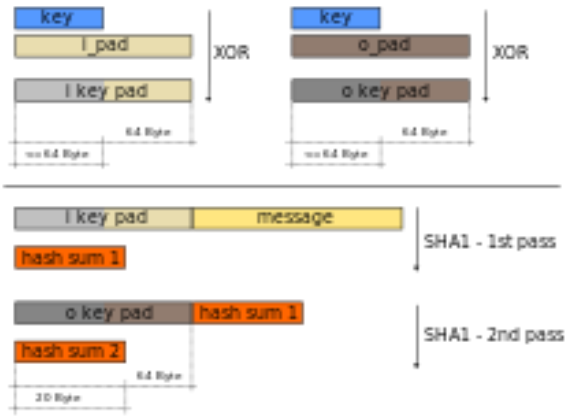
F. Internet of Things

Internet of Things atau IoT adalah semua objek fisik yang dilengkapi dengan *sensor*, perangkat lunak, dan teknologi lain yang membuat objek tersebut dapat tersambung dan bertukar pesan dengan objek lain melalui Internet. Konsep dari *smart devices* atau IoT diawali dengan dibuatnya sebuah *vending machine* di Carnegie Mellon University yang dimodifikasi agar bisa terhubung ke Internet, membuatnya menjadi IoT pertama. Konsep ini terus berkembang dan semakin populer hingga rasio kepemilikan IoT per orang meningkat dari 0.08 di tahun 2003 menjadi 1.84 di tahun 2010 [5].

Pada tingkat jaringan, IoT menggunakan beragam protokol untuk melakukan koneksi ke Internet, beberapa langsung koneksi ke internet melalui Wi-Fi atau Ethernet, lainnya berkoneksi dengan menggunakan *bluetooth* untuk koneksi ke alat perantara (seperti telepon genggam). Pada tingkat aplikasi, setiap IoT menggunakan protolol yang berbeda-beda sesuai dengan pembuatnya. Namun, pada akhirnya semua protokol tersebut tidak ada proses autentikasi dari alat IoT itu sendiri dengan asumsi bahwa siapa yang memegang alat IoT tersebut maka ialah pemiliknya. Asumsi ini memberi sebuah bahaya pada penggunaan IoT dengan penyerang hanya perlu berinteraksi fisik sekali dengan IoT agar penyerang memiliki akses penuh. Fakta ini menyebabkan pencurian IoT menjadi fatal. Oleh karena itu, makalah ini bertujuan untuk memberi solusi terhadap masalah ini dengan memberi sebuah algoritma autentikasi pada protokol koneksi IoT.

G. Keyed-hash Message Authentication Code dan Length extension attack

Perlu diketahui bahwa arti *hashing* dalam makalah ini bukan mengarah pada proses *hashing* tradisional yang hanya memrlukan pesan saja namun mengarah pada proses *hashing* yang menggunakan sebuah kunci tertentu. Proses *hashing* yang menambahkan kunci ini disebut *Keyed-hash Message Authentication Code* atau HMAC. HMAC dibuat karena adanya sebuah metode serangan yang bernama *length extension attack* dimana jika diketahui Hash(m1), sebuah pesan m2 dan panjang m1, penyerang dapat mengetahui Hash(m1 || m2). Serangan ini dapat mengancam integritas dari *password verification*. Oleh karena itu, dibuat HMAC.



Gambar II.7 Implementasi HMAC pada SHA-1

III. Implementasi dan Rancangan Solusi

A. Deskripsi Umum Solusi

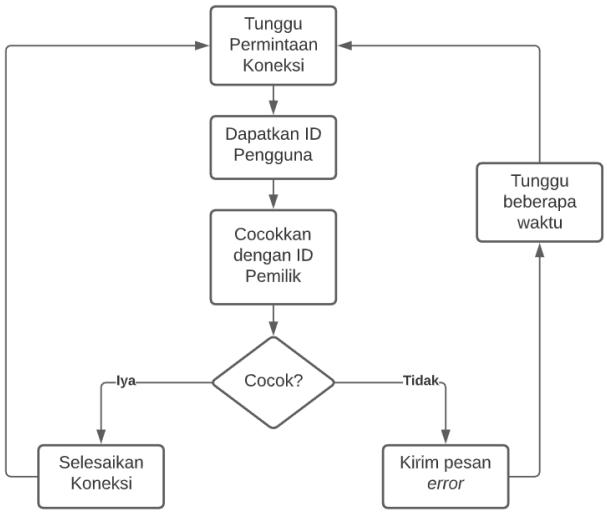
Solusi yang dibuat adalah sebuah algoritma verifikasi ID pengguna pada sisi alat IoT dengan membandingkan hasil *hash* ID pengguna dan hasil *hash* ID pemilik yang telah disimpan. *Hashing* yang dimaksud adalah pembuatan HMAC sesuai dengan ID yang diterima dan kunci yang disimpan IoT.

Fungsi *hash* yang digunakan adalah SHA-256 dan SHA3-256 karena *output size* nya yang kecil. Sesuai dengan algoritma yang digunakan, panjang kedua *hash* yang dibandingkan adalah 256 bit.

Setelah kedua fungsi *hash* diimplementasikan, hasil pencocokkan kedua *hash* tersebut akan diuji di verifikasi. Selain itu, performa *hashing* juga akan di evaluasi berikut dengan performa *hashing* algoritma SHA-1 dalam bentuk waktu yang dibutuhkan. Algoritma SHA-1 tidak termasuk dalam implementasi.

B. Rancangan Solusi

Algoritma autentikasi dibuat dengan menambahkan sebuah konstanta ID pemilik IoT yang telah di *hash* ke dalam IoT. Konstanta ini diharapkan telah di *hard-wired* ke dalam IoT tersebut sehingga sukar untuk diubah oleh orang lain. Setiap kali perangkat lain ingin melakukan koneksi ke IoT, IoT akan meminta ID dari perangkat tersebut dan melakukan *hashing* dari ID tersebut. Hasil *hash* akan dibandingkan dengan konstanta. IoT akan melanjutkan proses koneksi jika kedua hasil *hash* tersebut sama. Jika kedua hasil tidak sama, IoT akan mengirimkan pesan kegagalan autentikasi dan menunggu beberapa waktu sebelum dilakukan percobaan koneksi selanjutnya. Waktu tunggu ini memastikan bahwa penyerang tidak bisa melakukan *brute-force attack* untuk menebak ID pengguna yang sebenarnya. Berikut adalah diagram alur algoritma:



Gambar III.1 Diagram alur algoritma

C. Implementasi Solusi

Solusi akan menggunakan arduino sebagai basis pengembangannya. Mengapa menggunakan arduino? Karena arduino adalah salah satu *platform* terkemuka dalam pengembangan IoT yang bersifat *open source*. Selain itu, sifat *arduino* yang mendukung bahasa C dan banyaknya *library* yang tersedia membuat pembangunan solusi lebih mudah untuk arduino.

Berikut adalah contoh *pseudo-code* dalam sebuah arduino:

```
void setup(){
  const char *key = "secretKey";
  const char *hashedID = "HashedUserID";
  const int timeOut = 5000;
  char *input;
  char *hashedInput;
  bool connect;
}

void loop() {
  // Memulai proses koneksi
  if (connect) {
    // Mengambil input berupa ID pengguna
    input = getInput();
    // Melakukan hashing terhadap input menggunakan kunci
    hashedInput = hash(input, key)
    // Verifikasi kesamaan hasil hash
    if (hashedInput == hashedID) {
      // Jika benar lanjutkan koneksi
      connect();
    } else {
      // Jika salah, tunggu beberapa waktu
      delay(timeOut);
    }
  }
}
```

Gambar III.1 *Pseudo-code* algoritma di arduino

Semua implementasi dari masing masing algoritma *hash* pada dasarnya akan mengikuti skema *pseudo-code* ini, dengan perbedaan bersifat kecil seperti *library* yang digunakan dan untuk fungsi SHA-3 yang tidak menggunakan kunci maka kunci hanya akan di *append* ke ID.

```
void setup(){
  const char *key = "secretKey";
  const char *hashedID = "HashedUserID";
  const int timeOut = 5000;
  char *input;
  char *hashedInput;
  bool connect;
}

void loop() {
  // Memulai proses koneksi
  if (connect) {
    // Mengambil input berupa ID pengguna
    input = getInput();
    // Untuk SHA-3, key langsung di append ke input
    input = append(input, key)
    // Melakukan hashing terhadap input menggunakan kunci
    hashedInput = hash(input)
    // Verifikasi kesamaan hasil hash
    if (hashedInput == hashedID) {
      // Jika benar lanjutkan koneksi
      connect();
    } else {
      // Jika salah, tunggu beberapa waktu
      delay(timeOut);
    }
  }
}
```

Gambar III.2 *Pseudo-code* algoritma dengan SHA-3 di arduino

IV. Pengujian dan Evaluasi

Pengujian akan dilakukan untuk mengetahui kebenaran verifikasi ID yang dilakukan oleh algoritma. Evaluasi akan dilakukan untuk menghitung dan membandingkan performa

(waktu yang dibutuhkan) *hashing* antara satu sama lain.

A. Pengujian Solusi

Solusi akan diuji dengan menggunakan algoritma SHA-1, SHA-2, dan SHA-3 yang telah diimplementasikan dalam kode C dalam arduino untuk menghasilkan sebuah hasil *hash* ID pemilik yang akan disimpan. Solusi akan diuji dengan berbagai ID yang diubah untuk *collision resistance* dari tiap-tiap algoritma.

Dengan variabel awal yang akan di *hard-code* ke dalam arduino sebagai berikut:

Tabel I. Variabel Awal Pengujian

ID Pemilik	mzYQ3dprJeH164NytRb5E2JhbcuWSTZ6
Kunci	secretKey

Dengan variabel-variabel yang sudah ditentukan di atas, didapatkan hasil *hash* sebagai berikut:

Tabel II. Hasil *Hash* Variabel Awal

Hasil *hash* SHA-1	432c90f7d997a388898149b3cb6f1a9311f012fc
Hasil *hash* SHA-2	087ec2c9645096f7889844a1fe6a39395cc264a8
Hasil *hash* SHA-3	901e40449ce92cfc2219eeb979c2ebabfa837b1b

Untuk proses pengujian kebenaran dari verifikasi ID, akan digunakan beberapa variasi ID pengguna sebagai berikut:

Tabel III. Macam-macam Variabel Uji Coba

ID Pengguna yang Benar	mzYQ3dprJeH164NytRb5E2JhbcuWSTZ6
ID Pengguna dengan perubahan 1 bit	mzYQ3dprJeH164NytRb5E2JhbcuWSTZ7
ID Pengguna dengan tambahan di belakang	mzYQ3dprJeH164NytRb5E2JhbcuWSTZ60
ID Pengguna dengan tambahan di depan	0mzYQ3dprJeH164NytRb5E2JhbcuWSTZ6

Variasi-variasi ID tersebut akan digunakan sebagai input ke arduino untuk dilakukan uji coba autentikasi pengguna dengan ID pemilik. Berikut adalah hasil uji coba:

Tabel IV. Hasil Autentikasi Variabel ID Pengguna yang Benar

Hasil *hash* SHA-1	432c90f7d997a388898149b3cb6f1a9311f012fc
Hasil *hash* SHA-2	087ec2c9645096f7889844a1fe6a39395cc264a8
Hasil *hash* SHA-3	901e40449ce92cfc2219eeb979c2ebabfa837b1b
Output autentikasi	true
Pesan hasil autentikasi	"Autentikasi berhasil, melanjutkan koneksi"

Tabel V. Hasil Autentikasi Variabel ID Pengguna dengan Perubahan 1 Bit

Hasil *hash* SHA-1	a2dc25f1cab189c265520fb17dbda1773edd3ee6
Hasil *hash* SHA-2	0d967b07e79fd7cb90c4c3e0b734005b2ccef575
Hasil *hash* SHA-3	49f344df9846d1a503e03c0d34be63bc51fa87c4
Output autentikasi	false
Pesan hasil autentikasi	"Autentikasi gagal, tunggu beberapa saat lagi"

Tabel VI. Hasil Autentikasi Variabel ID Pengguna dengan Tambahan di Belakang

Hasil *hash* SHA-1	8462b48eb15b383c98f759dc8aa3fe69ead8d74c
Hasil *hash* SHA-2	51dfbe28848100dd3483db0922ee924773106619
Hasil *hash* SHA-3	1706bda94c0bb3b73452a2fbd8ba4da1060417a6
Output autentikasi	false
Pesan hasil autentikasi	"Autentikasi gagal, tunggu beberapa saat lagi"

Tabel VII. Hasil Autentikasi Variabel ID Pengguna dengan Tambahan di Depan

Hasil *hash* SHA-1	6be5a63c3dcee31b08ccc3fbabffa16e2a04f1cc
Hasil *hash* SHA-2	2f0e77c3adfe706069246965729c35a5cee4ae2e
Hasil *hash* SHA-3	f265b2f3c593b87374ba050f9e93ab4842dbf04d
Output autentikasi	false
Pesan hasil autentikasi	"Autentikasi gagal, tunggu beberapa saat lagi"

Dari hasil pengujian autentikasi tersebut, didapatkan bahwa semua algoritma telah berjalan dengan sempurna dan siap untuk diimplementasikan ke tahap selanjutnya yaitu tahap evaluasi performa. Selain itu, didapatkan juga bahwa hasil pengujian dari SHA-1 dapat disimpulkan bahwa keamanan dari algoritma tersebut terlihat baik-baik saja, lalu mengapa SHA-1 tidak diimplementasikan pada solusi? Jawabannya simpel, walaupun sebuah *collision* dari sebuah algoritma sangat sukar untuk ditemukan (sampai saat ini pun tidak ada sebuah algoritma untuk SHA-1 yang dapat menemukan sebuah *message* lain yang memiliki nilai *hash* yang sama), kenyataan bahwa ada *collision-attack* yang berhasil membuktikan bahwa SHA-1.

"Sebuah keberhasilan *collision-attack* pada sebuah algoritma *hash* itu seperti menemukan kenyataan bahwa alat operasi Anda tidak disterilkan secara betul. Walaupun belum pasti ada kuman di alat tersebut, kenyataan itu sudah cukup untuk menyimpulkan bahwa alat tersebut sudah tidak aman" - Matt Green, Profesor di Johns Hopkins University [7]

B. Evaluasi Kinerja

Untuk mencegah terjadinya kerusakan pada alat arduino dan karena sifat ketidakstabilan arduino itu sendiri. Evaluasi kinerja akan dilakukan pada sebuah komputer dengan spesifikasi sebagai berikut:

Tabel VIII. Spesifikasi Alat Penguji

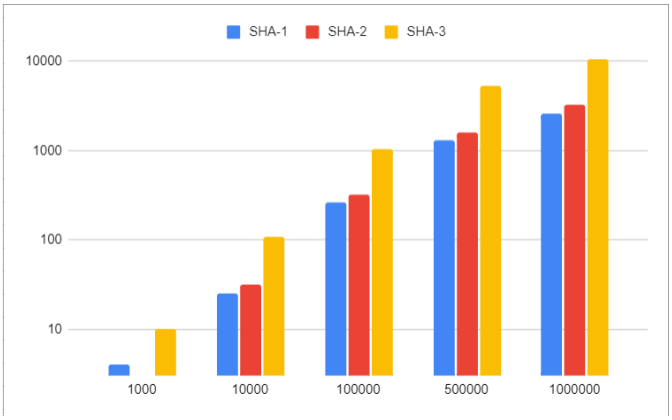
Prosesor	Intel Core i7-8750H @ 2.20 GHz
RAM	16 GB
OS	Windows 10

Untuk menyesuaikan dengan sifat alat IoT yang berkapasitas kecil, maka pengujian akan menggunakan pengulangan *hash* sebanyak n kali (dimana n akan dinaikkan secara eksponensial). Hal yang akan dihitung dan dibandingkan dari pengujian adalah waktu yang dibutuhkan (yang akan memberitahu kita tentang penggunaan CPU) yang dibutuhkan untuk menyelesaikan semua *hash*. Memori yang digunakan tidak akan dihitung pada pengujian ini mengingat proses berjalan secara *single-threaded* dan variabel sementara yang disimpan akan langsung dihapus setelah proses *hash* selesai

Dari hasil evaluasi, didapatkan data sebagai berikut:

Tabel IX. Jumlah waktu yang dibutuhkan untuk menyelesaikan n *hashing* (dalam milidetik)

Jumlah n	SHA-1	SHA-2	SHA-3
1000	4	3	10
10000	25	32	107
100000	262	319	1047
500000	1286	1606	5272
1000000	2574	3214	10543



Gambar IV.1 Diagram Hasil Evaluasi

Didapat dari hasil evaluasi bahwa peningkatan waktu yang dibutuhkan untuk setiap iterasi SHA bertambah semakin modern SHA yang digunakan. Hal ini konsisten dengan kenyataan bahwa semakin modern SHA yang digunakan, semakin kompleks algoritma yang diimplementasi. Pengujian dari pihak lain yang menjalankan algoritma SHA dengan Intel CPU juga memberikan hasil yang serupa, dengan waktu yang dibutuhkan SHA-3 dua kali lebih banyak daripada SHA-2, dan waktu yang dibutuhkan SHA-2 tiga kali lebih banyak daripada SHA-1.[8]

Selain itu, dengan kenyataan bahwa prosesor tipikal *arduino* memiliki *clock speed* setengah dan RAM 1/32 dari alat penguji. Maka dapat disimpulkan bahwa alat-alat IoT dapat mengatasi permintaan autentikasi pengguna yang dijalankan sekali tiap koneksi. Dimana permintaan koneksi tersebut hanya akan dijalankan sekali-sekali tiap waktu.

V. KESIMPULAN DAN SARAN PENGEMBANGAN

Solusi yang diimplementasikan berhasil melakukan verifikasi berdasarkan hasil *hash* kedua ID dan berhasil mempertahankan fitur utama dari fungsi *hash* yaitu *collision resistance*. Selain itu, telah dibuktikan bahwa penggunaan *hash* dalam autentikasi pada sisi alat IoT cukup memadai menimbang kapabilitas dari IoT yang biasanya.

Selain itu, didapatkan dari hasil evaluasi bahwa algoritma yang cocok untuk solusi ini adalah algoritma SHA-2. Hal ini mempertimbangkan peningkatan waktu SHA-3 yang jauh lebih besar dibandingkan SHA-2. Ditambah lagi, SHA-2 masih digunakan sebagai standar fungsi *hash* sehingga lebih cocok digunakan untuk meningkatkan konektivitas dengan perangkat-perangkat lain. Satu-satunya pertimbangan tidak menggunakan SHA-2 adalah kerentanannya pada *length-extension attacks*. Namun, solusi ini tidak perlu khawatir akan hal itu karena solusi menggunakan versi HMAC dari SHA-2 yang tahan terhadap *length-extension attack*.

Untuk kedepannya, solusi dapat dikembangkan untuk mendukung *multi-threading* yang bisa menangani beberapa percobaan koneksi secara bersamaan. Solusi juga dapat menggunakan enkripsi dalam pengiriman ID jika jaringan dianggap tidak aman dan kedua alat cukup memadai untuk melakukan enkripsi/dekripsi. Solusi juga dapat dikembangkan dednga mengimplementasikan fungsi *hash* lain seperti MD5 dan CryptoLUX.

VII. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Tuhan yang Maha Esa Allah Swt. karena atas berkah dan rahmat-Nya penulis dapat menyelesaikan makalah ini dalam rangka menyelesaikan studi IF4020 Kriptografi yang ada di Teknik Informatika Institut Teknologi Bandung ini.

Selain itu, penulis juga ingin mengucapkan terima kasih kepada kedua orang tua penulis yang telah mendukung penulis dari awal sampai akhir. Penulis juga ingin mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku dosen mata kuliah IF4020 Kriptografi yang telah memberi penulis ilmu agar bisa menyelesaikan makalah ini.

Terakhir, penulis ingin menyampaikan terima kasih juga kepada rekan-rekan satu angkatan UNIX 2017 yang telah membantu penulis dalam menjalani perkuliahan Kriptografi terutama teman-teman grup Jiwa yang telah sabar menghadapi penulis.

REFERENCES

[1] Kramer, Samuel (11 July 1994). "Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard". Federal Register.

[2] "Critical flaw demonstrated in common digital security algorithm". media.ntu.edu.sg.

[3] Guido Bertoni; Joan Daemen; Michaël Peeters; Gilles Van Assche. "The Keccak sponge function family: Specifications summary".

[4] Rouse, Margaret (2019). "internet of things (IoT)". IOT Agenda.

[5] Dave Evans (April 2011). "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything". CISCO White Paper.

[6] "Is HMAC needed for SHA-3 based MAC?" (Internet, diambil 21 Desember 2020): https://crypto.stackexchange.com/questions/17735/is-hmac-needed-for-a-sha-3-based-mac

[7] "At death's door for years, widely used SHA1 function is now dead" (Internet, diambil 21 Desember 2020): https://arstechnica.com/information-technology/2017/02/at-deaths-door-for-years-widely-used-sha1-function-is-now-dead

[8] "Why aren't we using SHA-3?" (Internet, diambil 21 Desember 2020): https://www.csoonline.com/article/3256088/why-arent-we-using-sha3.html

[9] Munir, Rinaldi. 2019. Slide Kuliah IF4020 Kriptografi: Fungsi Hash.

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, 21 Desember 2020

T. Antra Oksidian Tafly 13517020