Implementasi *HMAC-SHA-3-Based One Time Password* pada Skema *Two-Factor Authentication*

Muhamad Fakhrusy (13612020)
Program Studi Aeronotika dan Astronotika
Fakultas Teknik Mesin dan Dirgantara
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mfakhrusy@outlook.com

***Abstract*—skema *two-factor authentication* dibuat untuk memberikan tingkat keamanan tambahan bagi user. Salah satu contoh penerapan *two-factor authentication* pada proses login user adalah autentikasi dengan hash password ditambah dengan autentikasi menggunakan *one time password*. Pada jurnal ini, pembahasan akan fokus pada implementasi HMAC-SHA-3 untuk algoritma *HMAC-based one time password* yang masih menggunakan HMAC-SHA-1. Hal ini dilakukan karena SHA-3 akan memberikan keamanan yang lebih baik dibandingkan SHA-1.**

Keywords—keccak, SHA-3, one time password, HOTP, two-factor authentication, autentikasi.

I. Pendahuluan

Proses autentikasi adalah proses yang krusial pada sebuah proses login. Pada kebanyakan website, proses login user kepada server adalah proses yang hampir selalu diimplementasikan. Selain proses login, proses autentikasi pun terjadi di banyak tempat terutama website-website dinamik. Misalkan seperti proses autentikasi jika sebuah user sudah tersimpan dalam suatu klien, user itu harus terus dilakukan validasi selama dia melakukan proses-proses dalam website. Salah satu cara yang paling mudah adalah menyimpan ip address user di server dan melakukan validasi terhadap setiap proses user dengan ip server user.

Untuk memudahkan pengembangan keamanan dari proses autentikasi, biasanya dilakukan simulasi penyerangan dari segi di mana sistem itu memiliki kelemahan. Dengan cara itu, sang *web security* dari sebuah website bisa mendesain sistem keamanan yang paling cocok, paling aman, serta paling murah (secara harga dan waktu) untuk diimplementasikan pada websitenya. Seringkali juga sebuah perusahaan yang memiliki website dinamik memberikan sebuah sayembara untuk para pembobol serta penyerang website untuk melakukan pembobolan terhadap website itu, di mana jika ditemukan kelemahan pada websitenya, sang pembobol berhak mendapat hadiah tertentu (biasa disebut *bounty hunter*).

Untuk meningkatkan keamanan dari sebuah proses autentikasi user, dapat digunakan skema bernama *two-factor authentication*. Skema ini sesuai namanya yaitu memberikan lapisan keamanan ekstra sebagai proses autentikasi. Autentikasi pertama dilakukan dengan identifikasi antar user (dalam hal ini password user) sedangkan autentikasi kedua dilakukan dengan apa yang user punya. Autentikasi kedua dapat dilakukan melalui telepon genggam user, e-mail user, dan sebagainya.

II. Dasar Teori

A. SHA-3

SHA-3 adalah algoritma kriptografi standar yang dihasilkan pada lomba yang diadakan oleh NIST. Nama lain dari SHA-3 adalah algoritma Keccak dengan perancang: Guido Bertoni, Joan Daemen, Michael Peeters, dan Gilles van Assche. Pada kejuaraan oleh NIST itu, Keccak menjadi pemenang karena beberapa alasan yang diantaranya adalah tingkat keamanannya serta tingkat kerumitan dari algoritma yang tidak terlalu tinggi untuk diaplikasikan ke perangkat keras maupun perangkat lunak.

Pada Keccak, output serta blok perhitungan dapat ditentukan sendiri oleh user sesuai kebutuhan, namun biasanya menggunakan sebuah standar yang ditetapkan oleh NIST yang bernama standar FIPS 202 yang mencakup:

	r	c	d	Output length (bits)	Security level (bits)
SHAKE128	1344	256	0x1F	unlimited	128
SHAKE256	1088	512	0x1F	unlimited	256
SHA3-224	1152	448	0x06	224	112
SHA3-256	1088	512	0x06	256	128
SHA3-384	832	768	0x06	384	192
SHA3-512	576	1024	0x06	512	256

Fig. 1: Standar FIPS 202

Nilai r adalah nilai dari blok setiap iterasi perhitungan putaran. Nilai c didapat dari $1600 - r$ di mana 1600 didapat dari jumlah bit dari satu *state* Keccak yang digunakan secara *default*, selain 1600, nilai-nilai state yang lain adalah: $b \in 25, 50, 100, 200, 400, 800, 1600$

State keccak digambarkan sebagai sebuah balok array 3-dimensi yang memiliki nilai 5x5 pada dua sumbu serta nilai w yang bervariasi pada sumbu ketiga, di mana nilai $b = 25w$ untuk b yang telah disebutkan sebelumnya. Maka nilai w adalah: $b \in 1, 2, 4, 8, 16, 32, 64$ dengan nilai $w = 64$ sebagai nilai default yang biasa diimplementasi. Jika dilihat dalam bentuk 3-dimensi, akan terlihat seperti berikut:

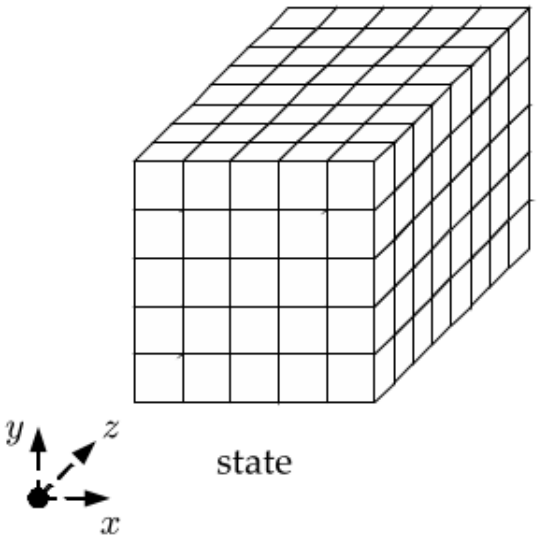


Fig. 2: Gambar State dari Keccak

Untuk setiap blok pada *state* di atas berisikan nilai bit. Proses yang dilakukan melibatkan perhitungan untuk tiap w-bit. Maka, proses akan dilakukan untuk tiap 64-bit jika mengacu pada standar operasi.

Algoritma yang menjadi inti perhitungan Keccak adalah algoritma Keccak-f. Keccak-f terdiri dari algoritma putaran khusus yang dilakukan selama $n_r - 1$ putaran. Jumlah n_r ditentukan dari persamaan $n_r = 12+2l$ di mana nilai l dihitung dengan hubungan $2^l = w$. Maka hal ini memberikan nilai $l = 6$ untuk $w = 64$. Oleh karena itu akan didapatkan nilai putaran yaitu $n_r = 24$.

Tiap putaran dari keccak akan menghitung suatu algoritma yang terdiri dari: (catatan: x dan y berarti sumbu-x dan sumbu-y pada bidang 5x5)

1) fasa θ

Algorithm 3 θ

for $x = 0$ to 4 **do**
  $C[x] = a[x,0]$
  for $y = 1$ to 4 **do**
    $C[x] = C[x] \oplus a[x,y]$
  end for
end for
for $x = 0$ to 4 **do**
  $D[x] = C[x-1] \oplus \text{ROT}(C[x+1],1)$
  for $y = 0$ to 4 **do**
    $A[x,y] = a[x,y] \oplus D[x]$
  end for
end for

Fig. 3: fasa θ

2) fasa ρ dan π

Algorithm 5 ρ

$A[0,0] = a[0,0]$
$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
for $t = 0$ to 23 **do**
  $A[x,y] = \text{ROT}(a[x,y],(t+1)(t+2)/2)$
  $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$
end for

Fig. 4: fasa ρ

Algorithm 4 π

for $x = 0$ to 4 **do**
  for $y = 0$ to 4 **do**
    $\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$
    $A[X, Y] = a[x, y]$
  end for
end for

Fig. 5: fasa π

3) fasa χ

Algorithm 2 χ

for $y = 0$ to 4 **do**
  for $x = 0$ to 4 **do**
    $A[x, y] = a[x, y] \oplus ((\text{NOT } a[x+1, y]) \text{ AND } a[x+2, y])$
  end for
end for

Fig. 6: fasa χ

4) fasa ι
Pada fasa ini, nilai A[0][0] hanya dilakukan XOR terhadap *round Constant* yang dapat dilihat di standar.

Nilai Round Constant:

RC[0]	0x0000000000000001	**RC[12]**	0x000000008000808B
RC[1]	0x0000000000008082	**RC[13]**	0x800000000000008B
RC[2]	0x800000000000808A	**RC[14]**	0x8000000000008089
RC[3]	0x8000000080008000	**RC[15]**	0x8000000000008003
RC[4]	0x000000000000808B	**RC[16]**	0x8000000000008002
RC[5]	0x0000000080000001	**RC[17]**	0x8000000000000080
RC[6]	0x8000000080008081	**RC[18]**	0x000000000000800A
RC[7]	0x8000000000008009	**RC[19]**	0x800000008000000A
RC[8]	0x000000000000008A	**RC[20]**	0x8000000080008081
RC[9]	0x0000000000000088	**RC[21]**	0x8000000000008080
RC[10]	0x0000000080008009	**RC[22]**	0x0000000080000001
RC[11]	0x000000008000000A	**RC[23]**	0x8000000080008008

Fig. 7: Round Constant untuk RC[i]

Nilai *rotation offsets*:

	x = 3	**x = 4**	**x = 0**	**x = 1**	**x = 2**
y = 2	25	39	3	10	43
y = 1	55	20	36	44	6
y = 0	28	27	0	1	62
y = 4	56	14	18	2	61
y = 3	21	8	41	45	15

Fig. 8: r[x,y] untuk *rotation offsets*

Lalu, proses Keccak sendiri melibatkan proses sponge di mana terdapat bagian *absorb* dan *squeeze* yang digambarkan dengan skema berikut:

B. HMAC Based One Time Password(HOTP)

HOTP merupakan jenis algoritma pembangkit *one time password* yang berbasis fungsi HMAC tertentu. Standar algoritma ini adalah menggunakan fungsi HMAC berbasis SHA-1. Untuk membangkitkan nilai HOTP, dapat dilakukan seperti 3 langkah di bawah:

1) Membangkitkan nilai HMAC-SHA-1 dengan menggunakan input berupa kunci dan *counter* (disingkat HS). Di mana HS adalah 20-byte string.
2) Membangkitkan 4-byte string (*dynamic truncation*)

$$Sbits = DT(HS)$$

(DT akan dijelaskan kemudian, Sbits akan mengembalikan 31-bit string)
3) Menghitung nilai HOTP

$$Snum = StToNum(Sbits)$$

Definisi dari DT:

$$String = String[0]...String[19]$$

(Definisikan OffsetBits sebagai *low-order* 4 bits dari $String[19]$)

$$Offset = StToNum(OffsetBits)$$

(0 <= Offset <= 15)

$$P = String[Offset]...String[Offset + 3]$$

(P mengembalikan nilai berupa 31-bit string)

III. Hasil dan Pembahasan

Algoritma yang dikembangkan pada tulisan ini adalah algoritma pengganti HOTP yang menggunakan SHA-1 dengan algoritma HOTP dengan basis SHA-3. SHA-3 digunakan karena merupakan standar hash pada saat ini.

A. Pembentukan HMAC-SHA-3

HMAC-SHA-3 digunakan berbasis hash yang dihasilkan oleh SHA-3 ditambah dengan input berupa key dari user. Key ini bebas, tidak ada patokan standar yang digunakan. Sebenarnya hash SHA-3 diklaim oleh pembuatnya sudah memiliki keamanan yang cukup tinggi sehingga tidak dibutuhkannya pembentukan HMAC berbasis SHA-3. Namun, jika seseorang ingin membentuk HMAC-SHA-3, cukup dengan melakukan *prepend* nilai hash dengan kunci dalam operasi bit.

$$HMAC = key + hash$$

B. HMAC-based One Time Password

Nilai *one time password* yang dihasilkan berupa n-digit integer yang dapat diatur pada input dari fungsi itu. *One time password* menghitung nilai dari input berupa sebuah counter yang akan divalidasi dari *server* dengan *client*. Tiap client akan memiliki sebuah kunci rahasia yang berbeda satu sama lain, kunci ini lah yang akan membedakan *sequence* hash yang dihasilkan oleh tiap client. Tiap penaikkan counter pun akan menghasilkan nilai HOTP yang berbeda yang dilakukan sebagai validasi user tertentu terhadap server.

C. Two-factor Authentication

Ada dua buah proses autentikasi yang dijalankan pada proses login user, yaitu:

1) Autentikasi dengan password
   Proses autentikasi dengan password dilakukan dengan cara menyimpan password user dengan bentuk hash pada server. Tiap user akan memiliki username dan password. Setiap kali user menginput password, server akan menghitung nilai hash dari password itu dan mencocokkannya dengan hash pada database server. Jika kedua nilai hash cocok, maka user telah terautentikasi.
2) Autentikasi HOTP
   Untuk proses kedua dilakukan autentikasi dengan HOTP. *Engine* dari HOTP ini akan dimasukkan ke dalam server. Setiap kali user melakukan request untuk membangkitkan HOTP, maka nilai HOTP akan dikirim pada sesuatu yang user punya (misalkan e-mail atau telepon genggam via SMS).
   Server dan client akan memiliki sepasang counter yang identik dan saling tersinkronasi satu sama lain. Tiap client memiliki counter masing-masing yang tidak tergantung kepada nilai counter dari client yang lain. Jika ada user lain yang dengan satu cara dapat mengakses tempat pembangkitan HOTP, user lain itu tidak akan bisa langsung menginput nilai HOTP yang ia dapatkan untuk melakukan autentikasi. Hal ini disebabkan user kedua itu (yang mencuri HOTP user pertama) akan dibaca sebagai client yang berbeda oleh server, salah satu contohnya adalah dengan mengecek ip address dari masing-masing client yang melakukan autentikasi.
   Server dan masing-masing client pun memiliki nilai kunci yang unik antar tiap client, hal ini pun dapat menjadi tingkat keamanan lebih jika server mengalami penyerangan. Nilai kunci ini yang nantinya akan menjadi pembeda dari *sequence* HOTP untuk tiap counter. User A tidak akan mendapat *sequence* HOTP yang sama dengan user B, walaupun nilai counternya sama.

D. HOTP Sequence

Kunci: ”22355”: digit: 6

Counter	HOTP
0	150709
1	216757
2	125093
3	691822
4	839836
5	731375
6	535231
7	616799
8	909354
9	331742

TABLE I: HMAC-SHA-3 Based OTP untuk kunci ”22355” dengan jumlah digit = 6

Kunci: ”22355”: digit: 8

Counter	HOTP
0	42150709
1	42216757
2	59125093
3	92691822
4	95839836
5	01731375
6	51535231
7	46616799
8	96909354
9	45331742

TABLE II: HMAC-SHA-3 Based OTP untuk kunci ”22355” dengan jumlah digit = 8

Kunci: ”satujamsaja”: digit: 6

Counter	HOTP
0	766645
1	022186
2	852001
3	906029
4	769907
5	562913
6	278122
7	764065
8	361701
9	767854

TABLE III: HMAC-SHA-3 Based OTP untuk kunci ”satujamsaja” dengan jumlah digit = 6

Kunci: ”satujamsaja”: digit: 8

Counter	HOTP
0	35766645
1	35022186
2	53852001
3	69906029
4	84769907
5	34562913
6	36278122
7	35765065
8	34361701
9	84767854

TABLE IV: HMAC-SHA-3 Based OTP untuk kunci ”satujamsaja” dengan jumlah digit = 8

Keempat pengujian di atas dilakukan dengan 2 buah kunci dengan masing-masing digit yang berbeda. Bisa terlihat bahwa nilai dari kedua *sequence* untuk tiap digit berbeda dengan berbedanya kunci. Nilai digit keluaran bisa diatur oleh user. Ada satu kelemahan yang masih terdapat di kode ini, di mana bisa terlihat pada pengujian di bawah:

Kunci: ”22354”: digit: 6

Counter	HOTP
0	150709
1	216757
2	125093
3	691822
4	839836
5	731375
6	535231
7	616799
8	909354
9	331742

TABLE V: HMAC-SHA-3 Based OTP untuk kunci ”22354” dengan jumlah digit = 6

Pada pengujian di atas, kunci diubah dari kunci = 22355, digit = 6 (pengujian 1) dengan kunci = 22354, digit = 6. Nilai *sequence* dari HOTP tidak berubah. Sedangkan jika kita uji sekali lagi dengan pendekatan yang berbeda:

Kunci: ”32355”: digit: 6

Counter	HOTP
0	927925
1	216757
2	125093
3	691822
4	839836
5	731375
6	535231
7	616799
8	909354
9	331742

TABLE VI: HMAC-SHA-3 Based OTP untuk kunci ”22354” dengan jumlah digit = 6

Nilai dari *sequence* HOTP berubah drastis jika nilai kunci diubah pada byte paling awal. Sedangkan jika byte kunci diubah satu bit saja, tidak akan mengalami perubahan sama sekali. Hal ini sepertinya terjadi akibat fungsi HMAC-SHA-3 yang terlalu simpel. Fungsi SHA-3 sendiri sudah kuat, namun jika dibuat HMAC dengan algoritma sesimpel menambahkan kunci dengan fungsi hash, dapat terjadi kekurangan pada sistem HMAC-SHA-3 sendiri.

IV. KESIMPULAN DAN SARAN

A. Kesimpulan

Kesimpulannya adalah HOTP yang dihasilkan sudah bisa dipakai sebagai satu lapis keamanan tambahan pada proses autentikasi. HOTP ini jika ditambah dengan proses autentikasi konvensional, dapat menghasilkan dua lapis keamanan yang disebut dengan skema *two-factor authentication*. HOTP yang sekarang beredar masih menggunakan SHA-1 yang relatif tidak terlalu kuat, maka dari itu jika membutuhkan sistem yang lebih kuat akan lebih baik jika diimplementasikan fungsi hash SHA-3 seperti yang dijelaskan pada jurnal ini.

Namun tentu saja, penggunaan SHA-3 akan membutuhkan *cost* yang lebih besar karena proses SHA-3 lebih berat jika dibandingkan dengan proses SHA-1.

B. Saran

Fungsi HMAC-SHA-3 harus dibuat lebih kuat lagi jika satu digit kunci diubah, maka seharusnya *sequence* dari HOTP pun akan berubah drastis. Fungsi HMAC-SHA-3 yang digunakan pada jurnal ini masih lemah untuk dapat memperoleh kekuatan keamanan yang optimal.

REFERENCES

[1] http://keccak.noekeon.org/. *The Keccak sponge function family*.

[2] B. Guido, et al. *The Keccak reference*, version 3.0. 2011.

[3] B. Guido, et al. *The Keccak implementation overview*, version 3.2. 2012.

[4] B. Guido, et al. *The Keccak SHA-3 submission*, version 3.0. 2011.

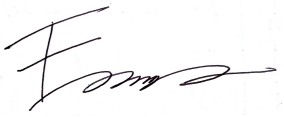
[5] M’Raihi, et al. *RCF4226, HOTP: An HMAC-Based One-Time Password Algorithm*. 2005.

[6] P. Christof and P. Jan. *SHA-3 and The Hash Function Keccak*. Springer.

PERNYATAAN

Dengan ini saya menyatakan bahwa tulisan pada jurnal ini adalah murni tulisan saya sendiri, bukan saduran, bukan plagiasi, atau terjemahan dari jurnal orang lain.

Bandung, 19 Mei 2016



Muhamad Fakhrusy