Implementasi Two Factor Authentication Dan Protokol Zero Knowledge Proof Pada Sistem Login

Willy Sudiarto Raharjo [#1], Ignatia Dhian E.K.Ratri [*2], Henry Susilo [#3]

[# *] *Program Studi Teknik Informatika Fakultas Teknologi Informasi*

Universitas Kristen Duta Wacana

Jl. Dr. Wahidin Sudirohusodo 5-25 Yogyakarta 55224

[1] willysr@ti.ukdw.ac.id

[2] ignatiadhian@ti.ukdw.ac.id

[3] henry.susilo@ti.ukdw.ac.id

Abstract — **This paper describes a login system utilizing Two Factor Authentication and Zero Knowledge Proof using Schnorr NIZK. The proposed system is designed to prevent password leak when being sent over insecure network or when used in an untrusted devices. Zero Knowledge Proof is used for maintaining the confidentiality of the password and Two Factor Authentication is used to secure login process on untrusted devices. The proposed system has been tested and initial results indicates that such system is able to secure the login process without leaking the user's password.**

Keywords— **Authentication, Security, Two Factor Authentication, Password, Zero Knowledge Proof**

I. PENDAHULUAN

Kata sandi (*password*) adalah metode otentikasi yang paling sering digunakan di berbagai sistem keamanan. Kemudahan dalam hal implementasi menjadi faktor utama dari pemanfaatan sistem berbasis *password*. Pengguna juga sudah terbiasa dengan sistem semacam ini sehingga waktu penyesuaian bisa diminimalkan. Di sisi lain, banyaknya penggunaan jaringan yang belum aman (misalnya protokol HTTP) masih menjadi ancaman bagi pengguna mengingat seringkali *password* menjadi satu-satunya mekanisme yang digunakan. Serangan penyadapan (baik aktif maupun pasif) yang terjadi di jaringan seperti *man-in-the-middle-attack* yang dikombinasikan dengan serangan *session hijacking* menjadi sangat praktis untuk digunakan. Firesheep [1] dan sslstrip [2] merupakan salah satu contoh nyata bagaimana pencurian *cookies* dan *password* bisa terjadi melalui jaringan nirkabel dan *platform web* (protokol HTTP), bahkan pada kasus Firesheep, pengguna tidak perlu melakukan apapun kecuali tersambung ke dalam jaringan publik yang sama dengan korban. Semenjak adanya ekspoloitasi memanfaatkan Firesheep, banyak orang dan perusahaan mulai menyadari pentingnya protokol HTTPS. Pada tahun 2013, Facebook mengumumkan bahwa mereka telah mengaktifkan fitur HTTPS untuk semua penggunanya secara *default* meskipun telah memperkenalkan fitur ini sejak 2011 [3], sedangkan Google sudah memperkenalkan HTTPS sejak tahun 2010 [4] meskipun baru mengaktifkannya secara *default* pada tahun 2014 untuk GMail [5].

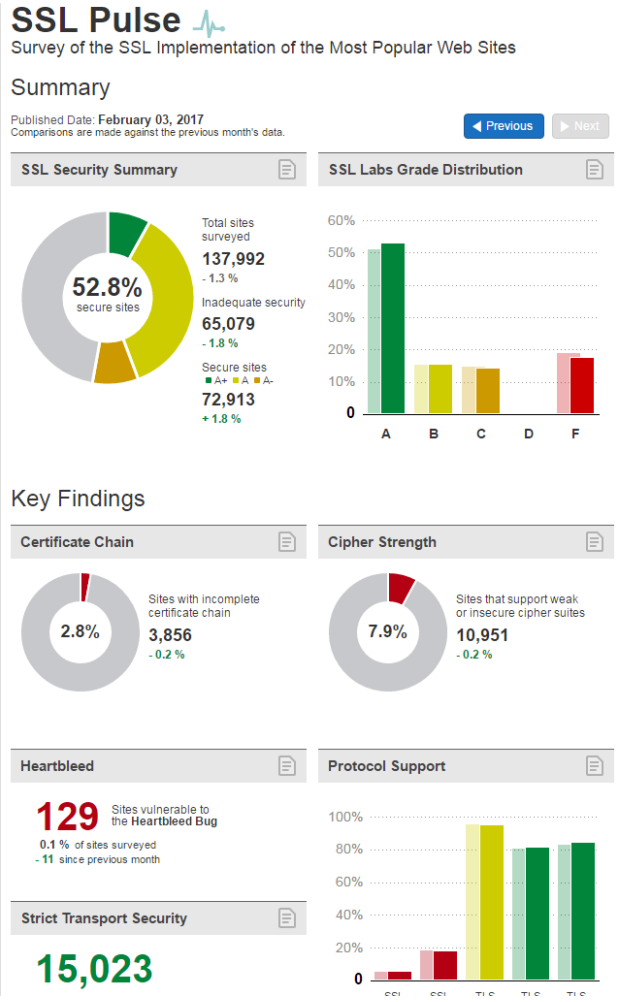
Implementasi protokol HTTPS sendiri bukan solusi yang sempurna, mengingat banyaknya serangan yang nyata dan sudah terbukti menyerang protokol HTTPS (SSL maupun TLS), seperti POODLE [6], BEAST [7], SmackTLS [8], LOGJAM [9], DROWN [10], dan HEARTBLEED [11]. Menurut data dari SSL Pulse per tanggal 3 Februari 2017 [12], masih tercatat 47,2% (65.079 dari 137.992 situs yang disurvei) yang masuk kedalam kategori *inadequate security* yang mengindikasikan bahwa situs masih memiliki masalah dalam pengamanan informasi, terutama berkaitan dengan implementasi protokol HTTPS (Gambar 1). Data tersebut juga menyebutkan bahwa masih ada sekitar 5.4% (7459 situs) dan 17.9% (24.697 situs) yang masih menggunakan protokol SSLv2 dan SSLv3 yang sudah dianggap tidak aman setelah ada berbagai jenis serangan yang praktis dan nyata.

Masalah lain yang muncul adalah banyak pengguna yang menggunakan perangkat yang tidak aman dalam melakukan proses *login*, misalnya menggunakan komputer di tempat publik seperti bandara, perpustakaan umum, dan warnet. Meskipun telah memanfaatkan jaringan yang aman sekalipun, namun *password* akan dapat terbaca oleh komputer yang telah terpasang perangkat lunak *keylogger*.

Two Factor Authentication (TFA) merupakan sebuah metode otentikasi pengguna dimana dua faktor (dari tiga) yang bersifat independen akan digunakan dalam membuktikan adanya klaim bahwa sebuah entitas atau identitas itu asli [13]. Dengan menggunakan TFA, maka *password* bukanlah menjadi *single point of attack* dalam hal akses kepada identitas pengguna. Banyak perusahaan besar telah mulai menggunakan TFA sebagai tambahan pengamanan terhadap serangan *brute-force* terhadap *password*, seperti Google [14], Twitter [15], dan Facebook [16].

Untuk mengurangi resiko kebocoran *password* kepada pihak ketiga, maka protokol *Zero Knowledge Proof* (ZKP) akan digunakan. Protokol ZKP menjamin bahwa pihak *verifier* akan dapat membuktikan identitas pihak *prover* tanpa mempelajari informasi rahasia sedikitpun dari yang diketahui oleh *prover* [17]. Dengan menggunakan protokol ZKP, maka pengguna tidak perlu khawatir bahwa pihak lain (misalnya sebuah *server*) akan dapat menyalahgunakan *password* yang dimilikinya.

Sejauh ini banyak sistem penelitian yang hanya mencoba meneliti penerapan TFA atau ZKP secara independen, namun belum ada penelitian yang mencoba mengkombinasikan TFA dan ZKP dalam upaya untuk memberikan lapisan keamanan yang lebih baik pada sistem login. Penelitian ini mencoba untuk mengkombinasikan penggunaan TFA dan ZKP menggunakan Schnorr NIZK.



Gambar 1 Statistik SSL Pulse per 3 Februari 2017 [12]

II. LANDASAN TEORI

A. *Zero Knowledge Proof*

Zero Knowledge Proof merupakan sebuah protokol kriptografi yang dapat digunakan seseorang (*prover*) untuk membuktikan kepemilikannya akan suatu informasi rahasia atau identitasnya kepada pihak ketiga (*verifier*) tanpa harus membocorkan seluruh informasi rahasia tersebut atau memberikan cara bagi orang lain untuk mengetahui rahasia tersebut. Konsep *Zero Knowledge Proof* digagas oleh Goldwasser, Micali, dan Rackoff pada tahun 1985 dengan istilah *interactive proof-systems* [18]. Sistem ini dikembangkan lagi oleh Uriel Feige, Amos Fiat, dan Adi Shamir dengan konsep "*truly zero knowledge proofs*" dengan cara tidak membocorkan informasi apapun [19].

Ide dasar dari ZKP adalah si *prover* mengetahui sebuah informasi yang hanya diketahui oleh dirinya sendiri, sedangkan *verifier* mencoba membuktikan bahwa *prover* memang mengetahui informasi tersebut dengan cara memberikan sebuah *challenge* yang dilakukan berulang-ulang hingga *verifier* merasa yakin. Apabila *prover* memang mengetahui informasi tersebut, maka dia akan selalu bisa menjawab *challenge* tersebut dengan persentase keberhasilan yang sangat besar.

Terdapat 3 sifat yang harus dipenuhi oleh metode *Zero Knowledge Proof*, yaitu [18]:

1. Completeness

Jika pernyataan *prover* benar, maka *verifier* akan percaya bahwa identitas atau entitas ini memang milik *prover*.

$$\Pr\left(\sigma \xleftarrow{R} \{0,1\}^{n^c}\right);\ Proof \xleftarrow{R} Prover\ (\sigma, x):\ Verifier(\sigma, x, Proof) = 1) = 1 \quad (1)$$

Goldwasser menyatakan bahwa tingkat kebenaran memiliki probabilitas sangat besar, yaitu sebesar $1 - \frac{1}{2^n}$.

2. Soundness

Jika pernyataan *prover* tidak benar, tidak ada *prover* yang mampu meyakinkan *verifier* bahwa pernyataannya benar, kecuali dengan probabilitas yang sangat kecil.

$$\Pr\left(\sigma \xleftarrow{R} \{0,1\}^{n^c}\right);\ (x,\ Proof) \xleftarrow{R} Adversary\ (\sigma):\ Verifier(\sigma, x, Proof) = 1) < 2^{-n} \quad (2)$$

Goldwasser menyatakan bahwa tingkat kebenaran memiliki probabilitas sangat kecil, yaitu sebesar $\frac{1}{2^n}$.

3. Zero-knowledge

Jika pernyataan *prover* benar, *verifier* tidak akan dapat mempelajari apapun kecuali fakta bahwa pernyataan yang diberikan *prover* benar.

Apabila terdapat sebuah algoritma S yang efisien dimana $\forall x \in L_n$ dan sejumlah nilai n yang cukup besar, maka

$$View(n, x) = S(1^n, x) \quad (3)$$

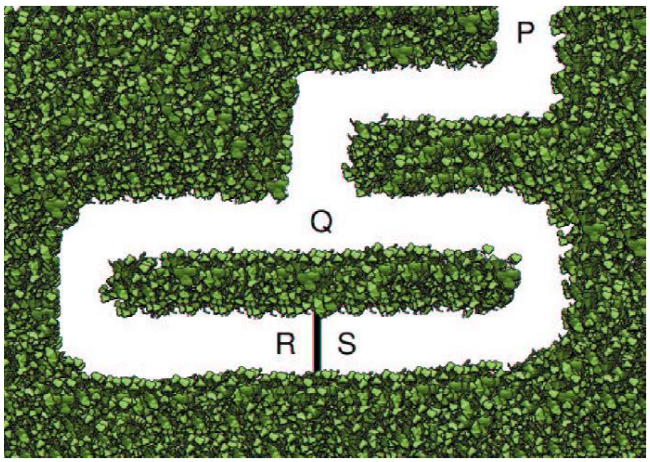
dimana

$$View(n, x) = \left\{\sigma \xleftarrow{R} \{0,1\}^{n^c}\right\}; Proof \xleftarrow{R} Prover\ (\sigma, x): (\sigma, Proof) \quad (4)$$

Ilustrasi kasus dari penerapan ZKP yang terkenal

adalah gua Alibaba (Gambar 2). Pada gua tersebut terdapat sebuah pintu rahasia yang berada diantara R dan S. Proses pembuktian bekerja sebagai berikut [20]:

1. *Verifier* berada pada posisi P, sedangkan *prover* berada di posisi Q.
2. *Prover* akan berjalan menuju ke pintu dengan menggunakan jalur R atau S sesuai dengan keinginan *prover* dan dilakukan secara acak.
3. Setelah itu verifier akan berjalan ke posisi Q dan meneriakkan jalur R atau S sesuai dengan keinginannya, kemudian *prover* berjalan ke arah posisi Q menggunakan jalur yang telah disebutkan oleh *verifier*.
4. Jika *prover* mengetahui kata rahasia untuk membuka pintu, maka *prover* akan mampu mengikuti semua permintaan *verifier*, tetapi jika *prover* ingin mengelabui *verifier*, maka *prover* memiliki probabilitas 50% untuk berhasil.



Gambar 2 Gua Alibaba [21]

Salah satu protokol *proof of knowledge* yang paling terkenal adalah protokol Schnorr yang berbasiskan pada konsep *number theory*.

B. Number Theory

Untuk mempermudah pembahasan, digunakan notasi sebagai berikut:

N = bilangan *integer* > 0

$\mathbb{Z}_N$ = sekumpulan bilangan integer antara {0, N-1}

Contoh $\mathbb{Z}_5$ = {0, 1, 2, 3, 4}

gcd = bilangan *integer* terbesar yang merupakan pembagi dari dua angka.

Contoh gcd(14, 21) = 7

Misalkan *a* adalah bilangan bulat dan *N* adalah bilangan bulat > 0. Operasi *a mod N* memberikan sisa *m* jika *a* dibagi dengan *N*. Bilangan *m* disebut bilangan modulus atau *modulo* dan hasil aritmetika *modulo N* terletak di dalam himpunan {0, 1, 2, …., N-1}. Contoh *12 mod 7* = 5.

Misalkan *b* adalah bilangan bulat, jika hasil dari *(a-b) mod N* adalah 0 maka dapat disebutkan *a* dan *b congruent modulo N*. Apabila untuk sebuah bilangan *integer* b terdapat sebuah bilangan *integer* c sehingga didapatkan persamaan *bc = 1 mod N*, maka b disebut sebagai *invertible modulo* N dan c adalah *inverse* dari *b modulo N* [22].

Untuk bisa menentukan apakah sebuah elemen memiliki *multiplicative inverse modulo N*, bisa dengan menggunakan algoritma Euclidean [22].

```
Input: Integer a, b, dengan
a ≥ b > 0
Output: GCD dari a dan b
if b divide a
  return b
else
  return GCD(b, [a mod b])
```

Algoritma Eucledian

Misalkan terdapat N dan $a \in \mathbb{Z}_N$ dengan *gcd(a, N) = 1*, maka terdapat *integer* X, Y dengan *Xa + YN = 1*. Hal ini mengindikasikan bahwa [*X mod N*] adalah *multiplicative inverse* dari a. *Integer* X dan Y bisa ditemukan dengan cepat dengan menggunakan algoritma Extended Eucledian [22].

```
Input: Integer a, b a ≥ b > 0
Output: (d, X, Y) dengan d = gcd(a,b)
        dan Xa + Yb = d
if b divide a
  return (b, 0, 1)
else
  hitung integer q, r dengan a = qb + r
  dan 0 < r < b
  (d, X, Y) := eGCD(b, r)
  return (d, Y, X-Yq)
```

Algoritma Extended Eucledian

Multiplicative inverse merupakan poin penting dalam skema identifikasi Schnorr karena menggunakan konsep kunci publik dan kunci privat sehingga harus dipastikan bahwa parameter yang digunakan harus memiliki *inverse*. Tanpa adanya *inverse*, maka proses pembuktian tidak akan bisa dilakukan. Hal ini sesuai dengan persamaan

$$a\, a^{-1} = 1 \bmod p \quad (5)$$

Pada tahun 1640, Fermat membuat teorema tentang bagaimana bilangan prima bisa digunakan untuk menghitung *inverse* [23] :

$$\forall x \in \left(\mathbb{Z}_p\right)^* = x^{p-1} = 1 \bmod Z_p \quad (6)$$

C. Schnorr NIZK Proof

Schnorr *NIZK proof* adalah primitif dari ZKP yang memungkinkan sebuah entitas untuk membuktikan sebuah informasi tentang pemahaman sebuah nilai logaritma diskrit tanpa membocorkan informasi apapun tentang nilainya. Skema ini mengasumsikan bahwa ada 2 pihak yang akan berinteraksi secara interaktif, yaitu Alice sebagai *Prover* dan Bob sebagai *Verifier* [24].

Alice pertama-tama akan membuat kunci publiknya terlebih dahulu. Proses pembentukan kunci dilakukan sebagai berikut:

1. Memilih 2 bilangan prima, p dan q , dan sebuah nilai g yang memenuhi 2 syarat berikut:
   a. Bilangan q adalah faktor prima dari (p-1). Ini berarti gcd(q, p-1) tidak boleh bernilai 1.
   b. Harus memenuhi persamaan
   $$g^q = 1 \ (mod\ p) \quad (7)$$
2. Pilih sebuah nilai x, dimana x < q. (dimana x adalah kunci privat)
3. Hitung nilai X dengan rumus
   $$X = g^x \ mod\ p \quad (8)$$

Alice akan mempublikasikan kunci publiknya (X) dan proses verifikasi bisa dimulai. Protokol ini bekerja dalam 3 tahapan [24]:

1. Alice memilih sebuah bilangan v secara *uniform* dari himpunan [0, q-1] dan menghitung
   $$V = g^v mod\ p \quad (9)$$
   Nilai dari V dikirimkan ke Bob.
2. Bob memilih sebuah *challenge* c secara *uniform* dari himpunan [0, 2^{t-1}] dimana t adalah panjang bit dari *challenge* (misalnya 80). Bob mengirimkan nilai c ke Alice.
3. Alice akan menghitung b dan mengirimkannya ke Bob.
   $$b = v - xc \ mod\ q \quad (10)$$

Bob akan memeriksa apakah persamaan berikut sesuai:

$$V = g^b * X^c \ mod\ p \quad (11)$$

Sebagai contoh, digunakan dua buah bilangan prima p = 1447, q = 241 dan g = 1338 yang memenuhi kedua syarat yang telah dijelaskan pada poin pertama untuk proses pembentukan kunci:

1. Pilih sebuah nilai x = 8 (x lebih kecil dari q yang bernilai 241).
2. Hitung: $X = g^x \bmod p$ sesuai (8)
3. $X = 1338^8 \bmod 1447$
   X = 249
4. Didapatkan
   Kunci privat: x = 8
   Kunci publik: X = 249.

Untuk proses verifikasi:

1. Alice memilih nilai v = 235 dan menghitung:
   $V = g^v \bmod p$ sesuai (9)
   $V = 1338^{235} \bmod 1447$
   V = 1332
2. Alice mengirimkan V = 1332 kepada Bob.
3. Bob mengirimkan sebuah nilai c = 19 (dipilih secara acak) kepada Alice.
4. Alice menghitung:
   b = (v - xc) mod q sesuai (10)
   b = (235 - 8*19) mod 241
   b = 83
5. Alice mengirimkan b = 83 kepada Bob.
6. Bob memverifikasi menggunakan (11)
   $1332 = 1338^{83} * 249^{19} \ mod\ 1447$
   1332 = 1332
   Proses otentikasi berhasil.

Bob disini tidak mendapatkan informasi apapun tentang nilai x yang hanya diketahui Alice, namun Bob bisa memverifikasi identitas atau klaim Alice. Apabila Bob masih tidak mempercayai Alice, maka Bob berhak untuk mengirimkan sebuah nilai lain, misalnya c = 54. Alice akan menghitung:

b = (235 - 8*54) mod 241
b = 44 dan dikirimkan ke Bob
Bob akan memverifikasi nilai tersebut
$1332 = 1338^{44} * 249^{54} \bmod 1447$
1332 = 1332

Proses ini bisa dilakukan berulang-ulang sampai Bob merasa yakin bahwa Alice memang benar-benar Alice dan proses bisa berhenti. Jumlah iterasi yang disarankan adalah sebanyak 20 kali [25].

D. Multi-Factor Authentication

Multi-Factor Authentication merupakan sebuah metode otentikasi yang memanfaatkan lebih dari satu faktor untuk menentukan identitas pengguna. Saat ini 3 faktor yang umum dijumpai adalah "*something you know*", "*something you have*", dan "*something you are*" [26]. Contoh dari "*something you know*" adalah *password* atau PIN (*personal identification number*); contoh dari "*something you have*" adalah *Smart Card* dan *Token*; sedangkan contoh dari "*something you are*" adalah penggunaan *biometric* untuk penindaian seperti sidik jari, retina, iris, telapak tangan, dan suara.

Two-Factor Authentication merupakan sebuah metode otentikasi pengguna dimana dua faktor (dari tiga) yang bersifat independen akan digunakan dalam membuktikan adanya klaim bahwa sebuah entitas atau identitas itu asli. Penggunaan *Two-Factor Authentication* akan dapat

mengurangi resiko seorang *adversary* dapat masuk ke sebuah sistem dengan menggunakan identitas kita karena selain harus mengetahui *password* yang kita gunakan, si *adversary* juga harus mendapatkan informasi kedua yang bisa dihasilkan dari sumber yang berbeda.

Salah satu alasan kenapa *password* dianggap kurang kuat sehingga diperlukan mekanisme tambahan untuk mengamankan adalah karena kecenderungan pengguna untuk memilih *password* yang mudah diingat dan juga banyak mengabaikan aturan yang umumnya direkomendasikan, misalnya melakukan *password sharing* dengan anggota keluarga atau rekan kerja, menggunakan informasi yang mudah ditebak dalam membuat *password*, hingga menempelkan informasi *password* pada tempat publik sehingga bisa dengan mudah diketahui oleh orang lain.

Pada kebanyakan sistem yang ada saat ini digunakan kombinasi antara *username*/*password* dan juga kode *OTP* (*One-Time Password*) yang dikirimkan melalui SMS ke perangkat digital (*smartphone*/*tablet*) atau dihasilkan melalui aplikasi pihak ketiga, misalnya Google Authenticator atau Authy. Salah satu kelemahan utama dari penggunaan *OTP* adalah rentan terhadap serangan *Phising* dan *Man-in-the-middle-Attack* [27]. Solusi yang lebih baik adalah dengan menggunakan perangkat keras dari pihak ketiga untuk mengurangi resiko penggunaan *password*, seperti Yubico (Gambar 3), Nitrokey (Gambar 4), atau RSA SecurID (Gambar 5).

Penggunaan perangkat keras mampu mengurangi waktu dari proses *sign-in* pengguna maupun beban yang harus ditanggung oleh organisasi dalam pengelolaannya [28]. Facebook dan Twitter pun baru-baru ini mengumumkan bahwa mereka juga sudah mendukung penggunaan *security key* untuk memberikan pengamanan lebih kepada pengguna [29] [30].

E. OTP (One-Time Password)

OTP merupakan pengembangan dari S/Key *One-Time Password System* yang dikembangkan oleh Bellcore [31] [32]. OTP dikembangkan untuk mengurangi resiko *password* dikirimkan kepada pihak lain melalui media yang tidak aman serta mencegah terjadinya *replay attack*, yaitu sebuah serangan yang mencoba untuk menggunakan sebuah informasi yang pernah disimpan dari sesi sebelumnya untuk digunakan di periode waktu yang berbeda. Karena *OTP* hanya bisa digunakan satu kali dan memiliki periode waktu yang terbatas, maka *OTP* sangat cocok untuk diimplementasikan pada sistem *login* berbasis *web*, terutama pada komputer publik dimana seringkali pengguna lupa untuk menghapus *cookies* atau *session*nya pada *browser*. Hal ini bisa mengurangi resiko terjadinya serangan *session hijacking*.

Terdapat dua entitas yang bermain dalam skema *OTP*, yaitu *generator* yang bertugas untuk menghasilkan *OTP* dari kombinasi *passphrase* pengguna dan informasi yang disediakan pada sebuah *challenge* dari server, serta *server* yang bertugas untuk mengirimkan *challenge* serta melakukan verifikasi terhadap nilai *OTP* yang diterima. Panjang dari keluaran yang dihasilkan adalah sebesar 64 bit yang kemudian diproses lagi sehingga menjadi 6-8 digit angka. Teknik yang digunakan dalam menghasilkan nilai *OTP* dicetuskan oleh Leslie Lamport [33].

OTP sendiri dibagi lagi menjadi dua kategori besar, yaitu HOTP (*HMAC-based OTP*) yang diatur pada RFC 4226 [34] dan TOTP (*Time-based OTP*) yang diatur pada RFC 6238 [35].

Gambar 3 Perangkat Yubico

Gambar 4 Perangkat NitroKey

e-ISSN : 2443-2229

Gambar 5 Perangkat SecurID

Algoritma HOTP menggunakan konsep dasar sebuah nilai *counter* yang selalu meningkat nilainya dan menggunakan Algoritma HMAC-SHA1.

Berikut langkah-langkah untuk menghasilkan nilai HOTP:

1. Menghasilkan sebuah nilai HMAC-SHA1; misalkan HS = HMAC-SHA-1(K, C)
2. Menghasilkan 4 *byte string*; Sbit = DT(HS)
   Fungsi DT akan menghasilkan 31-bit *string*
3. Menghitung nilai HOTP
   Mengkonversi Sbit menjadi angka dalam himpunan $\{0, \ldots, 2^{31} - 1\}$:

$$Snum = StToNum(Sbit) \quad (10)$$

$$D = Snum \bmod 10^{digit} \quad (11)$$

digit merupakan panjang yang diharapkan.

Algoritma TOTP merupakan pengembangan dari HOTP dimana algoritma yang digunakan lebih bervariasi, misalnya menggunakan fungsi HMAC-SHA-256 atau HMAC-SHA-512, yang berbasiskan pada fungsi hash SHA-256 atau SHA-512.

TOTP didefinisikan sebagai

$$TOTP = HOTP(K, T) \quad (12)$$

Dimana T merupakan sebuah *integer* dan didefinisikan sebagai jumlah *step* antara waktu T0 dan waktu unix aktual

$$T = \frac{(Current\ Unix\ time - T0)}{X} \quad (13)$$

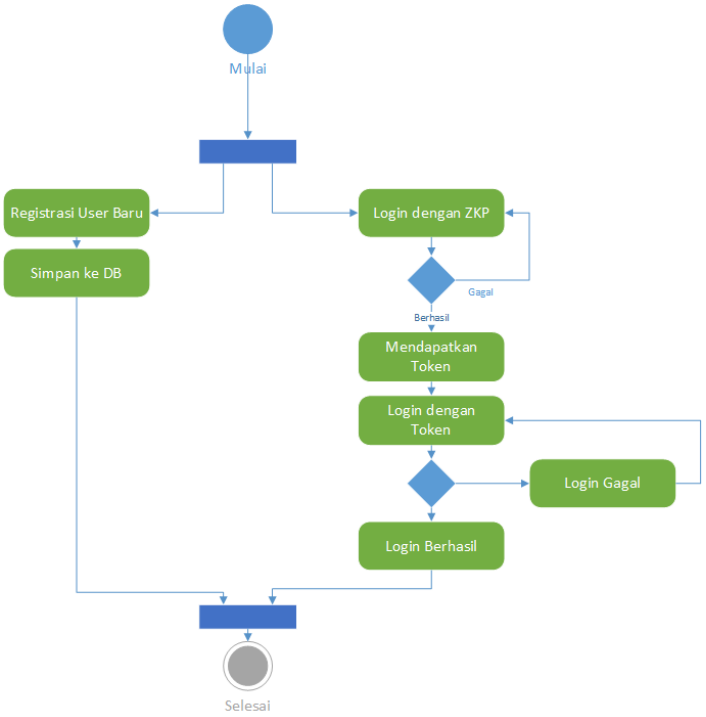
Sebagai contoh T0 = 0 dan *time step* X = 30, maka untuk $0 < T \leq 59$ akan menghasilkan T = 1, sedangkan detik ke-60 akan menghasilkan T = 2.

Masalah utama dari penggunaan TOTP adalah adanya kemungkinan bahwa jam pada setiap komputer bisa bergeser (*clock drift*), sehingga harus diberikan batas toleransi yang cukup. Batas yang disarankan adalah 2 siklus kebelakang dengan *default time step* sebanyak 30 detik.

III. METODOLOGI PENELITIAN

Proses sistem *login* yang dibangun pada penelitian ini mengacu pada Gambar 6, dimana pengguna bisa melakukan pendaftaran sebagai anggota baru atau melakukan *login*. Untuk proses pendaftaran tidak ada informasi selain nama pengguna dan *password* yang diminta mengingat penelitian ini menekankan pada penerapan ZKP dan TFA pada sistem *login* dan bukan fitur lengkap dari sebuah aplikasi.

Proses *login* akan terbagi menjadi dua tahap, yaitu proses *login* dengan menggunakan ZKP dimana pengguna akan melakukannya pada perangkat atau sistem yang aman dan diasumsikan *trusted*. Hasil dari proses ini adalah sebuah *token* yang bersifat sementara (*default* 60 detik) yang bisa digunakan untuk melakukan proses *login* pada sistem yang tidak aman atau perangkat yang *untrusted*. Setelah *token* kadaluarsa (*expired*), maka *token* tidak akan dapat digunakan kembali. *Token* juga hanya bisa digunakan untuk satu sesi *login* saja dan tidak akan bisa digunakan pada sesi *login* lainnya.



Gambar 6 Diagram Alir Sistem

Untuk bisa memastikan bahwa sistem berjalan dengan baik, maka perlu dipersiapkan lima skenario pengujian yang meliputi:

1. Pengguna *login* dengan menggunakan *password* yang salah pada fase pertama (ZKP)
   Output yang diharapkan: gagal login
2. Pengguna *login* dengan menggunakan *password*

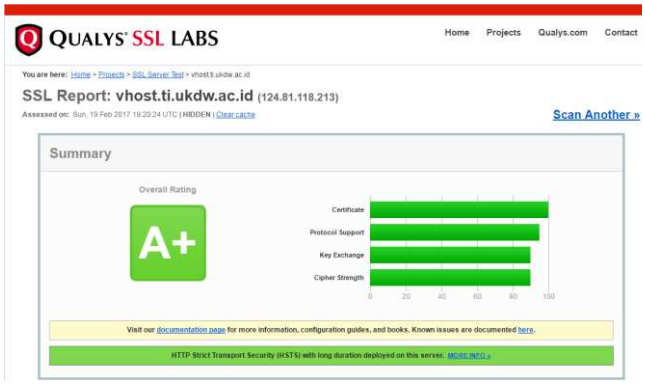
yang benar pada fase pertama (ZKP) dan memasukkan kode *token* yang benar pada fase kedua (TFA)
Output yang diharapkan: berhasil login

3. Pengguna *login* dengan menggunakan *password* yang benar pada fase pertama (ZKP) dan memasukkan kode *token* yang salah pada fase kedua (TFA)
   Output yang diharapkan: gagal login
4. Pengguna *login* dengan menggunakan *password* yang benar pada fase pertama (ZKP) dan memasukkan kode *token* yang kadaluarsa pada fase kedua (TFA)
   Output yang diharapkan: gagal login
5. Pengguna *login* dengan menggunakan *password* yang benar pada fase pertama (ZKP) dan memasukkan kode *token* yang digunakan pada sesi sebelumnya pada fase kedua (TFA)
   Output yang diharapkan: gagal login

Setiap pengujian akan dilakukan pada 5 pengguna yang berbeda dan pada masing-masing pengguna dilakukan sebanyak 20 iterasi sesuai dengan rekomendasi yang disyaratkan [25] pada skenario 2 dan 3.

IV. HASIL DAN ANALISIS

Untuk men-simulasikan sebuah lingkungan yang aman, sistem ini dibangun diatas sebuah *server* yang sudah dilengkapi dengan protokol HTTPS dengan sertifikat digital yang dikeluarkan oleh LetsEncrypt [36] [37] dan sudah diujikan dengan memanfaatkan layanan SSL Test dari SSL Labs [1] (Gambar 7). Dengan demikian, maka tidak ada *adversary* yang dapat melakukan penyadapan terhadap data yang keluar maupun masuk ke *server* ini.



Gambar 7 Hasil Pengujian *Server* oleh SSL Test

Sistem dibangun berbasiskan *platform web* menggunakan bahasa pemrograman PHP dan memanfaatkan *library* external, yaitu random_compat[2]. Sistem ini memiliki dua antarmuka, yaitu antarmuka untuk login menggunakan ZKP (diasumsikan hanya dilakukan pada *trusted network* atau *device*) dan antarmuka login berbasis *token* (diasumsikan *untrusted network*).

Halaman pertama adalah tampilan untuk menu registrasi pengguna baru (Gambar 8). Disini pengguna cukup memasukkan *username* dan *password*. Karena penelitian ini berfokus pada penerapan ZKP dan TFA, maka tidak ada data lain yang dikumpulkan.

Pada saat pengguna masuk di halaman registrasi ini, sistem akan menghasilkan 3 buah nilai *random*, yaitu p, q, dan g sesuai (5) dan (6). Nilai tersebut akan dikirimkan ke *server* bersamaan dengan informasi *username* dan *password* untuk pembentukan kunci privat dan kunci publik. Pada posisi ini, informasi tentang *token* masih bernilai kosong karena belum adanya proses *login*.

Gambar 8 Proses Pendaftaran Pengguna Baru

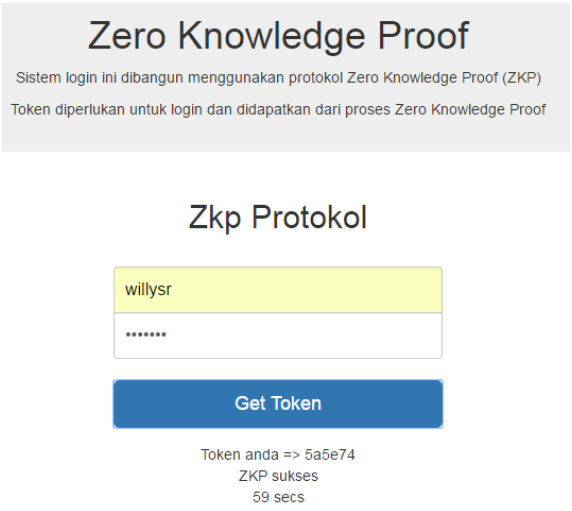
Langkah berikutnya adalah menggunakan halaman login ZKP (Gambar 9). Disini pengguna cukup memasukkan *username* dan *password* yang digunakan pertama kali dan sistem akan menjalankan pengujian identifikasi pengguna menggunakan ZKP dengan protokol Schnorr sebanyak 20 iterasi. Apabila semua iterasi berhasil dilalui dengan benar, maka *token* sementara akan dibuat dan disimpan didalam *database* serta diberikan kepada pengguna (Gambar 10). Kolom waktu merupakan waktu pembuatan *token* yang didapatkan dari *unix time stamp*. Panjang *token* yang digunakan adalah sebesar 6 digit dan merupakan kombinasi alphanumerik (alphabet dan numerik). Sistem juga akan melakukan perhitungan mundur selama 60 detik (2 siklus dengan *time step* 30 detik) yang mengindikasikan lama waktu sebelum *token* dianggap kadaluarsa oleh sistem. Nilai itu dirasa lebih dari cukup mengingat aplikasi pihak ketiga seperti Google Authenticator, Microsoft Authenticator dan

[1] https://www.ssllabs.com/ssltest/

[2] https://github.com/paragonie/random_compat

e-ISSN : 2443-2229

Authy juga menggunakan angka 60 sebelum melakukan perubahan pada nilai TOTP yang dihasilkan.



Gambar 9 Login Menggunakan ZKP

Sistem belum memanfaatkan fasilitas seperti *crontab* untuk menghapus *token* yang berusia lebih dari 60 detik secara otomatis. Penghapusan *token* baru akan dilakukan pada saat pengguna berhasil melakukan proses *login* atau sistem mendeteksi adanya usaha untuk menggunakan *token* yang sudah kadaluarsa. Penggunaan *crontab* akan lebih efisien karena pada *token* yang kadaluarsa akan secara otomatis terhapus dari *database* setiap menitnya. Hal ini juga mampu mencegah terjadinya penumpukan *token* yang kadaluarsa apabila terjadi situasi yang tidak lazim seperti *server* harus mengalami proses *reboot*.

Apabila pengguna berhasil memasukkan kode *token* dengan benar, maka sistem akan memverifikasi dengan sempurna (Gambar 11) dan kode token akan dihapus dari *database* (Gambar 12). Hal ini untuk mencegah penggunaan *token* berkali-kali.

Apabila pengguna salah dalam memasukkan *username* ataupun *token*, maka akan muncul pesan kesalahan (Gambar 13). Apabila pengguna mencoba melakukan *login* dengan *token* yang sudah kadaluarsa (lebih dari 60 detik), maka akan muncul pesan peringatan seperti pada (Gambar 14) dan data *token* akan dihapus dari *database*.

	id	username	token	status	waktu
Edit Copy Delete	27	zkp	f487a3	1	1465901403
Edit Copy Delete	28	henrysusilo	3de8f2	1	1472008614
Edit Copy Delete	36	henrys	07df8b	1	1478587171
Edit Copy Delete	44	willysr	5a5e74	1	1487528349

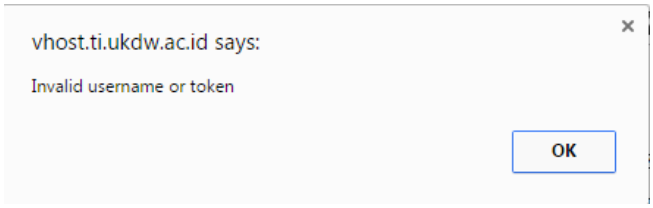
Gambar 10 Informasi *Token* pada Database

Gambar 11 Proses Login TFA Berhasil

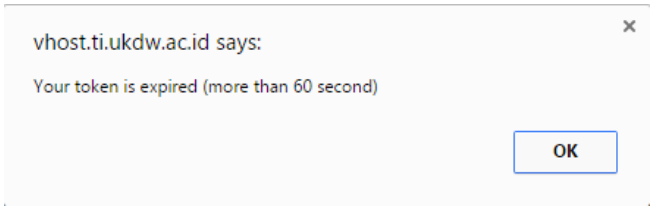
Pada percobaan yang dilakukan dengan menggunakan 5 skenario, didapatkan hasil bahwa sistem mampu menyelesaikan skenario yang memastikan bahwa hanya pengguna yang memasukkan kode *token* yang dihasilkan dari proses *login* ZKP ke dalam *login* TFA merupakan pengguna yang sah (skenario 2) dan tidak untuk pengguna yang gagal pada skenario yang lain (skenario 1, 3, 4, dan 5). Hal ini mengindikasikan bahwa kombinasi sistem ZKP dan TFA yang dibuat telah berjalan dengan baik.

	id	username	token	status	waktu
Edit Copy Delete	27	zkp	f487a3	1	1465901403
Edit Copy Delete	28	henrysusilo	3de8f2	1	1472008614
Edit Copy Delete	36	henrys	07df8b	1	1478587171

Gambar 12 Data *Token* Dihapus Setelah Berhasil Login



Gambar 13 Login TFA Gagal



Gambar 14 *Token* Kadaluarsa

Penggunaan 6 digit untuk panjang *token* dan waktu 60 detik tidak menjadi beban bagi pengguna dalam menjalankan proses login ini. Hal ini dapat terlihat bahwa

tingkat keberhasilan pengguna dalam melakukan proses *login* ini untuk skenario 2 adalah sebesar 100%. Hal yang sama juga didapatkan pada hasil penelitian lainnya [38] [39] [40] meskipun pada ketiga penelitian ini dilakukan dengan menggunakan *smartphone*.

Penambahan lapisan keamanan dengan menggunakan TFA memang menyebabkan waktu untuk login menjadi relatif lebih lambat karena harus menyelesaikan dua proses terlebih dahulu, namun dari sisi keamanan, banyak peningkatan yang dirasakan.

Pemanfaatan protokol HTTPS juga akan meningkatkan tingkat keamanan mengingat *adversary* tidak akan sanggup untuk membaca data yang keluar dan masuk di jaringan.

V. KESIMPULAN

Berdasarkan hasil implementasi dan analisis yang telah dilakukan, maka dapat disimpulkan sebagai berikut :

1. Proses verifikasi *login* dengan menggunakan ZKP dengan protokol Schnorr dan TFA bisa diterapkan dengan baik dengan tingkat keberhasilan dalam mendeteksi pengguna baik yang legal maupun *adversary* sebesar 100% dari 5 pengguna dengan 5 skenario yang berbeda dan 20 iterasi pada skenario yang berhubungan dengan ZKP, sehingga kerahasiaan *password* bisa tetap terjaga. *Secret key* dan *public key* menjadi kunci utama yang dimiliki oleh user untuk melakukan verifikasi. Jika ada pihak yang tidak memiliki *secret key* dan *public key* namun mencoba untuk melakukan verifikasi, maka sistem tidak akan memberikan hak akses untuk pengguna tersebut.
2. Pengguna tidak dapat *login* secara ilegal, karena saat proses verifikasi sistem memerlukan *secret key* yang hanya diketahui oleh pengguna. Jika pengguna mencoba memasukkan *secret key* yang tidak cocok maka sistem tidak akan memberikan hak akses kepada pengguna, sehingga hasil yang dikeluarkan oleh sistem akan memberikan keputusan yang tepat (memberikan akses atau tidak). Keberhasilan sistem memastikan kecocokan *secret key* ini didapatkan melalui hasil penghitungan dengan Schnorr NIZK.
3. Pemanfaatan HTTPS akan meningkatkan tingkat keamanan dengan memastikan bahwa data yang dikirimkan dari *server* ke *client* dan *client* ke *server* selalu dalam keadaan terenkripsi.

Pengembangan untuk kedepannya bisa dengan mengintegrasikan penggunaan perangkat keras untuk proses TFA atau mengintegrasikannya dengan sistem *Single Sign-On* yang berbasis TFA.

UCAPAN TERIMA KASIH

Penulis mengucapkan banyak terima kasih kepada Fakultas Teknologi Informasi dan Pusat Pelatihan dan Layanan Komputer (PPLK) Universitas Kristen Duta Wacana yang telah menyediakan infrastruktur demi kelancaran penelitian ini.

DAFTAR PUSTAKA

[1] (2010) Website Firesheep [Online]. Tersedia: https://codebutler.github.com/firesheep
[2] (2009) Website Proyek SSLStrip [Online]. Tersedia: https://github.com/moxie0/sslstrip
[3] S. Renfro (2013) “Secure browsing by default” [Online]. Tersedia: https://www.facebook.com/notes/facebook-engineering/secure-browsing-by-default/10151590414803920/
[4] S. Schillace (2010) "Default https access for Gmail" [Online]. Tersedia: https://gmail.googleblog.com/2010/01/default-https-access-for-gmail.html.
[5] N. Lidzborski (2014) “Staying at the forefront of email security and reliability: HTTPS-only and 99.978% availability” [Online]. Tersedia: https://gmail.googleblog.com/2014/03/staying-at-forefront-of-email-security.html
[6] B. Möller, et al. (2014) “This POODLE Bites: Exploiting The SSL 3.0 Fallback” [Online]. Tersedia: https://www.openssl.org/~bodo/ssl-poodle.pdf.
[7] T. Duong (2011) “BEAST” [Online]. Tersedia: https://vnhacker.blogspot.co.id/2011/09/beast.html
[8] B. Beurdouche, et al (Mei, 2015) “A Messy State of the Union: Taming the Composite State Machines of TLS” dalam IEEE Symposium on Security and Privacy [Online]. Tersedia: http://www.ieee-security.org/TC/SP2015/papers-archived/6949a535.pdf
[9] D. Adrian, et al (Oktober, 2015) “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice” dalam 22nd ACM Conference on Computer and Communications Security (CCS ’15) [Online]. Tersedia: https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf
[10] N. Aviram, et al (Agustus, 2016) “DROWN: Breaking TLS using SSLv2” dalam 25th USENIX Security Symposium [Online]. Tersedia: https://drownattack.com/drown-attack-paper.pdf
[11] (2014) Website Heartbleed Bug [Online]. Tersedia: http://heartbleed.com/
[12] (2017) Website SSL Pulse [Online]. Tersedia: https://www.trustworthyinternet.org/ssl-pulse/
[13] M. De Soete (2011), “Two-Factor Authentication,” dalam Encyclopedia of Cryptography and Security, Springer US, hal. 1341.
[14] N. Shah (2011) “Advanced sign-in security for your Google account” [Online]. Tersedia: https://gmail.googleblog.com/2011/02/advanced-sign-in-security-for-your.html
[15] Jimio (2013) “Getting started with login verification” [Online]. Tersedia: https://blog.twitter.com/2013/getting-started-with-login-verification
[16] A. Song (2011) “Introducing Login Approvals” [Online]. Tersedia: https://www.facebook.com/notes/facebook-engineering/introducing-login-approvals/10150172618258920/
[17] Feige, U., et al. (1988) “Zero-knowledge proofs of identity” dalam Journal of Cryptology Vol 1, Issue 2, hal 77.
[18] S. Goldwasser, et al (Desember 1985) “The knowledge complexity of interactive proof-systems” dalam STOC '85 Proceedings of the seventeenth annual ACM symposium on Theory of computing, hal. 291-304.
[19] U. Feige, et al (Juni 1988), “Zero-knowledge proofs of identity” dalam Jurnal of Cryptology, Springer, 1: 77. doi:10.1007/BF02351717

[20] T.N. Situngkir "Implementasi zero knowledge proof dengan feige fiat shamir dan quadratic linear congruential generator" Skripsi, Ilmu Komputer, Universitas Sumatera Utara, Medan, Indonesia, 2013
[21] D. Raffo, "Digital certificates and the feige-fiat shamir zero knowledge protocol. France : Traineeship report, 2012.
[22] J. Katz dan Y. Lindel, "Introduction to Modern Cryptography", Boca Raton, FL: CRC Press, 2015.
[23] J.S. Kraft, L.C. Washington, "An Introduction to Number Theory with Cryptography", Boca Raton, FL: CRC Press, 2014
[24] F. Hao (2016) "Schnorr NIZK Proof: Non-interactive Zero Knowledge Proof for Discrete Logarithm version 5" [Online]. Tersedia: https://tools.ietf.org/html/draft-hao-schnorr-05
[25] W.S. Raharjo dan D. Susanti, "Implementasi Zero Knowledge Proof Menggunakan Protokol Feige Fiat Shamir Untuk Verifikasi Tiket Rahasia" Jurnal ULTIMATICS UMN, Vol 2, No 2, 2015, hal 91-97.
[26] M. Stamp, "Information Security: Principles and Practices" San Jose, CA: Wiley, 2011, hal 276.
[27] J.S., Railton, K. Kleemola (2015) "London Calling: Two-Factor Authentication Phishing From Iran" [Online]. Tersedia: https://citizenlab.org/2015/08/iran_two_factor_ phishing/
[28] J. Lang, et al (Februari, 2016) "Security Keys: Practical Cryptographic Second Factors for the Modern Web" dalam Financial Cryptography and Data Security 2016 [Online]. Tersedia: http://fc16.ifca.ai/preproceedings/25_Lang.pdf
[29] B. Hill (Januari, 2017) "Security Key for safer logins with a touch" [Online]. Tersedia: https://www.facebook.com/notes/facebook-security/security-key-for-safer-logins-with-a-touch/10154125089265766/
[30] M. Coates (Juni, 2016) "Keeping your account safe" [Online]. Tersedia: https://blog.twitter.com/2016/keeping-your-account-safe
[31] Neil H. (Februari, 1994) "The S/KEY One-Time Password System", Proceedings of the ISOC Symposium on Network and Distributed System Security, San Diego, CA
[32] N. Haller, (1995) "The S/KEY One-Time Password System", RFC 1760 [Online]. Tersedia https://tools.ietf.org/html/rfc1760.
[33] L. Lamport (November, 1981) "Password Authentication with Insecure Communication" dalam Communications of the ACM 24.11 hal. 770-772
[34] D. M'Raihi, et al (Desember, 2005) "HOTP: An HMAC-Based One-Time Password Algorithm" RFC 4226 [Online]. Tersedia https://tools.ietf.org/html/rfc4226
[35] D. M'Raihi, et al (Mei, 2011) "TOTP: Time-Based One-Time Password Algorithm" RFC 6238 [Online]. Tersedia https://tools.ietf.org/html/rfc6238
[36] (2017) Website Let's Encrypt [Online]. Tersedia https://letsencrypt.org/
[37] (2017) Website Proyek Certbot [Online]. Tersedia https://github.com/certbot/certbot/
[38] R. Nikhil (2013) "Two Factor Authentication Using Mobile Phones" dalam ASM International of E-Journal of Ongoing Research in Management and IT, INCON13 [Online]. Tersedia: https://pdfs.semanticscholar.org/b607/ef444fe10250fa1a3c42f09dc02c4c4ed6b5.pdf
[39] K.P., Kaliyamurthie dan D. Parameswari (Desember 2011) "Two Factor Authentication Using Mobile Phones" International Journal of Computer Trends and Technology (IJCTT), vol 2, issue 2 number 4 [Online]. Tersedia: http://www.ijcttjournal.org/Volume2/issue-2/number-4/IJCTT-V2I2N4P3.pdf
[40] F. Aloul, et al (Mei, 2009) "Two factor authentication using mobile phones" 2009 IEEE/ACS International Conference on Computer Systems and Applications.