ISSN: 2503-054X
Vol. 5 No: 1, April 2020

Implementasi Algoritma *Username, Resolution, Color, and Hash* Dalam Otentikasi *Login* Sistem

Eza Budi Perkasa
STMIK Atma Luhur
Pangkalpinang, Indonesia
ezabudiperkasa@atmaluhur.ac.id

***Abstrak*— Algoritma *Username, Resolution, Color, and Hash* (URCH) adalah sebuah algoritma yang digunakan dalam proses otentikasi *login* sistem. Algoritma ini bekerja dengan empat tahap pemeriksaan data. Dalam algoritma ini, pemeriksaan data tidak hanya dilakukan pada *username* dan *password* saja, melainkan juga *passimage*. Teknik pemeriksaan data tersebut merupakan penggabungan dari teknik kriptografi (fungsi *hash*) dan steganografi (metode *Least Significant Bit*) pada citra digital. Penelitian ini akan memaparkan cara mengimplementasikan algoritma URCH ke dalam sebuah aplikasi sederhana. Diharapkan algoritma URCH dapat menjadi standar dalam algoritma otentikasi *login* sistem.**

Kata kunci* — *Fungsi* hash, *kriptografi*, *metode* Least Significant Bit, *otentikasi* login, *steganografi

I. PENDAHULUAN

Masalah keamanan dan kerahasiaan data merupakan salah satu aspek penting dari suatu sistem informasi. Ketika berbicara mengenai masalah keamanan yang berkaitan dengan penggunaan komputer, umumnya hal utama yang dibahas adalah mengenai kriptografi. Kriptografi merupakan suatu ilmu yang mepelajari cara menjaga data atau pesan tetap aman saat dikirimkan dari pengirim ke penerima tanpa mengalami gangguan dari pihak lain [1]. Data atau pesan yang dikirim tidak boleh jatuh ke tangan orang yang tidak berhak ataupun tidak berkepentingan [2]. Kriptografi itu sendiri bertujuan untuk memberikan layanan keamanan, termasuk keamanan untuk menjaga kerahasiaan dari *password*. *Password* yang telah dimiliki tidak boleh jatuh ke tangan orang yang tidak berhak ataupun tidak berkepentingan. *Password* biasanya digunakan untuk layanan otentikasi, baik otentikasi kebenaran pihak-pihak yang berkomunikasi maupun otentikasi kebenaran sumber pesan.

Untuk menjaga keamanan dari *password*, dapat dilakukan dengan cara mengubah *password* (*plaintext*) menjadi *ciphertext* melalui proses enkripsi. Adapun proses pengembalian *ciphertext* menjadi *plaintext* kembali disebut sebagai proses dekripsi [3]. Dikarenakan cara ini memiliki kekurangan –yaitu panjang *ciphertext* selalu sama dengan *plaintext* sehingga *plaintext* dapat ditebak dengan menghitung frekuensi kemunculan karakter, digunakan teknik lain yang disebut dengan fungsi *hash*. Fungsi ini merupakan fungsi satu arah, sehingga ketika *password* sudah di-*hash*, maka *hash password* tersebut tak dapat dikembalikan menjadi *password* semula. Adapun panjang *hash* suatu *password* adalah sama untuk *password* dengan panjang berapapun juga [4]. Akan tetapi, ternyata fungsi *hash* pun memiliki kelemahan. Jika diberikan dua buah *password* yang berbeda, maka terdapat kemungkinan bahwa kedua *password* tersebut memiliki *hash* yang sama. Ini mengakibatkan seorang *user* yang lupa dengan *password* saat masuk ke dalam sistem dapat menggunakan *password* lain yang memiliki *hash* sama tanpa mengubah *password*-nya terlebih dahulu.

Pada umumnya, otentikasi *login* sistem dilakukan dalam dua sampai tiga tahap. Otentikasi login yang dilakukan dalam dua tahap meliputi pemeriksaan *username* dan *hash password* yang di-*input*. Sementara otentikasi *login* yang dilakukan dalam tiga tahap memiliki teknik pemeriksaan lainnya, misalnya dengan menambahkan *captcha*. *Captcha* bertujuan untuk meyakinkan bahwa *user* yang memasukkan data ke dalam *form login* adalah benar-benar manusia dan bukan program yang digunakan untuk *login* otomatis (misalnya, *bot*). *Captcha* merupakan berkas multimedia yang di dalamnya terekam kode acak yang harus disalin *user* ke dalam *textbox* yang disediakan. Terkadang *captcha* justru membingungkan *user*, terutama *captcha* yang berupa gambar sederetan karakter yang diletakkan secara acak juga dan *capitalization* karakter yang hampir sama satu dengan yang lainnya.

Dalam penelitian ini, penulis bermaksud untuk membahas sebuah teknik otentikasi *login* sistem yang bernama algoritma URCH (***U**sername, **R**esolution, **C**olor, and **H**ash*). Algoritma ini merupakan penggabungan dari teknik kriptografi (fungsi *hash*) dan steganografi (metode *Least Significant Bit*) pada citra digital. Keunggulan dari algoritma ini adalah data-data *user* akan dimasukkan ke dalam sebuah gambar. Gambar tersebut nantinya akan digunakan untuk memeriksa kebenaran data dalam *form login* yang telah di-*input* oleh sistem. Gambar penampung data *user* ini tentu tidak jauh berbeda dengan gambar aslinya, sehingga kerahasiaan data dapat terjamin.

II. METODOLOGI PENELITIAN

A. Algoritma URCH

Algoritma URCH (***U**sername, **R**esolution, **C**olor, and **H**ash*) adalah salah satu algoritma dalam proses otentikasi *login* sistem. Sesuai dengan namanya, algoritma URCH memeriksa data yang dimasukkan dalam *form login* melalui empat tahap. Keempat tahap tersebut adalah sebagai berikut.

ISSN: 2503-054X
Vol. 5 No: 1, April 2020

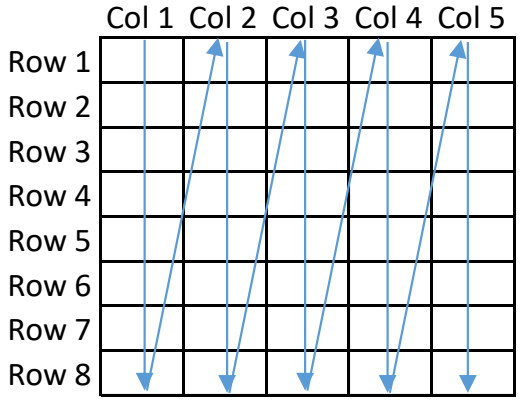
1) Pemeriksaan *username* (tahap U);
2) Pemeriksaan resolusi/ukuran *passimage* (tahap R);
3) Pemeriksaan matriks warna *passimage* (tahap C); dan
4) Pemeriksaan *hash* dari *password* (tahap H).

Algoritma ini merupakan sebuah inovasi baru dalam otentikasi *login* sistem. Selama ini, biasanya seorang *user* akan diberikan ID dan *password*. Untuk menampilkan *portrait* dirinya, *user* tersebut biasanya mengunggah foto diri ataupun gambar lainnya yang menjadi ciri khas dari sang *user* (seperti dalam situs jejaring sosial dan forum). Permasalahannya, gambar tersebut hanya menjadi hiasan semata sehingga *user* dapat menggantinya sesuka hati. Dalam algoritma URCH, gambar tersebut tidak hanya berfungsi sebagai penunjuk identitas dari seorang *user* saja: Jika *user* mengganti gambarnya, maka ia pun akan mengubah identitasnya. Hal ini berbeda dengan pemikiran sebelumnya bahwa jika *user* mengganti gambarnya, maka identitasnya belum tentu akan berubah. Adapun algoritma otentikasi *login* sistem konvensional selanjutnya disebut sebagai algoritma UH (***U****sername and* ***H****ash*). Algoritma tersebut dinamai demikian karena pemeriksaan datanya hanya melalui dua tahap saja (pemeriksaan *username* dan *hash* dari *password*) walaupun terkadang beberapa sistem menambahkan metode *captcha* untuk mempertegas bahwa *user* yang dimaksud bukanlah mesin *login* otomatis.

Dengan algoritma URCH, privasi dari *user* akan semakin terjaga. Hal ini disebabkan data *user* untuk pengujian validitas terekam dari gambar yang diunggah oleh *user*. Gambar tersebut tidak akan terlihat berbeda oleh kasat mata karena penyisipan datanya menggunakan metode steganografi LSB. Pemeriksaan gambar yang diunggah sepenuhnya dilakukan oleh komputer tanpa campur tangan manusia. Pemeriksaan gambar dilakukan dengan cara membandingkan gambar asli dan gambar yang diunggah oleh *user*. Jika gambar unggahan terlihat sama dengan gambar asli, maka pemeriksaan dilanjutkan dengan pengecekan *bit-bit* warna gambar unggahan. Jika terdapat sedikit saja perbedaan pada *bit-bit* tersebut dibandingkan dengan *bit-bit* pada gambar asli, maka dapat dikatakan bahwa data *user* yang di-*input* valid.

B. Perekaman Pesan ke *Passimage*

Secara praktis, *passimage* dapat dianalogikan dengan *password*. Perbedaan keduanya hanya terletak pada hal yang menjadi persyaratan: *Pass****image*** mensyaratkan gambar dan *pass****word*** mensyaratkan kata. Dalam penelitian ini, *passimage* yang dimaksud bukan sekedar gambar biasa, namun berupa gambar yang memiliki pesan terekam di dalamnya. Pengisian data ini dilakukan secara *column major order* (CMO) pada matriks gambar dan menerapkan metode LSB untuk tata cara pengisiannya.



Gambar 1 *Column Major Order*

Misalkan terdapat matriks *passimage* berikut.

222	43	7	131	97	159
248	251	21	233	123	201
52	84	64	253	203	59
123	0	104	208	186	172
148	191	155	235	174	235
61	234	43	239	94	175
240	7	135	148	25	217
85	221	188	34	63	185
121	238	220	160	184	21

Jika pesan yang akan dimasukkan ke dalam *passimage* tersebut adalah "*Hello*", maka *bit* pesan tersebut adalah **010010000110010101101100011011000110111 1**. Deretan *bit* tersebut merupakan penggabungan kode ASCII setiap karakter pada pesan dalam bentuk biner. Untuk memasukkan pesan tersebut ke dalam *passimage*, pertama-tama ubah setiap elemen matriks *passimage* menjadi bilangan biner 8 bit. Bilangan biner 8 bit ini dipilih karena matriks *passimage* sebenarnya adalah matriks dari warna yang digunakan pada setiap piksel yang nilainya berkisar dari 0 hingga 255 ($2^8 - 1$).

11011110	00101011	00000111	10000011	01100001	10011111
11111000	11111011	00010101	11101001	01111011	11001001
00110100	01010100	01000000	11111101	11001011	00111011
01111011	00000000	01101000	11010000	10111010	10101100
10010100	10111111	10011011	11101011	10101110	11101011
00111101	11101010	00101011	11101111	01011110	10101111
11110000	00000111	10000111	10010100	00011001	11011001
01010101	11011101	10111100	00100010	00111111	10111001
01111001	11101110	11011100	10100000	10111000	00010101

Sebelum memasukkan *bit-bit* pesan, periksalah terlebih dahulu jumlah *bit* pesan dan jumlah elemen dari matriks *passimage*. Jika jumlah *bit* pesan kurang dari atau sama dengan jumlah elemen matriks *passimage*, maka pesan tersebut dapat dimasukkan ke dalam *passimage*. Pada contoh, jumlah *bit* pesan adalah 40 dan jumlah elemen matriks *passimage* adalah 54. Dengan demikian, pesan tersebut dapat dimasukkan ke dalam *passimage*. Pesan dimasukkan dengan cara mengganti *bit* terkanan setiap elemen matriks *passimage* dengan *bit* dari pesan dari atas ke bawah, kiri ke kanan.

1101111**0**	0010101**1**	0000011**1**	1000001**0**	0110000**1**	10011111
1111100**1**	1111101**1**	0001010**0**	1110100**1**	0111101**1**	11001001
0011010**0**	0101010**0**	0100000**1**	1111110**1**	1100101**1**	00111011
0111101**0**	0000000**0**	0110100**1**	1101000**0**	1011101**1**	10101100
1001010**1**	1011111**1**	1001101**0**	1110101**0**	10101110	11101011
0011110**0**	1110101**0**	0010101**0**	1110111**0**	01011110	10101111
1111000**0**	0000011**1**	1000011**0**	1001010**1**	00011001	11011001
0101010**0**	1101110**0**	1011110**1**	0010001**1**	00111111	10111001
0111100**0**	1110111**1**	1101110**1**	1010000**0**	10111000	00010101

Agar matriks tersebut dapat diubah kembali menjadi warna, konversikan matriks tersebut ke bentuk desimal.

222	**43**	**7**	**130**	**97**	159
249	**251**	**20**	**233**	**123**	201
52	**84**	**65**	**253**	**203**	59
122	**0**	**105**	**208**	**187**	172
149	**191**	**154**	**234**	174	235
60	**234**	**42**	**238**	94	175
240	**7**	**134**	**149**	25	217
84	**220**	**189**	**35**	63	185
120	**239**	**221**	**160**	184	21

Pada keadaan sebenarnya, matriks warna pada gambar berukuran lebih besar. Selain itu, setiap gambar pun memiliki komposisi warnanya sendiri-sendiri. Sebagai contoh, gambar *grayscale* hanya memiliki satu komposisi warna, gambar RGB memilki tiga komposisi warna, dan gambar CMYK memiliki empat komposisi warna. Akibat dari komposisi warna ini, matriks yang dihasilkan tidak berupa matriks dua dimensi seperti dikenal selama ini, melainkan berupa matriks tiga dimensi. Dimensi ketiga dari matriks warna ini merupakan komposisi. Komposisi warna dapat diibaratkan tinggi atau kedalaman pada sebuah balok. Dalam penelitian ini, gambar yang digunakan dibatasi pada gambar *grayscale* dan RGB saja. Namun, tidak menutup kemungkinan gambar dengan komposisi lainnya dapat digunakan sebagai *passimage* juga.

Dalam gambar *grayscale*, pengisian data dengan CMO dilakukan seperti biasa, yaitu dari atas ke bawah, kiri ke kanan. Sedangkan pada gambar RGB, pengisian data tersebut juga mempertimbangkan komposisi warnanya. Komposisi warna pada gambar RGB terdiri dari ***R**ed* (merah), ***G**reen* (hijau), dan ***B**lue* (biru). Jika telah mencapai baris terakhir dan kolom terakhir pada komposisi R, maka data sisa diisi mulai dari baris pertama dan kolom pertama komposisi G. Jika telah mencapai batas akhir komposisi G dan masih terdapat sisa data yang belum dimasukkan, maka sisa data tersebut dimasukkan ke dalam komposisi B. Adapun untuk menyatakan resolusi dari gambar, tidak perlu menyebutkan jumlah komposisi warna, melainkan cukup dengan menyebutkan jumlah baris dan jumlah kolom saja. Komposisi warna hanya diperlukan saat tahap C dalam otentikasi *login* dengan

algoritma URCH untuk pengecekan apakah *passimage* yang dimasukkan sama dengan gambar dalam sistem karena perbedaan sedikit saja dalam warna pada piksel *passimage* akan mempengaruhi keabsahan data yang dimasukkan.

Perekaman data ke dalam *passimage* dengan metode LSB tidak akan menimbulkan perubahan secara kasat mata. Akan tetapi, komputer mampu membedakan warna piksel karena setiap warna –walaupun hanya berubah sedikit- memiliki kodenya masing-masing. Kode ini biasanya dinyatakan dalam bentuk heksadesimal dengan format #RRGGBB. Setiap digit heksadesimal mewakili empat *bit* bilangan biner sehingga setiap komposisi warna memilki dua digit heksadesimal. Sebagai contoh, warna merah murni memilki kode #ff0000. Kode tersebut dapat dinyatakan dalam biner menjadi 11111111 00000000 00000000. Untuk membuktikan bahwa metode LSB tak menimbulkan perubahan secara kasat mata, perhatikan Tabel 1 yang menampilkan perubahan *bit* warna dan warna yang dihasilkan setelah perubahan.

Tabel 1 Contoh Penerapan Metode LSB

Nilai Komposisi (Biner)			Sampel	Kode Warna
R	G	B		
11111111	00000000	00000000		#ff0000
1111111**0**	00000000	00000000		#fe0000
11111111	0000000**1**	00000000		#ff0100
11111111	00000000	0000000**1**		#ff0001
1111111**0**	0000000**1**	00000000		#fe0100
1111111**0**	00000000	0000000**1**		#fe0001
11111111	0000000**1**	0000000**1**		#ff0101
1111111**0**	0000000**1**	0000000**1**		#fe0101

Dari tabel tersebut, terlihat bahwa pengubahan sedikit pada *bit* nilai komposisi tak mempengaruhi warna secara kasat mata. Bahkan di perangkat pengolah kata penulis, sampel warna yang diambil tersebut semuanya dikenali sebagai "*Red*". Pengubahan tersebut hanya berdampak pada kode warna yang dihasilkan.

C. *Stegohash*

Pada penelitian ini, penulis menggunakan dua fungsi *hash* sekaligus: MD5 dan SHA1. Penggunaan kedua fungsi *hash* tersebut bertujuan untuk mengurangi resiko terjadinya *collision* karena sebagaimana diketahui, penggunaan satu fungsi *hash* saja dapat menghasilkan nilai *hash* yang dapat dimilki sedikitnya dua buah *password* yang berbeda. Dengan menggunakan dua fungsi *hash*, kemungkinan nilai sama tersebut dapat diperkecil. Dua fungsi *hash* yang digunakan tersebut selanjutnya disebut sebagai *stegohash.*

Stegohash adalah penggabungan nilai dari fungsi-fungsi *hash* yang berbeda dengan model steganografi. Dalam penelitian ini, *stegohash* ditentukan dengan memasukkan setiap digit heksadesimal pada *hash* SHA1 berselang-seling dengan digit heksadesimal pada *hash* MD5. Karena jumlah digit heksadesimal MD5 lebih sedikit daripada SHA1 (MD5 mempunyai 32 digit, sedangkan SHA1 mempunyai 40 digit), maka akan terdapat sisa digit. Sisa digit ini diisi sepenuhnya dengan digit-digit sisa dari SHA1. Sebagai contoh, seorang *user* memilki akun dengan *password* "user123" (tanpa tanda petik). Nilai *hash* MD5 *password* ini adalah

6ad14ba9986e3615423dfca256d04e3f

dan nilai *hash* SHA1-nya adalah

95c946bf622ef93b0a211cd0fd028dfdfcf7e39e

Jadi, nilai *stegohash* dari password sang *user* tersebut adalah

9**6**5**a**c**d**9**1**4**4**6**b**b**a**f**9**6**9**2**8**2**6**e**e**f**3**9**6**3**1**b**5**0**4**a**2**2**3**1**d**1**f**c**c**d**a**0**2**f**5**d**6**0**d**2**0**8**4**d**e**f**3**d**f**fcf7e39e.

Stegohash ini selanjutnya digabungkan dengan *username*. Jika *username* dari *user* tersebut adalah "*maniac_user*" (tanpa tanda petik), maka penggabungan *stegohash* dan *username*-nya menjadi *maniac_user*9**6**5**a**c**d**9**1**4**4**6**b**b**a**f**9**6**9**2**8**2**6**e**e**f**3**9**6**3**1** b**5**0**4**a**2**2**3**1**d**1**f**c**c**d**a**0**2**f**5**d**6**0**d**2**0**8**4**d**e**f**3**d**f**fcf7e39e.

Hasil dari penggabungan inilah yang selanjutnya dimasukkan ke dalam *passimage*. Tentu saja, *string* tersebut tidak langsung dimasukkan, melainkan harus dikonversi ke dalam deretan bit terlebih dahulu. Pengonversian ini dilakukan dengan ketentuan setiap digit *stegohash* diperlakukan sebagai karakter biasa, bukan angka. Dengan demikian, pengonversian dari penggabungan *username* dan *stegohash* pada contoh ke deretan bit menjadi

01101101011000010110111001101001011000010110001101011111011101010111001101100101011100100011100100110110
00110101011000010110001101100100001110010011000100110100001101000011011001100010011000100110000101100110
00111001001101100011100100110010001110000011001000110110011001010110010101100110001100110011100100110110
00110011001100010110001000110101001100000011010001100001001100100011001000110011001100010110010000110001
01100110011000110110001101100100011000010011000000110010011001100011010101100100001101100011000001100100
00110010001100000011100000110100011001000110010101100110001100110110010001100110011001100110001101100110
00110111011001010011001100111001011001 01

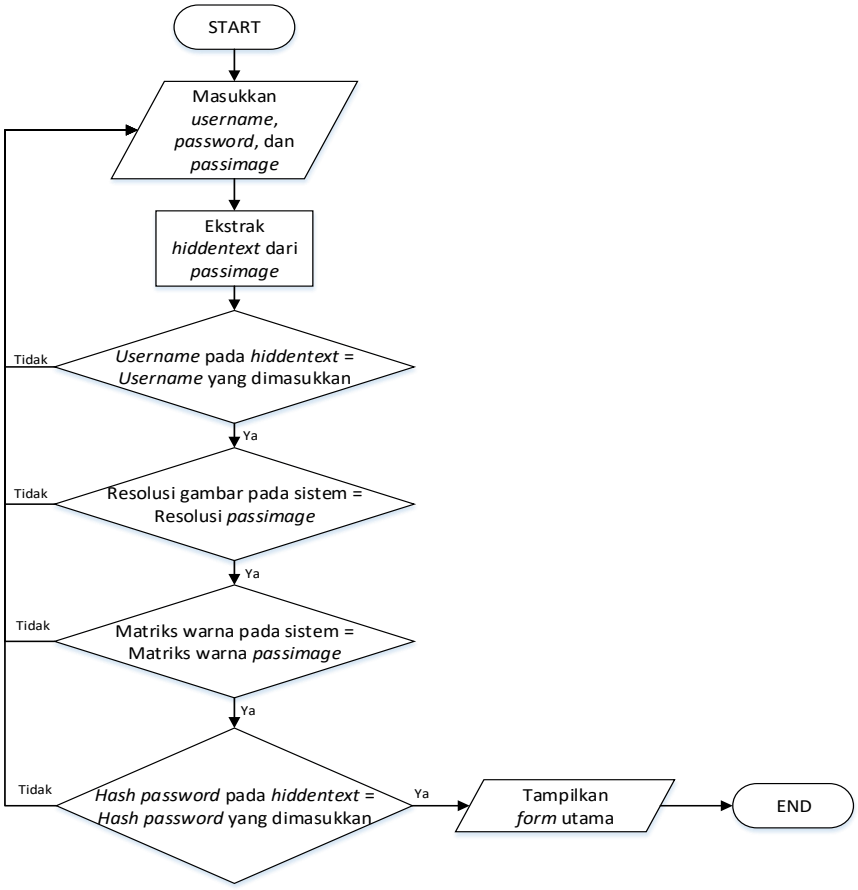
Deretan *bit* inilah yang dimasukkan ke dalam *passimage* dengan cara yang sama dengan perekaman pesan ke dalam *passimage* yang telah diuraikan sebelumnya.

Untuk memeriksa keabsahan data yang dimasukkan ke dalam *form login*, pertama-tama komputer akan mengekstrak *hiddentext* dari *passimage* (dengan asumsi *passimage* yang dimasukkan sudah benar). *Hiddentext* ini tidak lain merupakan penggabungan *username* dan *stegohash* yang terekam dan sudah dikonversi ke dalam bentuk karakter. Karena jumlah karakter dalam *stegohash* adalah tetap –yaitu 32 karaker dari MD5 + 40 karakter dari SHA1 = 72 karakter, maka dapat dipastikan bahwa 72 karakter terakhir *hiddentext* adalah *hash password* yang benar dan karakter-karakter awal dari *hiddentext* adalah *username* yang benar. Jika *username* dan *hash password* yang dimasukkan sama dengan data pada *hiddentext*, maka *user* tersebut dapat memasuki halaman/ *form* utama dari aplikasi.

D. Algoritma

Secara ringkas, algoritma URCH dalam proses otentikasi *login* sistem dapat dijabarkan sebagai berikut.

1) *User* memasukkan *username*, *password*, dan *passimage*.
2) Komputer akan melakukan pengekstrakan *hiddentext* dari *passimage*.
3) Jika *username* yang dimasukkan sama dengan *username* yang ditemukan pada *hiddentext*, maka pemeriksaan dilanjutkan. Jika tidak, maka kembali ke langkah pertama.
4) Jika resolusi gambar yang tersimpan pada sistem sama dengan resolusi *passimage* (tanpa memperhatikan komposisi warna keduanya), maka pemeriksaan dilanjutkan. Jika tidak, maka kembali ke langkah pertama.
5) Jika matriks warna pada gambar yang tersimpan pada sistem sama dengan matriks warna pada *passimage*, maka pemeriksaan dilanjutkan. Jika tidak, maka kembali ke langkah pertama.
6) Jika nilai *stegohash password* yang dimasukkan sama dengan *stegohash password* yang ditemukan pada *hiddentext*, maka *form* utama akan ditampilkan. Jika tidak, maka kembali ke langkah pertama.



Gambar 2 Algoritma URCH

III. HASIL DAN PEMBAHASAN

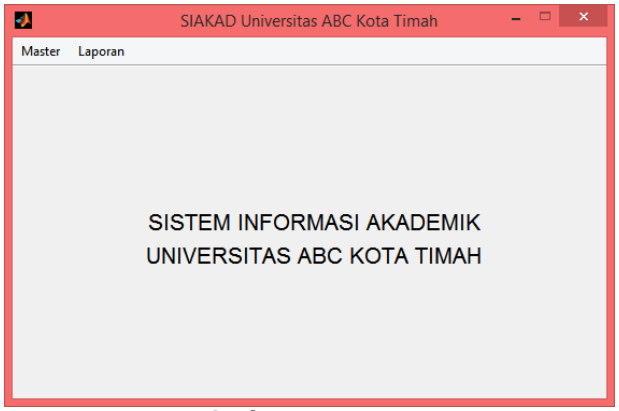
A. Implementasi

Sistem yang dibuat untuk melakukan uji coba adalah sebuah sistem informasi akademik dari sebuah universitas fiktif. *Form* yang digunakan untuk *login* ke sistem tersebut dapat dilihat pada Gambar 3.

ISSN: 2503-054X
Vol. 5 No: 1, April 2020

Gambar 3 *Form Login*

Untuk masuk ke *form* utama seperti terlihat pada Gambar 4, *user* diharuskan untuk memasukkan *username* dan *password* yang benar serta meng-*upload passimage* yang benar dengan mengklik tombol "Telusuri gambar…".



Gambar 4 *Form* Utama

B. Pengujian

Pengujian kali ini dilakukan dengan cara mencoba memasukkan data ke dalam *form login*. Pengujian akan dilakukan beberapa kali hingga terdapat data yang benar sehingga *form* utama dapat ditampilkan. Tabel 2 dan Tabel 3 menunjukkan data-data yang akan diuji.

Tabel 2 Gambar yang Digunakan Dalam Pengujian

Gambar	Nama Gambar	Resolusi	Komposisi Warna
	ubl_bw.png	156 × 160	*Grayscale*
	ubl_bw_stego.png	156 × 160	*Grayscale*

ISSN: 2503-054X
Vol. 5 No: 1, April 2020

Gambar	Nama Gambar	Resolusi	Komposisi Warna
	ubl_ori.png	156 × 160	RGB
	ubl_ori_stego.png	156 × 160	RGB

Keempat gambar tersebut sepintas terlihat sama dengan pasangannya masing-masing (gambar dengan tanpa akhiran nama dan nama dengan akhiran _stego). Akan tetapi, dari dua pasang gambar itu, hanya terdapat satu gambar yang merupakan *passimage* yang valid.

Tabel 3 *Username* dan *Password* yang Digunakan Dalam Pengujian

Username	***Password***
`user`	`12345`
`admin`	`admin`

Hasil pengujian terhadap data-data tersebut dapat dilihat pada Tabel 4.

Tabel 4 Hasil Pengujian

Percobaan	***Username***	***Password***	***Passimage***	***Form* Utama Tampil ?**
I	`user`	`12345`	ubl_bw.png	Tidak
II	`admin`	`admin`	ubl_bw.png	Tidak
III	`user`	`12345`	ubl_bw_stego.png	Tidak
IV	`admin`	`admin`	ubl_bw_stego.png	Tidak
V	`user`	`12345`	ubl_ori.png	Tidak
VI	`admin`	`admin`	ubl_ori.png	Tidak
VII	`user`	`12345`	ubl_ori_stego.png	Tidak
VIII	`admin`	`admin`	ubl_ori_stego.png	Ya

Dari tabel tersebut, dapat dilihat pada percobaan VIII, *form* utama dapat ditampilkan karena data yang dimasukkan sudah benar. Dengan demikian, data yang valid pada pengujian kali ini adalah sebagai berikut.

Username: `admin`
Password: `admin`
Passimage: ubl_ori_stego.png
Resolusi *passimage*: 156 × 160
Komposisi warna *passimage*: RGB
Hiddentext dalam *passimage*:
`admind2013233e22f22a9e73a4587aae5ba57646308f9c42a104e04aae8c0315f8c530c4da997`

IV. PENUTUP

A. Kesimpulan

Dari hasil penelitian penulis mengenai algoritma URCH dalam otentikasi *login* sistem, maka dapat diambil kesimpulan berikut.

1) Algoritma URCH dapat diimplementasikan dengan cara memasukkan *field input* tambahan berupa *passimage* pada *form* login, selain *username* dan *password* yang sudah dikenal selama ini.
2) Dalam proses otentikasi, *passimage* tersebut turut diperiksa untuk mendapatkan data masukan yang benar.
3) *Passimage* memiliki sedikit perbedaan yang tak tampak kasat mata dengan gambar aslinya.
4) *Passimage* dapat digunakan sebagai penunjuk identitas *user* ketika berinteraksi dengan *user* lain karena data yang terdapat di dalamnya tersamarkan sehingga aman digunakan tanpa khawatir mengenai kerahasiaan data tersebut.

B. Saran

Penelitian ini masih jauh dari sempurna. Oleh karena itu, penulis memberikan saran-saran berikut untuk penelitian selanjutnya.

1) Menggunakan fungsi *hash* lain dalam penyimpanan *password* dan membandingkan hasilnya.
2) Menggunakan metode steganografi lainnya untuk pembuatan *passimage*.

PENGAKUAN

Makalah ini dibuat untuk memenuhi salah satu komponen dalam Tri Dharma Perguruan Tinggi dan disponsori oleh Lembaga Penelitian dan Pengabdian Masyarakat STMIK Atma Luhur.

DAFTAR PUSTAKA

[1] E. Handayani, W. L. Pratitis, A. Nur, S. A. Mashuri, dan B. Nugroho, "Perancangan Aplikasi Kriptografi Berbasis Web Dengan Algoritma Double Caesar Cipher Menggunakan Tabel ASCII," dalam *Seminar Nasional Teknologi Informasi dan Multimedia 2017*, 2017, hal. 241-246.

[2] C. A. Sari, E. H. Rachmawanto, D. W. Utomo, dan R. R. Sani, "Penyembunyian Data Untuk Seluruh Ekstensi File Menggunakan Kriptografi *Vernam Cipher* dan *Bit Shiffting*," *Journal of Applied Intelligent System*, vol. 1, hal. 179-190, Okt. 2016.

[3] R. K. Hondro. "Aplikasi Enkripsi dan Dekripsi SMS Dengan Algoritma Zig Zag Cipher Pada Mobile Phone Berbasis Android," *Pelita Informatika Budi Darma*, vol. X, Jul. 2015.

[4] K. Aryasa dan Y. T. Paulus, "Implementasi Secure Hash Algorithm-1 Untuk Pengamanan Data Dalam Library Pada Pemrograman Java," *Citec Journal*, vol. 1, hal. 57-66, Nov. 2013.