Pembangkit Bilangan Acak Semu dengan Quasigroup pada Steganografi Citra

Ali Akbar
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Jawa Barat
ali.akbar.mawardi@gmail.com

***Abstrak*—Steganografi adalah sebuah salah satu teknik kriptografi untuk menyimpan data pada sebuah media tanpa membuat media menjadi mencurigakan bagi pihak lain atau penyerang. Pada steganografi citra, diharapkan citra yang disisipkan pesan memiliki tingkat *imperceptibility* yang tinggi sehingga tidak dicurigai memiliki pesan. Untuk itu diperlukan pembangkit bilangan acak semu untuk meningkatkan keamanan dari citra stego yang dihasilkan. Makalah ini mengusulkan pemakaian sebuah sistem pembangkit bilangan acak yang lebih baik dengan menggunakan matriks quasigrup orde N untuk membangkitkan sekuens bilangan acak. Sekuens nantinya akan digunakan pada sebuah algoritma steganografi citra, BPCS (*Bit-Plane Complexity Steganography*) untuk mengacak sekuens *bit-plane* yang disisipi pesan. Sekuens akan dikombinasikan dengan fungsi pengacak yang menggunakan algoritma Fisher-Yates pada implementasinya.**

Keywords— Pembangkit Bilangan Acak Semu, Quasigrup, Steganografi, BPCS, Fisher-Yates.

I. PENDAHULUAN

Teknologi informasi terus berkembang di era digital saat ini. Orang-orang mulai menggunakan teknologi informasi sebagai media komunikasi dan bertukar informasi utama. Hal ini dikarenakan kemudahan dan kecepatan yang diberikan oleh teknologi informasi saat ini. Untuk itu diperlukan keamanan dalam pertukaran informasi dengan teknologi saat ini. Dari sejak dahulu kala, telah diterapkan prinsip Kriptografi dalam pertukaran informasi demi menjaga keamanan data.

Kriptografi atau sandisastra adalah keahlian atau ilmu yang mempelajari hal-hal terkait komunikasi yang aman walaupun terdapat pihak ketiga (penyerang) dalam berkomunikasi. Seiring berkembangnya jaman teknik kriptografi terus berkembang. Dahulu digunakan kriptografi klasik yang memanfaatkan prinsip transposisi dan substitusi kata dalam melakukan penyandian. Setelah ditemukannya komputer, kriptografi klasik mulai dapat dengan mudah dipecahkan sehingga dikembangkan kriptografi modern yang memanfaatkan pemrosesan pada bit-bit data. Melalui kriptografi modern dikembangkan berbagai teknik kriptografi lainnya seperti steganografi citra, suara, visual kriptografi, kriptografi kunci public, kriptografi kunci simetri, dsb.

Algoritma kriptografi disebut aman apabila memenuhi 2 prinsip menurut Shannon yaitu *confusion* dan *diffusion*. Maka dari itu dalam perkembangan algoritma penyandian terdapat operasi pengacakan agar meningkatkan *confusion* bagi penyerang nantinya. Contoh penggunaan pembangkit bilangan acak terdapat pada *key generation* pada kriptografi kunci publik, pembentukan *one-time-pad key,* dsb. Tapi karena keterbatasan teknologi saat ini, untuk menghasilkan bilangan yang benar-benar acak cukup sulit sehingga digunakan pembangkit bilangan acak semu (*pseudorandom number generator*) untuk menghasilkan bilangan yang acak.

Oleh karena itu kemampuan pembangkit blangan acak akan menentukan keamanan dari algoritma kriptografi. Pada makalah ini akan diusulkan sebuah teknik pembangkit bilangan acak dengan memanfaatkan matriks Quasigroup. Quasigroup adalah struktur aljabar dalam bentuk grup yang berarti proses "pembagian" pada grup dapat selalu dilakukan. Pada makalah ini akan digunakan Quasigroup yang memanfaatkan operator "+" pada tiap elemen dan diimplementasikan pada sistem steganografi citra dengan BPCS (*Bit-Plane Complexity Segmentation Steganography*). Pembangkit bilangan acak akan digunakan untuk mengacak urutan *bitplane-bitplane* terenkripsi menggunakan algoritma Fisher-Yates.

II. DASAR TEORI

A. Quasigroup

Quasigrup adalah matriks n x n yang berisi nilai permutasi dari sebuah *field* Z yang mana tidak ada elemen yang sama sebaris maupun sekolom. Quasigrup memenuhi hukum berikut.

$$\forall u, v \in Q, \exists x, y \in Q \text{ memenuhi } u.x = v \text{ dan } y.u = v$$

"." adalah operasi *lookup* yang mana berarti operan pertama adalah indeks baris dan operan kedua adalah indeks kolom. Hasil dari fungsi ini adalah elemen matriks quasigrup (dinotasikan dengan Q) pada indeks baris dan kolom tersebut. Ada beberapa properti yang dimiliki oleh quasigrup.

a. Jika $x.y = x.z$ maka $y = z$

b. Berlaku operasi adjoin "/" dan "\" yang didefinisikan sebagai berikut.

$$x.y = z \leftrightarrow y = x\backslash z \leftrightarrow y = x/z$$

B. Pembangkit Bilangan Acak Semu

Pembangkit bilangan acak semu (disingkat PRNG dari bahasa inggirs *pseudorandom number generator*) atau sering disebut juga dengan pembangkit bit deterministik adalah algoritma untuk menghasilkan sekuens bilangan acak. Seperti namanya, pembangkit bilangan acak semu tidak menghasilkan sekuens bilangan yang benar-benar acak (*truly random*). Namun sekuens bilangan acak yang dihasilkan ditentukan oleh *initial value* yang disebut dengan *seed*. Walaupun sekuens yang *truly random* dapat dibentuk oleh *hardware number generator*, kecepatan komputasi dan kemampuan untuk diproduksi ulang dari PRNG menjadi keunggulan yang dapat digunakan di berbagai aspek pada kriptografi.

Secara matematis pembangkit bilangan acak semu adalah fungsi yang mana hasil dari fungsi sebanyak n adalah sekuens bilangan acak (sesuai kriteria *randomness*). Adapun kriteria pembangkit bilangan acak semu menurut BSI, sebuah badan keamanan informasi federal milik jerman adalah sebagai berikut.

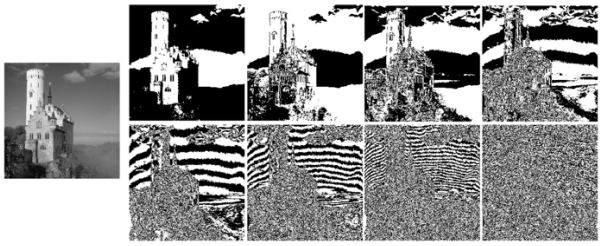
a. K-1 - Harus ada kemungkinan yang tinggi bahwa sekuens bilangan acak berbeda dengan sekuens lainnya.
b. K-2 – Sekuens tidak bisa dibedakan dengan sekuens *truly random* berdasarkan uji statistik yang digunakan.
c. K-3 – Penyerang harus tidak mungkin untuk mengkalkulasi ataupun menebak nilai fungsi selanjutnya dan *inner state* fungsi saat ini dari data yang ada
d. K-4 – Penyerang harus tidak mungkin untuk mengkalkulasi ataupun menebak nilai-nilai sekuens sebelumnya dari sebuah *inner state* fungsi.

Untuk aplikasi kriptografi, hanya pembangkit yang memenuhi kriteria 3 dan 4 saja yang menjadi standar untuk diterima sebagai pembangkit bilangan acak semu.

C. Bit-Plane Complexity Segmentation Steganography (BPCS)

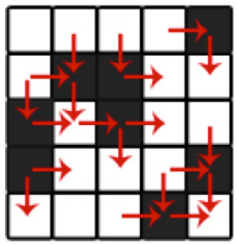
BPCS adalah algoritma steganografi yang dikembangkan oleh Eij Kawaguchi dan R.O Eason pada 1997. Metode BPCS ini idenya sama dengan penyisipan pesan menggunakan LSB yaitu memanfaatkan kekurangan sensivitas mata manusia terhadap perubahan warna yang sedikit. Namun pada BPCS digunakan *bitplane* daripada konfigurasi bit tiap pixelnya sehingga data yang disisipkan jumlahnya dapat jauh lebih banyak.

Bitplane adalah citra biner yang berisi bit ke-*i* dari pixel-pixel dalam citra. Sebagai contoh pada sebuah citra biner dengan 1 pixel sama dengan 1 byte dapat dihasilkan 8 bitplane berbeda seperti yang digambarkan pada gambar 1.



Gambar 1 Contoh Bitplane dari Sebuah Citra

Dengan menghasilkan setiap bitplane dari sebuah citra nantinya akan dicari bitplane-bitplane kompleks untuk disisipi dengan pesan. Bitplane kompleks ini disebut dengan *noise-like regions*. Kompleksitas (dinotasikan dengan α) dari bitplane dihitung dengan perubahan warna hitam dan putih pada tiap sel dalam bitplane sebuah citra. Semisal kita memiliki bitplane dari citra berukuran 5x5 seperti pada Gambar 2. Maka besar perubahan warna hitam dan putih pada bitplane tersebut adalah 20. Nilai kompleksitas mengikuti persamaan 1 dimana k adalah banyak perubahan hitam dan putih pada citra sementara n adalah banyak kemungkinan perubahan hitam dan putih pada citra berukuran N x N. Karena BPCS bekerja pada blok citra 8x8 maka besar n pada perhitungan kompleksitas adalah 112 karena maksimum perubahan warna hitam putih yang dapat terjadi adalah 112 pada citra 8x8. Citra disebut disebut kompleks apabila nilai kompleksitasnya berada diatas *threshold* yang telah ditentukan.



Gambar 2 Contoh Perhitungan Kompleksitas Bitplane.

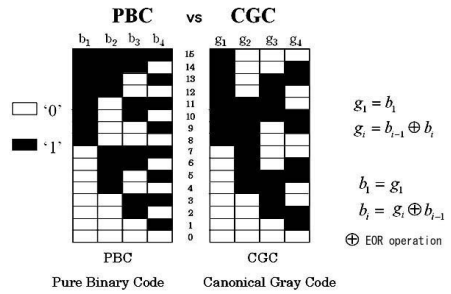
$$\alpha = \frac{k}{n}$$

Persamaan 1 Rumus Perhitungan Kompleksitas Bitplane

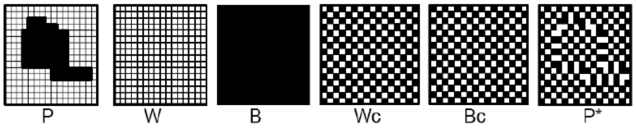
Pada proses penyisipan pesan menggunakan BPCS sistem bitplane dapat diubah menjadi sebuah sistem CGC (*Canonical Gray Coding*) sehingga penyisipan pesan menjadi lebih mudah. Gambar 3 menunjukkan skema perubahan citra dari PBC (*Pure Binary Coding* atau bentuk image citra asli) ke sistem CGC.

Bitplane hasil penyisipan pesan belum tentu kompleks sehingga diperlukan operasi transformasi agar bitplane kembali menjadi bitplane yang kompleks (apabila tidak kompleks) menggunakan operasi konjugasi. Operasi konjugasi menghasilkan sebuah citra biner dengan kompleksitas 1-α. Persamaan 2 memperlihatkan proses konjugasi dengan P^* adalah citra biner hasil konjugasi, W adalah citra biner pixel putih, B adalah negasi dari W (citra biner pixel hitam), Wc adalah citra biner papan catur dan Bc adalah negasi dari Wc. P

adalah citra yang akan dikonjugasi. Penjelasan mengenai variable ini digambarkan pada gambar 4.



Gambar 3 Perubahan Citra dari Sistem PBC ke CGC



Gambar 4 Variabel Persamaan Konjugasi

$$P^* = P \oplus Wc$$

Persamaan 2 Fungsi Konjugasi Citra Biner

Adapun algoritma penyisipan pesan (*message embedding*) dan ekstraksi pesan pada BPCS adalah sebagai berikut.

a. Penyisipan Pesan
   1. Bagi citra sampul menjadi blok 8x8 pixel
   2. Bentuk bitplane-bitplane dari tiap blok yang dihasilkan
   3. Bitplane-bitplane tersebut dapat diubah menjadi sistem CGC jika diinginkan
   4. Tentukan bitplane-bitplane kompleks (*noise-like regions*) menggunakan *threshold* yang ditentukan. (*default*-nya adalah 0.3)
   5. Bagi pesan menjadi segmen 64 bit lalu ubah menjadi blok citra berukuran 8x8
   6. Jika blok pesan S belum kompleks (sesuai threshold yang ditentukan) maka lakukan konjugasi sehingga dihasilkan blok pesan S* yang lebih kompleks.
   7. Sisispkan blok ke *noise-like regions* dengan mengganti seluruh bitplane
   8. Jika blok S dikonjugasi maka simpan pesan dalam *conjugation map*
   9. Sisipkan juga *conjugation map* dalam citra
   10. Ubah citra hasil dari sistem CGC ke PBC jika sebelumnya diubah ke CGC.

b. Ekstraksi Pesan
   1. Bagi citra stego menjadi blok 8x8 pixel
   2. Bentuk bitplane-bitplane dari tiap blok
   3. Ubah bitplane menjadi CGC (jika sebelumnya dilakukan)
   4. Hitung kompleksitas dari bitplane, apabila nilai kompleksitas lebih dari *threshold* maka bitplane tersebut adalah pesan. Perhatikan pula apakah bitplane perlu dikonjugasi (menggunakan *conjugation map*) sebelum pesan diekstraksi.
   5. Pesan berhasil diekstraksi

Untuk meningkatkan keamanan dan mengurangi *detectability* dari sebuah citra stego maka kerap kali digunakan algoritma kriptografi kunci simetri dan fungsi hash untuk mengenkripsi pesan. Lalu bitplane-bitplane kompleks juga diajak lokasi penyisipannya sehingga citra stego lebih sulit untuk dicurigai.

D. Algoritma *shuffling* Fisher-Yates

```
=== Untuk array a dengan n elemen (index
dari 0 ke n-1) ====
for i from n-1 downto 1 do
    j ← random number such that i ≤ j ≤ n
    swap a[i] with a[j]
```

Algoritma 1 Algoritma Pengacakan Fisher-Yates

Untuk meningkatkan keamanan dan kebaikan hasil citra pada BPCS diperlukan pengacakan pada *bitplane-bitplane* yang ter-*embed* pesan. Untuk itu diperlukan proses pengacakan (*shuffling*) pada urutan *bitplane-bitplane* citra. Salah satu algoritma *shuffling* yang cepat dan mudah untuk diimplementasikan adalah algoritma Fisher-Yates. Algoritma ini bekerja untuk menghasilkan permutasi acak dari urutan yang terbatas. Pengacakan dalam implementasiknya memanfaatkan fungsi pembangkit bilangan acak. *Pseudocode* dari algoritma Fisher-Yates adalah tertera pada Algoritma 1.

Namun algoritma Fisher-Yates ini pada implementasinya tidak dapat diimplementasikan pada quasigrup karena tidak mungkin untuk membentuk quasigrup orde baru (sebesar i) pada tiap pembangkitan bilangan acak. Maka dari itu diusulkan pembentukan sekuens acak dengan membentuk sekuens bilangan acak dari orde N dan menggunakan elemen pada tiap sekuens sebagai bilangan acak.

III. IMPLEMENTASI PEMBANGKIT BILANGAN ACAK SEMU DENGAN QUASIGROUP PADA BPCS

Terdapat publikasi-publikasi sebelumnya mengenai pemanfaatan Quasigroup dalam kriptografi dan pembangkitan bilangan acak, contohnya adalah implementasi pembangkit bilangan acak menggunakan quasigroup yang dibuat oleh Godavarty [3]. Namun pada implementasi tersebut tidak dijelaskan mengenai pembangkitan matriks quasigroup dengan orde *N*. Pada implementasi penelitian ini, akan dikembangkan metode pembangkitkan matriks orde N mengadaptasi dari teknik yang dikembangkan oleh Battey dan Parakh [4]. Nantinya pembangkit bilangan acak ini akan digunakan untuk mengacak sekuens *bitplane* pada arsitektur BPCS menggunakan algoritma Fisher-Yates.

A. Pembangkit Matriks Quasigroup Acak Orde N

Pada implementasi pembangkit matriks quasigroup acak orde N diadaptasi algoritma milik Battey dan Parakh dengan sedikit modifikasi. Pada algoritma sebelumnya, digunakan 2 buah seed untuk menghasilkan nilai *random* pada matriks. Pada implementasi yang dibuat, digunakan sebuah seed berbentuk string yang nantinya akan dibagi menjadi 2 bagian sehingga menghasilkan 2 buah nilai seed. Hal ini dilakukan

agar selalu dapat dihasilkan matriks yang sama menggunakan kunci yang dimasukkan pada BPCS. Pembangkitan seed dilakukan dengan fungsi pada algoritma 2.

```
function GenerateSeed(key_seed : string){
        s1← first half of key_seed
        s2 ← remaining string of key_seed
        seed1 ← sum of all character asci
values in s1 mod N
        seed2 ← sum of all character asci
values in s2 mod N
        return seed2
}
```

Algoritma 2 Fungsi GenerateSeed menggunakan String Key

Algoritma pembangkit matriks quasigroup acak adalah sebagai berikut.

1. Pilih sebuah kunci dan bangkitkan 2 seed menggunakan fungsi pembangkit seed pada algoritma 2. Seed yang dihasilkan adalah *s1* dan *s2*.
2. Bentuk matriks quasigroup awal dengan nilai elemen setiap sel adalah (*i*+*j*) mod *N* dengan *i* adalah indeks baris, *j* adalah indeks kolom pada matriks, dan *N* adalah orde matriks. Matriks ini disimbolkan dengan QG.
3. Inisiasi sebuah variable bernama *idx* dengan nilai 0
4. Ambil nilai elemen dari matriks QG pada indeks *s1* dan *s2* (QG[*s1*][*s2*]) dan simpan pada variable *r*.
5. Lakukan penukaran nilai, *s1* = *s2* dan *s2* = *r*.
6. Lakukan penukaran elemen-elemen pada kolom dengan indeks *idx* dan *s1*
7. Inkremenkan nilai *idx* dengan operasi *idx*+*1* mod *N*
8. Ulangi langkah 4 sampai 7 sebanyak apapun hingga matriks dirasa sudah cukup teracak.

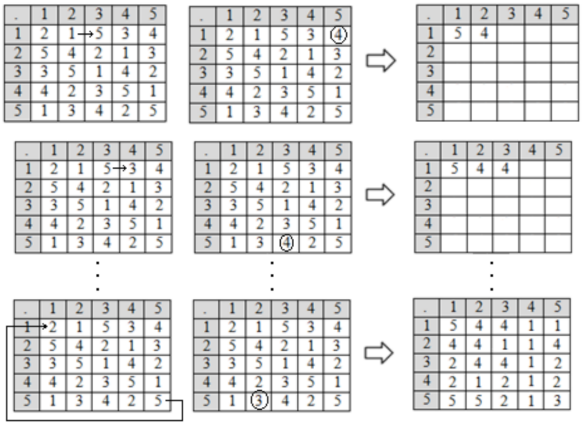
B. Pembangkit Bilangan Acak Semu dengan Quasigroup

Pada implementasi pembangkit bilangan acak semu digunakan teknik yang dikembangkan Godavarty dalam pembentukan sekuens bilangan acak. Algoritma Godavarty terbagi menjadi 3 bagian.

1. Bagian 1 berisi fungsi untuk membentuk matriks yang teracak menggunakan fungsi *lookup*.
2. Bagian 2 berisi fungsi untuk membangkitkan sekuens bilangan acak orde N dari matriks quasigroup
3. Bagian 3 berisi fungsi untuk menggeser nilai-nilai matriks secara *row-wise* sebanyak konstanta yang ditentukan.

Adapun penjelasan untuk tiap bagian adalah sebagai berikut.

a. Bagian 1
Masukan dari fungsi bagian 1 adalah sebuah matriks quasigroup orde N acak sebagai inisiasi awal algoritma. Matriks ini dibentuk menggunakan teknik pembangkitan yang telah dibuat pada bagian sebelumnya (III.A). Hasil dari fungsi ini adalah sebuah matriks quasigroup baru yang disebut *g*. Elemen-elemen pada *g* diisi dengan fungsi *lookup* pada quasigroup awal dengan indeks i dan elemen setelahnya. Untuk menentukan elemen setelahnya digunakan prinsip list sirkuler secara *row-wise* dari matriks quasigroup. Pemrosesan secara list sirkuler yang dimaksud dijelaskan pada gambar. *Pseudocode* dari fungsi *lookup* dan bagian 1 dijelaskan pada algoritma 3 & 4.



Gambar 5. Proses Lookup pada Bagian 1

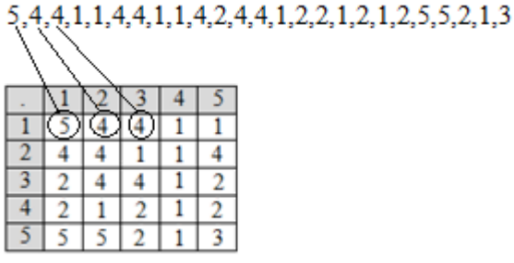
```
function Phase_1(M : Quasigroup)
{
   g = Empty matrix N x N
   for i from 0 to N-1 do
       for j from 0 to N-1 do
          g[i][j] = lookup(M[i][j], next
   of M[i][j])
   return g
}
```

Algoritma 3 Pseudocode Bagian 1 (Phase 1)

b. Bagian 2
Pada bagian ini dibentuk sekuens bilangan acak dari hasil pengacakan yang dilakukan pada bagian 1 dengan membentuk list dari baris per baris pada matriks quasigroup. Pseudocode dari bagian 2 ditunjukkan pada algoritma 4 dan gambaran proses yang terjadi dapat dilihat pada gambar 6.

```
function Phase_2(g : Quasigroup){
   seq = list of number
   for i from 0 to N-1 do
        seq += row[i] of g
   return seq
}
```

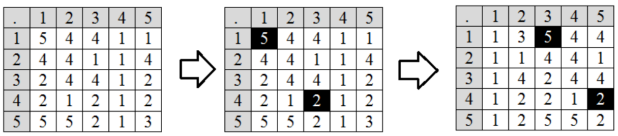
Algoritma 4 Pseduocode bagian 2 (Phase 2)



Gambar 6 Proses Ekstraksi Sekuens

c. Bagian 3

Masukan dari fungsi pada bagian 3 hanyalah sebuah konstanta yang lebih kecil dari N untuk melakukan operasi pergeseran matriks. Penggambaran dari proses yang dilakukan pada bagian 3 digambarkan pada gambar . Konstanta geser yang digunakan pada pengammbaran adalah 2 dan matriks berukuran 5x5.



Gambar 7 Proses Penggeseran dengan Kontanta 2

Dengan menggunakan 3 algoritma di atas algoritma pembangkitan bilangan acak menggunakan quasigroup adalah sebagai berikut.

1. Bangkitkan sebuah matriks quasigroup acak
2. Lakukan algoritma bagian 1
3. Lakukan algoritma bagian 2 dan simpan hasil sekuens sementara
4. Lakukan algoritma bagian 3
5. Ulangi langkah 2 sampai 4 hingga terbentuk sekuens sepanjang yang diinginkan
6. Apabila sekuens yang dihasilkan jauh lebih panjang, potong hingga sesuai

Pseudocode dari langkah-langkah algoritma pembangkitan bilangan acak dengna quasigrup dituliskan pada Algoritma 5.

```
Function GenerateSequence(key: string, c : int){
   g = GenerateRandomMatrix(key)
   seq = empty list of number
   while (seq.length < c) do
     g = Phase_1(g)
     seq += Phase_2(g)
     g = Phase_3(g)
   if (seq.length > c) do
     seq = seq[:c]
   return seq
}
```

Algoritma 5 Algoritma Pembangkit Bilangan Acak dengan Quasigrup

C. *Fungsi Pengacak Sekuens dengan Quasigroup Pseudorandom Number Generator*

Dari penjelasan mengenai pembangkitan bilangan acak sebelumnya kita tahu bahwa untuk membangkitkan bilangan acak dari 0 ke N diperlukan sebuah matriks quasigrup berorde N. Hal ini membuat adanya kesulitan dalam pengacakan sekuens dengan panjang sekuens sebesar *l* karena harus dibentuk matriks quasigrup berukuran *l* pula. Untuk itu dilakukan sedikit modifikasi dalam algoritma pengacakan sekuens menggunakan algoritma Fisher-Yates pada quasigrup. Pertama kita bentuk dahulu sebuah sekuens bilangan acak sepanjang *l* menggunakan sebuah orde kecil yang sudah diuji tingkat randomnessnya. Lalu tiap elemen digunakan sebagai bilangan acak yang akan digunakan untuk proses pertukaran elemen di dalam sekuens yang diproses. Pseudocode dari proses ini dituliskan pada algoritma 6.

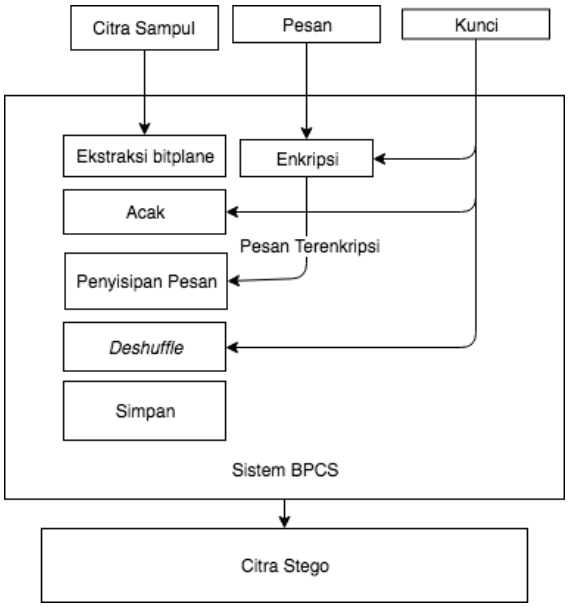
```
function shuffle(seq: list of object, seed: string){
    n = seq.length
    rnd_seq = GenerateSequence(seed, n)
    for i from n-1 downto 1 do
      j ← rnd_seq[i] mod n
      swap a[i] with a[j]
}
```

Algoritma 6 Algoritma Pengacakan dengan Quasigrup PRNG

Operasi modulo dilakukan pada penentuan nilai j agar nilai j masih memenuhi aturan nilai j yang diterapkan pada algoritma Fisher-Yates yang asli. Proses *deshuffle* dapat dilakukan dengan mengubah urutan iterasi (*reverse*).

D. *Implementasi Sistem BPCS*

Sistem BPCS yang diimplementasikan sama dengan algoritma BPCS yang telah dijelaskan pada bagian kedua. Namun untuk meningkatkan *imperceptibility* dari citra stego diimplementasikan sebuah pengacak sekuens pada saat proses *message embedding* dilakukan. BPCS menerima kunci dan sebuah citra sampul untuk melakukan proses penyisipan pesan. Kunci digunakan untuk mengenkripsi pesan menggunakan algoritma enkripsi kunci simetri. Kunci juga digunakan untuk melakukan pengacakan sekuens menggunakan fungsi *shuffling* yang telah didefinisikan pada bagian III.C. Pada saat penyimpanan ke dalam citra stego urutan bitplane dikembalikan (*deshuffle*) agar menjadi citra yang utuh. Gambaran dari proses penyisipan pesan pada gambar menggunakan BPCS dapat dilihat pada gambar.



Gambar 8 Sistem BPCS Terimplementasi

Proses pengacakan menggunakan matriks quasigrup cukup memakan waktu terutama pada proses pembentukan matriks quasigrup acak. Untuk meningkatkan kecepatan maka

matriks acak orde N disimpan ke file eksternal dan dibuka pada saat fungsi pengacak dipanggil.

IV. Desain Eksperimen dan Skema Pengujian

Pada eksperimen penelitian ini, digunakan bahasa python sebagai bahasa pemograman lingkungan pengujian. Dilakukan 3 pengujian standar terhadap fungsi pembangkit bilangan acak semu dengan quasigrup berdasarkan standar NIST. Pengujian tersebut antara lain uji monobit dan *block frequency*. Uji *randomness* dilakukan terhadap orde matriks quasigrup 3-100 dan akan dicari orde terbaik untuk menghasilkan sekuens yang cukup acak. Untuk pengujian implementasi pada sistem BPCS digunakan nilai PSNR antara citra sampul dan citra stego hasil penyisipan dan pengacakan menggunakan matriks quaasigrup dengan orde terbaik.

A. Desain Parameter Pembangkit Bilangan Acak Semu dengan Quasigrup

Karena dibutuhkan minimal sebuah string untuk dijadikan seed awal dalam melakukan pengacakan, maka harus dipilih sebuah string default untuk menghasilkan sekuens. Diperlukan pula limitasi iterasi untuk pembetukan matriks acak awal. Dipilih string "default" sebagai *default* string untuk *seed generator* dan 5 iterasi berdasarkan hasil uji coba.

B. Skema Uji *Randomness*

Skema pengujian *randomness* mengikuti standar uji frekuensi yang diajukan oleh Charmaine pada tahun 2005. Pada eksperimen ini pengujian frekuensi lebih dipilih agar hasil dari sekuens acak pada citra stego aman dari serangan secara statistik. Nantinya hasil sekuens bilangan acak akan dibandingkan dengan *default* fungsi random yang telah diimplementasi oleh bahasa pemograman. Bahasa pemograman yang dipilih pada implementasi Python sehingga fungsi *random* yang akan dijadikan sebagai pembanding adalah fungsi *random* milik Python. Pengujian dilakukan dengan membentuk sekuens bit (string berisi 0 1) secara acak dengan panjang 128 bit. Pada referensi yang diberikan oleh NIST, standar panjang sekuens untuk diuji minimal 100 sehingga 128 bit adalah panjang yang cukup valid untuk meguji tingkat keacakan sekuens. Untuk setiap pengujian berlaku hipotesa sebagai berikut.

$$H_0 : Sekuens\ bit\ tidak\ random$$
$$H_1 : Sekuens\ bit\ random$$

Digunakan tingkat kepercayaan (α) sebesar 1% (0.01) sehingga H_0 ditolak apabila *P-Values* dari tiap pengujian lebih besar sama dengan 1%.

Adapun penjelasan mengenai 2 jenis pengujian tersebut adalah sebagai berikut.

1. *Monobit Test*
   Uji monobit adalah pengujian sekuens bit (0,1) untuk menentukan apakah sekuens dapat mendekati sekuens *truly random* yang diharapkan. Pengujian ini mengharapkan kedekatan antara fraksi 1 ke 0.5 dimana banyak elemen 0 dan 1 pada sekuens seimbang. Adapun perhitungan *P-Values* pada tes ini diperlihatkan pada persamaan berikut.

$$s_{obs} = \frac{|S|}{\sqrt{n}}$$
$$P = erfc\left(\frac{s_{obs}}{\sqrt{2}}\right)$$

   Dengan S adalah sekuens bit dan n adalah panjang sekuens. H_0 ditolak apabila $P \geq 0.01$.

2. *Block Frequency Test*
   Fokus dari tes ini adalah menguji proporsi dari angka 1 dari blok bit berukuran M. Tujuan dari tes ini adalah menguji apakah frekuensi bit 1 pada blok M-bit mendekati M/2 sesuai dengan harapan dari asumsi keacakan. Adapun langkah-langkah untuk membentuk P-value adalah sebagai berikut.
   a. Partisi sekuens masukan ke N buah blok yang tidak tumpeng tindih dengan nilai N adalah sebagai berikut.

$$N = \left\lfloor \frac{n}{M} \right\rfloor$$

   b. Semisal ε adalah sebuah sekuens bit. Hitung proporsi dari 1, π_i, untuk tiap M-bit blok dengan persamaan.

$$\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M}$$

   c. Hasilkan statistik chi-square dari observasi.

$$\chi^2(obs) = 4M \sum_{j=1}^{M} (\pi_i - \frac{1}{2})^2$$

   d. Nilai *P-Values* dari tes adalah sebagai berikut.

$$P = igamc(\frac{N}{2}, \frac{\chi^2(obs)}{2})$$

   H_0 ditolak apabila $P \geq 0.01$ dan M yang dipakai pada pengujian ini adalah 10 sesuai standar yang tertulis pada referensi.

C. Skema Uji PSNR pada Citra Stego

PSNR (*Peak Signal-to-Noise Ratio*) adalah rasio antara daya sinyal dan daya *noise* terhadap *fidelity* dari representasi sinyal. Pada pengolaahn citra, PSNR adalah rasio citra asli terhadap citra yang memiliki *noise* dengan efek kepada *fidelity* citra. PSNR sering digunakan untuk mengukur kualitas citra setelah *lossy compression* atau pemrosesan seperti penyisipan pesan. Adapun nilai PSNR adalah sebagai berikut.

$$PSNR = 20\ x \log(\frac{256}{RMSE})$$

Dengan RMSE (*root mean square error*) adalah akar pangkat 2 dari rata-rata error (perbedaan pixel pada citra asli dan citra stego) dengan rumus.

$$RMSE = \sqrt{\frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} (A_{ij} - B_{ij})^2}$$

A adalah pixel citra asli dan B adalah pixel citra hasil pemrosesan dengan masing-masing berukuran M x N. Semakin tinggi nilai PSNR maka kualitas citra dianggap

semakin baik. Nilai mimal yang biasa digunakan untuk menentukan PSNR yang baik adalah 60.

V. HASIL EKSPERIMEN DAN ANALISIS

Dengan desain dan skema pengujian yang telah dijelaskan sebelumnya. Berikut adalah hasil eksperimen yang dihasilkan.

A. Hasil Uji *Randomness*

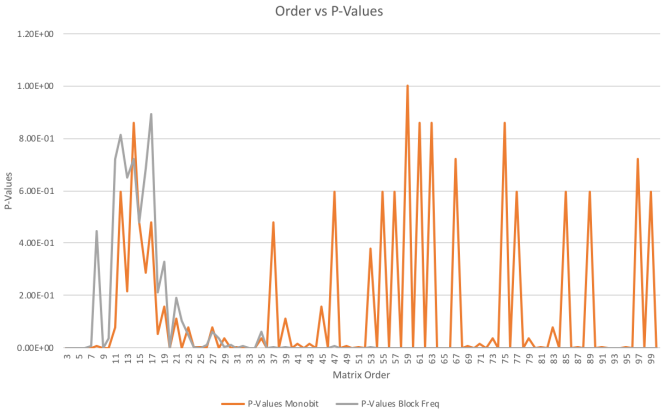
Sesuai dengan desain eksperimen dan pengujian yang telah dijelaskan sebelumnya. Dilakukan pengujian nilai *P-Values* dari uji monobit dan *block frequency* antara algoritma yang diimplementasikan dengan fungsi *random default* dari bahasa pemograman python. Dibuat sekuens bit (0,1) sepanjang 128 bit dalam pengujian. Pengujian pertama dilakukan antara matriks orde 5, 8, 11 dan 14 dengan fungsi *random default* Python.

Pada tabel 1 terlihat bahwa fungsi pengacak *default* dari python jauh lebih unggul dibandingkan dengan fungsi pengacak dengan quasigrup pada orde 5. Untuk orde yang lebih tinggi, hanya pada tes monobi untuk orde 8 yang nilai *P-Values* yang lebih kecil dibandingkan dengan *default* Python sementara yang lainnya memiliki nilai yang lebih tinggi. Nilai *P-Values* tertinggi diraih pada pembangkit bilangan acak dengan quasigrup orde 14. Dari tabel juga terlihat bahwa orde yang berbeda menghasilkan nilai *P-Values* yang berbeda pula. Terdapat dugaan bahwa semakin tinggi orde maka semakin tinggi pula nilai *P-Values*. Dilakukan pengujian nilai *P-Values* terhadap orde 3 hingga 100 pada fungsi pengacak dengan quasigrup.

Grafik 1 memperlihatkan nilai *P-Values* monobit (garis berwarna oranye) tersebar secara acak dan memiliki *region-region* khusus dimana nilai akan memiliki angka yang tinggi. Namun pada *P-Values* dari *block frequency* (garis berwarna abu-abu) nilai *P-Values* tinggi untuk 1 daerah saja, yaitu di sekitar orde 9 sampai 20. Untuk orde selanjutnya terjadi penurunan terus menerus dan tidak ditemukan kenaikan nilai pada daerah lainnya. Dari hasil ini terbukti bahwa dugaan awal mengenai kenaikan nilai *P-Values* terhadap orde secara linier itu salah. Dari hasil juga dapat disimpulkan bahwa nilai orde yang baik untuk digunakan untuk mengacak sekuens adalah nilai orde 9-20 dengan nilai orde 14 adalah nilai terbaik karena memiliki nilai *P-Values* yang tinggi untuk kedua tes.

Pengacak	P-Values Monobit	P-Values Block
Python	0.288	0.288
QG orde 5	1.54e-12	3.37e-11
QG orde 8	0.008	0.44
QG orde 11	0.077	0.72
QG orde 14	**0.86**	**0.72**

Tabel 1 Pengujian Pertama Antara Default Python dan Quasigrup



Grafik 1 Grafik Orde Matriks vs P-Values

B. Hasil Uji Kecepatan Komputasi

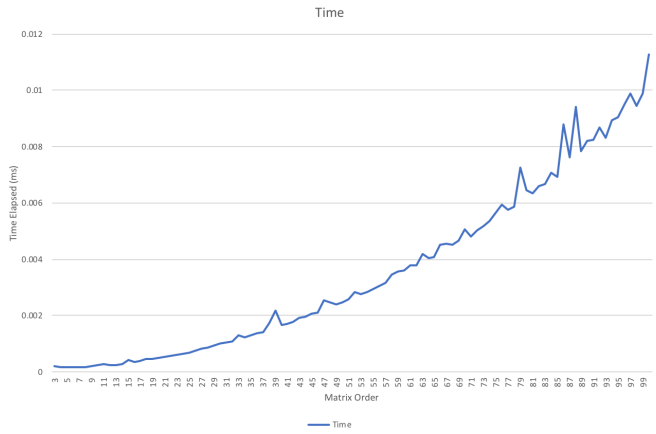
Dari kompleksitas algoritma, sudah jelas bahwa waktu komputasi dari fungsi pembangkit bilangan acak semu dengan quasigrup pastinya akan naik secara kuadratik akibat operasi matris N x N. Grafik 2 menunjukkan plot grafik antara waktu dan orde matriks untuk membuktikan bahwa waktu komputasi akan naik secara kuadratik seiring bertambahnya orde. Sehingga untuk diimplementasikan ke dalam sistem steganografi sebaiknya digunakan orde yang kecil agar komputasi menjadi lebih cepat. Rata-rata waktu komputasi dari orde 3 hingga 100 sekitar 0.003 ms dengan nilai minimal (orde 3) 0.00019 ms. Nilai ini lebih lambat dibandingkan dengan kecepatan fungsi *random default* dari Python yang memakan waktu sekitar 0.00015 ms. Namun hal ini tidak terlalu menjadi masalah karena perbedaan tidak terlalu signifikan (hanya sekitar 0.00004 ms). Waktu komputasi untuk orde 14 adalah sekitar 0.00027 ms yang mana pada orde 14 nilai *P-Values* jauh lebih tinggi dibandingkan fungsi pembanding. Waktu tersebut juga tidak signifikan perbedaannya dengan fungsi pembanding.

C. Hasil Uji PSNR

Dilakukan eksperimen untuk memasukkan *file* teks sebesar 2KB ke dalam sebuah citra *greyscale* lena dengan format bmp berukuran 263KB. Citra lena yang dipakai dapat dilihat pada gambar 9. Dilakukan pengujian antara pengacak dengan quasigrup orde 14, tanpa pengacakan, dan pengacakan dengan fungsi pengacak *default* milik Python. Dari hasil yang diperlihatkan pada tabel 2 nilai PSNR quasigrup orde 14 jauh lebih besar dibandingkan nilai PSNR milik pengacak *default* Python dan nilai PSNR terbesar tetap dimiliki oleh fungsi BPCS tanpa pengacakan. Namun seperti yang dapat dilihat pada gambar 10 bahwa gambar tanpa pengacakan akan membuat citra kelihatan rusak (lihat pada bagian elips merah) di mata manusia biasa dibandingkan dengan hasil citra yang sekuensnya sudah diacak dengan pengacak quasigrup. Perbedaan nilai PSNR juga tidak terlalu signifikan sehingga dapat disimpulkan bahwa nilai PSNR yang lebih kecil tidak terlalu berarti dibandingkan dengan kualitas *imperceptibility* citra yang dihasilkan.

Kategori	Nilai PSNR
Tanpa Pengacakan	**70.73**
Pengacakan Fungsi Python	67.1
Pengacakan QG 14	70.56

Tabel 2 Tabel Hasil Pengujian Nilai PSNR



Grafik 2 Grafik Waktu Komputasi vs Orde Pada Fungsi Pengacak Quasigrup

Gambar 9 Citra Lena yang digunakan

Gambar 10 Perbedaan Citra Stego antara Tanpa Pengacakan (kiri) dan dengan Pengacakan (kanan)

VI. KESIMPULAN DAN SARAN

Dari hasil percobaan yang telah dilakukan dapat disimpulkan bahwa dapat dibentuk fungsi pembangkit bilangan acak semu menggunakan matriks quasigrup dengan orde terbaik adalah 14. Fungsi pembangkit bilangan acak semu dengan quasigrup berorde 14 memiliki nilai *P-Values* uji monobit dan *block frequency* tertinggi dibandingkan dengan orde lain dan fungsi *default* Python. Untuk hasil citra stego yang dihasilkan, berdasarkan nilai PSNR nilai citra stego yang memiliki urutan bitplane teracak dengan fungsi pengacak quasigrup sedikit lebih rendah dibandingkan citra stego tanpa pengacakan. Namun kualitas citra stego hasil pengacakan lebih baik dari segi *imperceptibility* karena sulit untuk dibedakan dengan mata manusia biasa dibandingkan hasil citra stego tanpa pengacakan.

Adapun hal-hal yang dapat ditingkatkan apabila penelitian ini dilanjutkan adalah sebagai berikut :

1. Mengoptimasi kecepatan komputasi dari algoritma pengacakan
2. Mengurangi pemakaian memori untuk menyimpan matriks quasigrup
3. Membuat im (Pseudorandom Number Generator, n.d.) (Randomness, n.d.) (BPCS-Steganography, n.d.) (Fisher Yates, n.d.)plementasi yang tidak memerlukan *seed string* untuk memulai pengacakan matriks quasigrup

REFERENSI

[1]*BPCS-Steganography*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/BPCS-Steganography

[2] *Fisher Yates*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

[3] Godavarty, V. K. (2011). Using Quasigroups for Generating Pseudorandom Numbers.

[4] Matthew , B., & Parakh, A. (2012). A Quasigroup Based Random Number Generator for Resource Constrained Environments.

[5] *Pseudorandom Number Generator*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Pseudorandom_number_generator

[6]*Randomness*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Randomness

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makal (Godavarty, 2011)ah orang lain, dan bukan plagiasi.

Bandung, 10 Mei 2018

Penulis