E-ISSN : 2540-8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 06, Nomor 02, Desember 2021: 400–405

DESAIN PEMBANGKIT KUNCI BLOCK CIPHER BERBASIS CSPRNG CHAOS MENGGUNAKAN FUNGSI TRIGONOMETRI

Kuniarti Paraditasari[1], **Alz Danny Wowor**[2]
[1,2]Program Studi Teknik Informatika, Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Email: 672017291@student.uksw.edu[1], alzdanny.wowor@uksw.edu[2]

Abstrak

Banyak fungsi pembangkit yang dapat menghasilkan bilangan acak berbasis CSPRNG chaos salah satunya seperti fungsi logistik dan fungsi polinomial. Penelitian ini meregenerasi fungsi trigonometri dan dijadikan sebagai pembangkit bilangan acak berbasis CSPNRG *chaos*. Setiap fungsi trigonometri dapat digunakan sebagai fungsi pembangkit, walaupun untuk $\cos x$ dan $\csc x$ memerlukan sedikit modifikasi. Pengujian enkripsi menunjukkan setiap kunci yang dihasilkan dari setiap fungsi pembangkit, dapat membuat plainteks dan cipherteks tidak berhubungan secara statistik. Kondisi ini akan mempersulit kriptanalis untuk melakukan serangan pada algoritma. Secara spesifik, rancangan algoritma merupakan proses yang baik, karena dapat menghasilkan bilangan acak berbasis CSPRNG *chaos*. Sehingga fungsi trigonometri dapat digunakan sebagai pembangkit dalam menghasilkan kunci pada kriptografi *block cipher*.

Kata Kunci: Fungsi Trigonometri, CSPRNG Chaos, Fungsi Pembangkit, Block Cipher

Abstract

There are many generating functions that can generate random numbers based on CSPRNG chaos, one of which is the logistic function and the polynomial function. This research regenerates trigonometric functions and is used as a random number generator based on CSPNRG chaos. Any trigonometric function can be used as a generating function, although $\cos x$ and $\csc x$ require some modification. Encryption testing shows that each key generated from each generating function can make plaintext and ciphertext statistically unrelated. This condition will make it difficult for cryptanalysts to attack the algorithm. Specifically, the algorithm design is a good process, because it can generate random numbers based on CSPRNG chaos. So that the trigonometric function can be used as a generator in generating keys in block cipher cryptography.

KeyWords : Trigonometry Function, CSPRNG Chaos, Generating Function, Block Cipher

I. PENDAHULUAN

SETIAP algoritma kriptografi akan memperhatikan proses pembangkitan kunci, karena menjadi variabel utama untuk menghilangkan atau menyamarkan plainteks menjadi cipherteks. Banyak metode yang dapat digunakan, salah satunya dengan menggunakan pembangkit bilangan acak yang menggunakan sebuah fungsi.

Fungsi yang berhasil menjadi pembangkit bilangan acak biasanya akan selalu diupayakan sehingga memenuhi sifat *Cryptographically Secure Pseudorandom Generator* (CSPNRG) berbasis Chaos, yaitu peka terhadap perubahan kecil pada input atau nilai awal. Sebuah fungsi pembangkit bilangan acak yang berhasil memenuhi sifat CSPRNG Chaos maka bilangan tersebut dapat digunakan sebagai kunci pada kriptografi, dan memberikan jaminan untuk sebuah algortima memenuhi prinsip Shanonn.

Banyak fungsi pembangkit yang dapat menghasilkan bilangan acak berbasis CSPRNG *chaos*, yang paling terkenal adalah fungsi logistik $f(x) = rx(1-x)$ yang dalam iterasi $x_{i+1} = rx_i(1-x_i)$, dan Penelitian [1]–[4] juga menggunakan fungsi logistik sebagai pembangkit untuk menghasilkan sederatan bilangan acak, yang kemudian digunakan sebagai kunci dalam algoritma. Penelitian [5] menggunakan polinomial derajat-1, derajat-2, dan derajat-3, kemudian setiap polinomial diubah menjadi fungsi iterasi dengan metode *fixed poin iteration* dan dapat menghasilkan beberpa fungsi iterasi yang kemudian beberapa fungsi iterasi berhasil menghasilkan kunci berbasis CSPRNG chaos. Penelitian [6] juga mencari fungsi polinomial derajat tiga yang dapat dijadikan sebagai pembangkit bilangan acak. Diperoleh fungsi $f(x) = 3(x^3 - x^2 - x) + 2$ sebagai salah satu yang memenuhi dan berhasil memperoleh bilangan acak berbasis CSPRNG chaos.

Penggunaan fungsi yang polinomial dapat memberikan efek difusi dan konfusi pada algoritma, karena bilangan acak yang dihasilkan dapat memenuhi sifat *butterfly effect*, dimana perubahan kecil pada *input* dan akan mengakibatkan perubahan signifikan terjadi pada *output*. Penelitian yang dilakukan saat ini melakukan pendekatan dengan fungsi yang berbeda dengan penelitian sebelumnya, yaitu menggunakan fungsi trigonometri. Ke-enam nisbah trigonometri yaitu $\sin x, \cos x, \tan x, \sec x, \csc x$, dan $\cot x$ dijadikan sebagai generator untuk menghasilkan bilangan acak berbasis CSPRNG chaos. Penelitian [7]–[9] sebelumnya juga menggunakan fungsi trigonemetri sebagai pembangkit bilangan acak. Tetapi pendekatan dalam menggunakan fungsi berbeda, penelitian ini menggunakan fungsi pemotongan untuk mengambil bilangan bulat dari mantisa hasil iterasi.

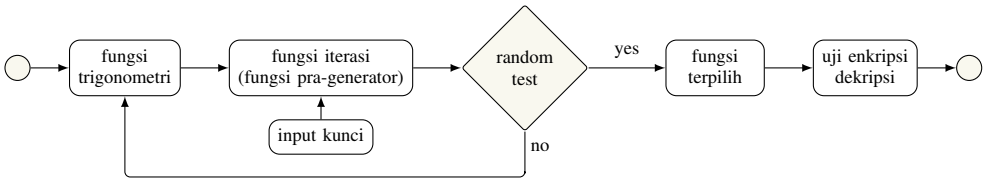
Pengujian visualisasi menggunakan *scatter plot*, uji statistika, uji keacakan [10], dan juga proses enkripsi menjadi metode pengujian untuk memastikan apakah setiap fungsi trigonometri dapat dijadikan sebagai pembangkit bilangan acak dan kemudian dapat digunakan sebagai kunci pada proses enkripsi.

E-ISSN : 2540-8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 06, Nomor 02, Desember 2021: 400–405

II. Skema Penelitian

Skema penelitian merupakan alur kerja yang dilakukan untuk menuju pada tujuan penelitian berdasarkan masalah penelitian yang telah ditunjukkan pada bagian sebelumnya. Langkah penelitian diberikan dalam beberapa tahapan, dimana setiap tahap secara garis besar diberikan pada Gambar 1. Proses awal, enam nisbah trigonometri $\sin x, \cos x, \tan x, \sec x, \csc x$, dan $\cot x$ akan dijadikan sebagai fungsi, dan kemudian akan dilanjutkan pada tahapan berikut yaitu mengubah fungsi trigonometri menjadi fungsi iterasi. Fungsi iterasi dilakukan dengan menggunakan nilai input x_j, yang akan digunakan untuk mendapatkan nilai selanjutnya x_{j+1} dimana j akan dimulai dari 0. Pengujian keacakan (*random test*) dilakukan untuk setiap fungsi trigonometri standar, apabila fungsi tersebut memenuhi sifat keacakan maka akan dilanjutkan pada tahap selanjutnya, apabila tidak maka proses modifikasi perlu dilakukan dan kemudian akan dilakukan pengujian kembali. Setiap fungsi yang lolos pengujian akan menjadi fungsi terpilih, atau dapat dikatakan sebagai fungsi pembangkit bilangan acak berbasis CSPRNG *chaos*. Proses terakhir adalah menguji setiap kunci yang diperoleh dalam proses kriptografi blok cipher.



Gambar 1: Skema Alur Penelitian

Kajian chaos dapat terlihat apabila setiap luaran divisualisasi ke dalam koordinat Cartesius, dimana iterasi sebagai absis dan ordinatnya adalah hasil iterasi. Apabila pada diagram menunjukkan sebuah pola, atau bentuk tertentu yang memudahkan penebakan maka fungsi tersebut gagal sebagai pembangkit. Sebaliknya, grafik yang membentuk chaos akan terlihat setiap hasil iterasi nampak pecah dan tidak mempunyai kecenderungan membentuk sebuah pola.

III. Hasil dan Pembahasan

A. Pencarian Fungsi Trigonometri

Pencarian pembangkit bilangan acak digunakan untuk setiap fungsi trigonometri. Proses pencarian dilakukan menggunakan alur pada Gambar 1. Fungsi pra-generator dilakukan dengan mengubah setiap fungsi trigonometri menjadi fungsi iterasi seperti yang ditunjukkan pada Persamaan 1.

$$y = f(x) \;\rightarrow\; x_{i+1} = f(x_i) \tag{1}$$

Apabila setiap fungsi dapat menghasilkan urutan bilangan dan lolos pengujian keacakan, maka fungsi tersebut dapat digunakan sebagai fungsi pembangkit bilangan acak. Pengujian pertama adalah fungsi $f(x) = \sin x$, berdasarkan Persamaan 1, diperoleh fungsi iterasi $x_{i-1} = \sin x_i$. Hasil dari lima belas iterasi pertama diberikan pada Tabel I.

Tabel I: Hasil Iterasi Fungsi $y = \sin x$

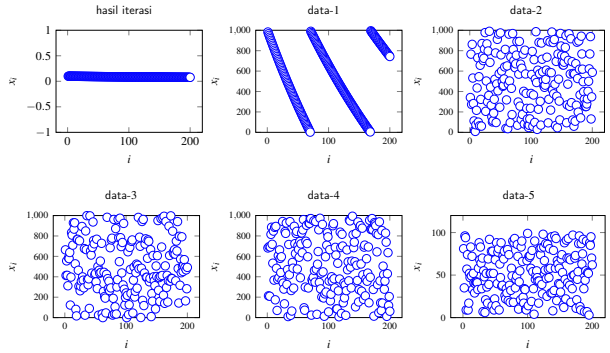
i	x_i	Data 1	Data 2	Data 3	Data 4	Data 5
1	0.099833416646828	983	341	664	682	81
2	0.099667664132134	966	766	413	213	35
3	0.099502735566960	950	273	556	695	96
4	0.099338624142099	933	862	414	209	94
5	0.099175323126911	917	532	312	691	9
6	0.099012825868156	901	282	586	815	57
7	0.098851125788857	885	112	578	885	74
8	0.098690216387181	869	21	638	718	9
9	0.098530091235332	853	9	123	533	17
10	0.098370743978475	837	74	397	847	52
11	0.098212168333675	821	216	833	367	47
12	0.098054358088848	805	435	808	884	83
13	0.097897307101745	789	730	710	174	49
14	0.097741009298937	774	100	929	893	72
15	0.097585458674833	758	545	867	483	33

Tabel I menunjukkan bagaimana mengambil *integer* dari mantisa, pada fungsi iterasi $x_{i+1} = \sin x_i$ menghasilkan 5 data. Penelitian ini mengambil tiga digit untuk setiap data, karena memperhatikan maksimum *digit* dari karakter ASCII sebanyak 256 karakter. Hasil iterasi diberikan pada Gambar 2 untuk 200 iterasi pertama.

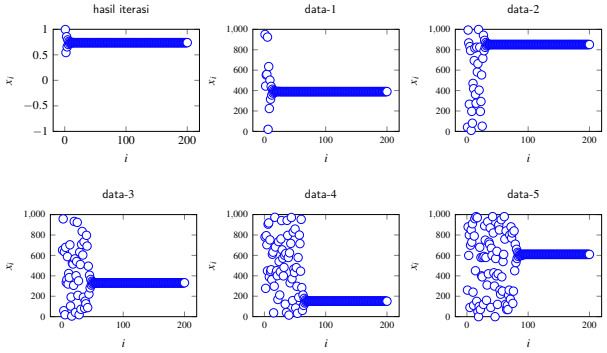
Hasil iterasi dari $x_{i+1} = \sin x_i$ pada Gambar 2, nampak secara visual hanya data-3, data-4, dan data-5 yang menghasilkan diagram *Scatter* yang sudah membentuk *chaos*. Ketiga bilangan ini dapat dilanjutkan pada pengujian selanjutnya, tetapi untuk data-1 dan data-2 masih gagal dalam uji visualisasi. Data-2 walaupun diagramnya sudah pecah, tetapi masih membentuk pola.

E-ISSN : 2540-8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 06, Nomor 02, Desember 2021: 400–405

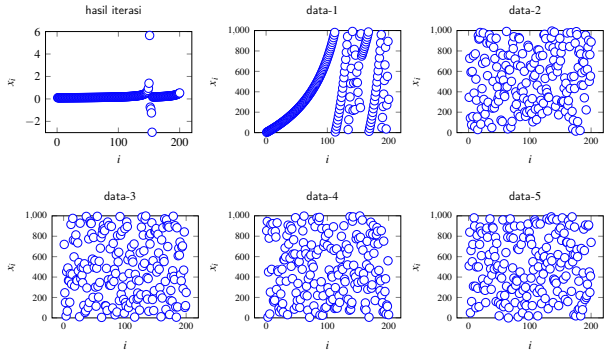


Gambar 2: Hasil Iterasi $x_{i+1} = \sin x_i$.

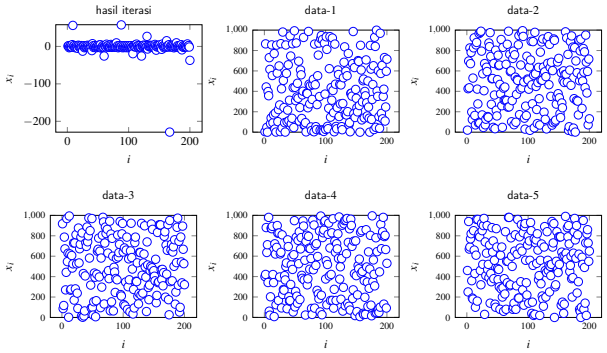


Gambar 3: Hasil Iterasi $x_{i+1} = \cos x_i$.

Pengujian yang kedua adalah untuk fungsi $f(x) = \cos x$ dalam bentuk iterasi adalah $x_{i+1} = \cos x_i$. Digunakan input $x_0 = 0.1$, hasil 200 iterasi pertama diberikan pada Gambar 3. Hasil pada fungsi $f(x) = \cos x$ berbeda dengan $f(x) = \sin x$, semua hasil iterasi untuk setiap data-i, $i = 1, \cdots, 4$ gagal menghasilkan bilangan acak.

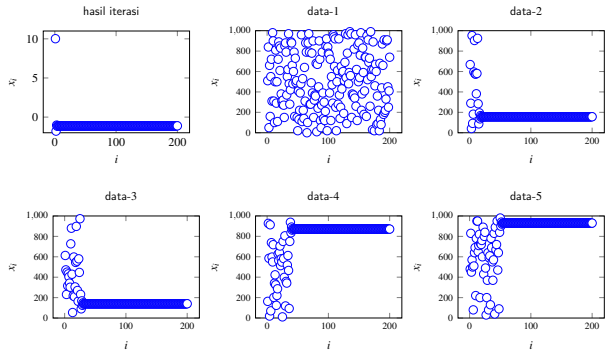


Gambar 4: Hasil Iterasi $x_{i+1} = \tan x_i$.

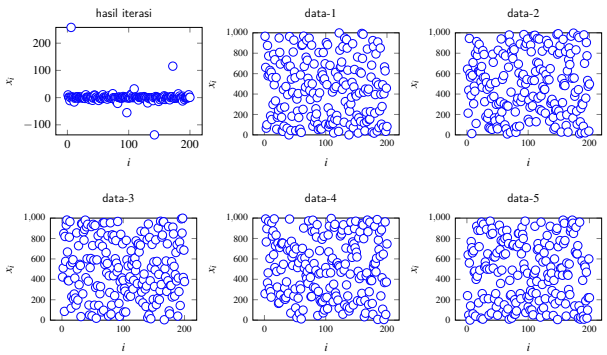


Gambar 5: Hasil Iterasi $x_{i+1} = \sec x_i$.

Hasil iterasi untuk fungi $f(x) = \tan x$ atau dalam bentuk iterasi adalah $x_{i+1} = \tan x_i$. Input $x_0 = 0.1$ sebagai inisialisasi maka dihasilkan 200 iterasi pertama yang diberikan pada Gambar 4, berhasil menghasilkan diagram *Scatter* yang sudah acak. Hanya data-1 yang gagal kerena membentuk pola tertentu.



Gambar 6: Hasil Iterasi $x_{i+1} = \csc x_i$.



Gambar 7: Hasil Iterasi $x_{i+1} = \cot x_i$.

Fungsi $f(x) = \sec x$ juga berhasil memperoleh hasil iterasi yang baik seperti pada fungsi $f(x) = \tan x$. Berdasarkan Gambar 5, data-2, data-3, data-4, dan data-5 menghasilkan diagram *Scatter* yang acak, dan pada data-1 walaupun pada iterasi tertentu membentuk sebuah pola tetapi hasil iterasi lainnya nampak sudah acak.

Gambar 6 adalah hasil iterasi dari $x_{i+1} = \csc x_i$ dengan $x_0 = 0.1$, diperoleh data-i, untuk $i = 1, \cdots, 5$ semuanya membentuk sebuah pola berupa garis lurus. Walaupun ada beberapa bilangan berbeda yang dihasilkan, tetapi gagal untuk 200 iterasi pertama yang acak.

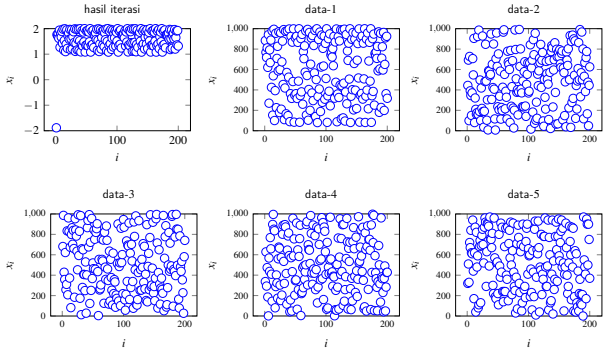
Fungsi $f(x) = \cot x$ dengan fungsi iterasi $x_{i+1} = \cot x_i$ dan input $x_0 = 0.1$, berhasil mendapatkan diagram *Scatter* yang sudah *chaos* untuk data-2, data-3, data-4, dan data-5. $f(x) = \cot x$ menjadi salah satu fungsi yang berhasil dengan baik dalam menghasilkan bilangan acak, sehingga dapat digunakan sebagai fungsi pembangkit.

E-ISSN : 2540-8984

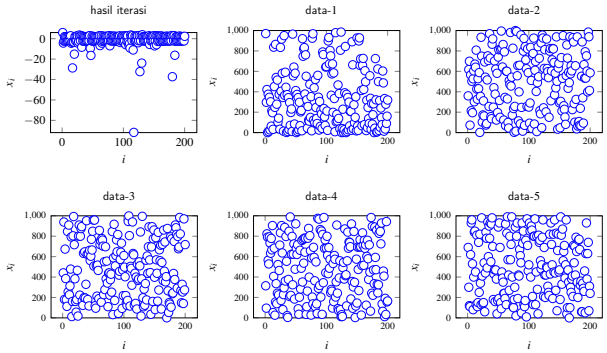
JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 06, Nomor 02, Desember 2021: 400–405

B. Modifikasi Fungsi Triginometri

Modifikasi dilakukan apabila fungsi trigonometri tidak berhasil menghasilkan bilangan acak berbasis *chaos* pada pengujian pertama. Oleh karena itu modifikasi hanya dilakukan pada fungsi $f(x) = \cos x$ dan $f(x) = \csc x$, dengan mengubah setiap x dengan $(3-2x)$ dan kemudain dikalikan dengan konstanta 2. Sehingga diperoleh fungsi yang baru sebagai hasil modifikasi adalah $f(x) = 2\cos(3-2x)$ dan $f(x) = 2\csc(3-2x)$.



Gambar 8: Hasil Iterasi $x_{i+1} = 2\cos(3-2x_i)$.



Gambar 9: Hasil Iterasi $x_{i+1} = 2\csc(3-2x_i)$.

Digunakan nilai inisialisasi $x_0 = 0.1$, diperoleh secara berturut-turut hasil dari fungsi iterasi $x_{i-1} = 2\cos(3-2x_i)$ diberikan pada Gambar 8, dan Gambar 9 untuk fungsi iterasi $x_{i-1} = 2\csc(3-2x_i)$. Fungsi $x_{i-1} = 2\cos(3-2x_i)$ dapat menghasilkan lima data-i, $i = 1, \cdots, 5$ dalam bentuk chaos, yang sebelumnya tidak ada data yang berhasil. Sedangakan $x_{i-1} = 2\csc(3-2x_i)$ dapat menghasilkan 4 bilangan acak, yang sebelumnya juga tidak dapat menghasilkan bilangan acak.

C. Pengujian Keacakan

Setiap fungsi yang berhasil menghasilkan diagram *Scatter* yang *chaos*, selanjutnya akan dilakukan pengujian bilangan acak dengan metode *Mono Bit* dan *Run Test*. Pengujian dilakuan dengan mengambil $\alpha = 1\%$ dan apabila *p-value* $> \alpha$ maka data set dikatakan acak, dan sebaliknya tidak acak.

Tabel II: Pengujian Keacakan Fungsi

Fungsi	Data	*p-value*		Hasil	Fungsi	Data	*p-value*		Hasil
		Mono Bit	Run-Test				Mono Bit	Run-Test	
$x_i = \sin(x_{i-1})$	3	0.2030	0.5005	acak	$x_i = \cot(x_{i-1})$	3	0.2578	0.5018	acak
$x_i = \sin(x_{i-1})$	4	0.3961	0.5015	acak	$x_i = \cot(x_{i-1})$	4	0.5716	0.4986	acak
$x_i = \sin(x_{i-1})$	5	0.5716	0.4986	acak	$x_i = \cot(x_{i-1})$	5	0.8875	0.5000	acak
$x_i = \tan(x_{i-1})$	2	0.8875	0.4994	acak	$x_i = 2\cos(3-2x_{i-1})$	1	0.5716	0.5008	acak
$x_i = \tan(x_{i-1})$	3	0.3961	0.4989	acak	$x_i = 2\cos(3-2x_{i-1})$	2	0.7772	0.5002	acak
$x_i = \tan(x_{i-1})$	4	0.6713	0.5005	acak	$x_i = 2\cos(3-2x_{i-1})$	3	0.3221	0.5032	acak
$x_i = \tan(x_{i-1})$	5	0.5716	0.5008	acak	$x_i = 2\cos(3-2x_{i-1})$	4	0.3961	0.4981	acak
$x_i = \sec(x_{i-1})$	2	0.4795	0.5012	acak	$x_i = 2\cos(3-2x_{i-1})$	5	0.8875	0.4994	acak
$x_i = \sec(x_{i-1})$	3	0.3221	0.5009	acak	$x_i = 2\csc(3-2x_{i-1})$	2	0.8875	0.5000	acak
$x_i = \sec(x_{i-1})$	4	0.8875	0.4994	acak	$x_i = 2\csc(3-2x_{i-1})$	3	4.2902	0.4809	acak
$x_i = \sec(x_{i-1})$	5	0.4795	0.5012	acak	$x_i = 2\csc(3-2x_{i-1})$	4	1.0073	0.4875	acak
$x_i = \cot(x_{i-1})$	2	0.8875	0.4994	acak	$x_i = 2\csc(3-2x_{i-1})$	5	1.7036	0.4877	acak

Hasil pengujian untuk kedua metode secara berturut-turut diberikan pada Tabel II, data dari setiap fungsi memenuhi pengujian *mono bit* dan *run test*, sehingga secara keseluruhan hasilnya adalah acak. Hasil menunjukkan penggunaan fungsi trigonometri sebagai fungsi pembangkit bilangan acak berhasil dengan baik, sehingga setiap fungsi pembangkit yang diberikan pada Tabel 1, berhasil menjadi pembangkit yang akan menghasilkan bilangan acak berbasis CSPRNG *chaos*.

Semua data dari fungsi yang telah berhasil mempunyai digaram *Scatter* yang tidak berpola, mempunyai nilai *p-value* yang lebih besar dari $\alpha = 0.01$, Hasil ini menggambarkan keunikan dari fungsi trigonometri dalam proses iterasi, berbeda dengan penggunaan fungsi polinomial, yang telah dilakukan sebelumnya.

D. Pengujian Kriptografi Blok Cipher

Pengujian ini dilakukan dalam dua bagian, pertama adalah pengujian korelasi antara plainteks dan cipherteks. Kedua adalah pengujian *butterfly effect*, dimana akan dilihat perubahan kecil pada inisialisasi x_0 apakah akan menghasilkan perubahan yang besar pada cipherteks. Dimana pengujian ini untuk melihat seberapa baik kunci memberikan peran untuk menyamarkan cipherteks.

$$E_k : \ P + K = C \ (mod \ 256) \tag{2}$$

E-ISSN : 2540-8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 06, Nomor 02, Desember 2021: 400–405

Setiap fungsi iterasi yang menghasilkan bilangan acak berbasis CSPRNG *chaos* digunakan sebagai kunci dalam blok cipher. Pengujian dilakukan dengan proses enkripsi $E_k : P+K = C \bmod (256)$, dimana P, C, dan K secara berturut-turut adalah plainteks, cipherteks dan kunci. Pengambilan modulus 256 karena karakter ASCII berada dalam 256 karakter.

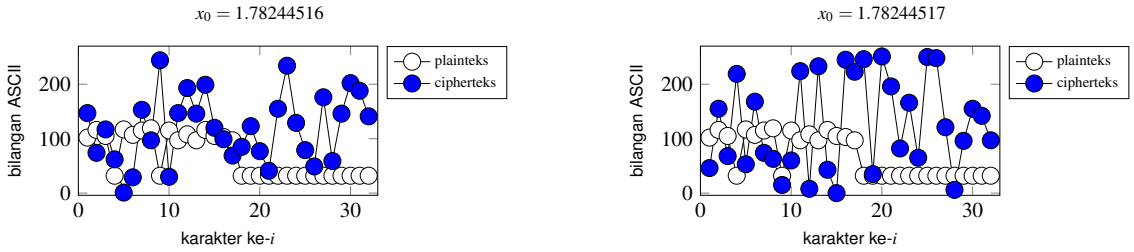
Diambil plainteks "fti uksw salatiga", dan digunakan ukuran *block* kunci sebesar 256 bit atau sebanding dengan 32 karakter. Pengujian pertama adalah dengan menguji setiap data dari setiap fungsi yang telah berhasil menjadi bilangan acak berbasis CSPRNG *chaos* untuk dijadikan kunci. Sehingga kunci adalah 32 karakter dari masing-masing data, dan dilakukan operasi enkripsi berdasarkan Persamaan 2, dan menguji korelasi antara plainteks dan cipherteks. Hasil dari setiap pengujian diberikan pada Tabel III.

Tabel III: Pengujian Enkripsi dengan Kunci 256 Bit

Fungsi Iterasi	Kunci	Korelasi	Fungsi Iterasi	Kunci	Korelasi
$x_i = \sin(x_{i-1})$	*data* 3	−0.0002	$x_i = \cot(x_{i-1})$	*data* 3	0.0059
$x_i = \sin(x_{i-1})$	*data* 4	0.0008	$x_i = \cot(x_{i-1})$	*data* 4	0.0008
$x_i = \sin(x_{i-1})$	*data* 5	0.0018	$x_i = \cot(x_{i-1})$	*data* 5	0.0002
$x_i = \tan(x_{i-1})$	*data* 2	0.0015	$x_i = 2\cos(3-2x_{i-1})$	*data* 1	−0.0082
$x_i = \tan(x_{i-1})$	*data* 3	−0.0001	$x_i = 2\cos(3-2x_{i-1})$	*data* 2	0.0065
$x_i = \tan(x_{i-1})$	*data* 4	−0.0021	$x_i = 2\cos(3-2x_{i-1})$	*data* 3	0.0008
$x_i = \tan(x_{i-1})$	*data* 5	0.0004	$x_i = 2\cos(3-2x_{i-1})$	*data* 4	0.0014
$x_i = \sec(x_{i-1})$	*data* 2	−0.0009	$x_i = 2\cos(3-2x_{i-1})$	*data* 5	0.0067
$x_i = \sec(x_{i-1})$	*data* 3	−0.0019	$x_i = 2\csc(3-2x_{i-1})$	*data* 2	0.0092
$x_i = \sec(x_{i-1})$	*data* 4	0.0014	$x_i = 2\csc(3-2x_{i-1})$	*data* 3	−0.0027
$x_i = \sec(x_{i-1})$	*data* 5	0.0060	$x_i = 2\csc(3-2x_{i-1})$	*data* 4	0.0025
$x_i = \cot(x_{i-1})$	*data* 2	0.0031	$x_i = 2\csc(3-2x_{i-1})$	*data* 5	−0.0078

Hasil dari pengujian korelasi dapat digunakan untuk melihat kekuatan kunci dalam proses enkripsi, sehingga seberapa baik kunci dapat mengamankan informasi pada *block cipher*. Nilai korelasi negatif maupun positif tidak terlalu diperhatikan, yang dilihat hanya seberapa dekat dengan 0. Hasil yang sangat baik untuk semua pengujian pada Tabel 3, dimana nilai korelasi sangat dekat dengan 0, baik itu berbanding terbalik atau berbanding lurus.

Hasil ini menunjukkan pembangkit kunci dengan fungsi trigonometri mampu membuat plainteks dan cipherteks tidak berhubungan secara statistik. Pengujian ini memberikan informasi bahwa kekuatan kunci dapat membuat kriptanalisis akan kesulitan untuk mencari relasi plainteks dengan cipherteks. Dengan demikian setiap fungsi pembangkit yang diberikan pada Tabel 3, dapat memberikan efek difusi-konfusi pada sebuah algoritma, hal inilah yang perlu diperhatikan oleh para kriptografer dalam merancang algoritma kriptografi.



Gambar 10: *Perbandingan Plainteks-Cipherteks*

Berikut adalah pengujian *butterfly effect*, diambil plainteks “fti uksw salatiga”, dan digunakan ukuran *block* sebesar 256 bit atau sebanding dengan 32 karakter. Simulasi dilakukan dengan memilih $x_i = \sin x_i$, khususnya pada *data*-4 untuk menjadi kunci. Dipilih sembarang dua nilai inisialisasi $x_0 = 1.78244517$ dan $x_0 = 1.78244516$, dimana keduanya adalah konstanta yang mempunyai perbedaan yang begitu kecil yaitu 10^{-8}.

Hasil pengujian yang diberikan dalam koordinat Cartesius untuk plainteks dan cipherteks seperti yang diberikan pada Gambar 10. Perbedaan nilai inisialisasi sebesar 10^{-8} untuk fungsi $x_{i-1} = \sin x_i$, dapat menghasilkan cipherteks yang sangat berbeda. Hasil ini menunjukkan bahwa fungsi trigonometri juga dapat memenuhi sifat *butterfly effect*, dimana perubahan kecil pada input fungsi pembangkit, tetapi kunci yang dihasilkan dapat mengubah cipherteks secara signifikan.

IV. Simpulan

Pembangkitan kunci menggunakan fungsi trigonometri sangat baik, karena dapat menghasilkan bilangan acak berbasis CSPRNG *chaos*. Walaupun untuk fungsi $\cos x$ dan $\csc x$ memerlukan modifikasi untuk menemukan bilangan acak dengan input inisialisasi $x_0 = 0.1$. Pengambilan tiga bilangan pada mantisa, fungsi trigonometri dapat menghasilkan data set bilangan acak yang lebih banyak, rata-rata untuk setiap fungsi pembangkit dapat menghasilkan 3-5 data set yang memenuhi sifat CSPRNG *chaos*.

Pengujian visualisasi dan juga pengujian keacakan dengan metode *mono bit* dan *run test*, kemudian dilanjutkan dengan pengujian enkripsi, diperoleh bahwa fungsi trigonometri berhasil dalam menghasilkan bilangan acak yang dapat dijadikan sebagai kunci pada kriptografi blok cipher. Pengujian korelasi antara plainteks-cipherteks dapat menghasilkan nilai korelasi yang sangat mendekati nol, sehingga fungsi trigonometri dapat menghasilkan kunci yang dapat memenuhi sifat difusi-konfusi pada algoritma kriptografi, sehingga akan mempersulit kriptanalisis melakukan kriptanalisis dengan mencari hubungan secara langsung antara cipherteks dan plainteks.

PUSTAKA

[1] H. Solís-Sánchez and E. G. Barrantes, "Using the logistic coupled map for public key cryptography under a distributed dynamics encryption scheme," *Information*, vol. 9, no. 7, 2018. [Online]. Available: https://www.mdpi.com/2078-2489/9/7/160

[2] A. D. Wowor and et. all, "Domain examination of chaos logistich function as a key operator in cryptography," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 6, pp. 4577–4583, 2018.

[3] M. Lawnik, "Generalized logistic map and its application in chaos based cryptography," *Journal of Physics: Conference Series*, vol. 936, p. 012017, dec 2017. [Online]. Available: https://doi.org/10.1088/1742-6596/936/1/012017

[4] Y. Guodong, J. Kaixin, P. Chen, and H. Xiaoling, "An effective framework for chaotic image encryption based on 3d logistic map," *Security and Communication Networks*, vol. 936, p. 012017, dec 2018. [Online]. Available: https://www.hindawi.com/journals/scn/2018/8402578/

[5] A. D. Wowor, "Regenerasi fungsi polinomial dalam rancangan algoritma berbasis csprng chaos sebagai pembangkit kunci pada kriptografi block cipher," *Limits Journal*, vol. 14, no. 1, pp. 1–15, 2017.

[6] R. Yopeng, M and A. D. Wowor, "The implementation of $f(x) = 3(x^3 - x^2 - x) + 2$ as csprng chaos-based random number generator," *ndonesian Journal on Computing*, vol. 6, no. 1, pp. 41–52, 2021.

[7] J. Yetao, T. Jiaze, Z. Hanfeng, G. Wenyong, L. Guoxi, C. Huiling, and W. Mingjing, "An Adaptive Chaotic Sine Cosine Algorithm for Constrained and Unconstrained Optimization," *Hindawi*, vol. 2020, pp. 1–36, 2020. [Online]. Available: https://doi.org/10.1155/2020/6084917

[8] A. Moatsum, S. Azman, T. Je Sen, and H. A. Wafa', "Digital Cosine Chaotic Map for Cryptographic Applications," *IEEE*, vol. 7, pp. 150 609–150 622, 2019.

[9] L. Albert and M. K.-H. Kiessling, "Order and chaos in some deterministic infinite trigonometric products," *Journal of Statistical Physics*, vol. 168, no. 3, p. 595–619, Jun 2017. [Online]. Available: http://dx.doi.org/10.1007/s10955-017-1811-1

[10] L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, N. A. Heckert, J. F. Dray, and S. Vo, "Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications," Gaithersburg, MD, USA, Tech. Rep., 2010.