IMPLEMENTASI ALGORITME GRAIN UNTUK PENGAMANAN DATA REKAM MEDIS

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Yoga Rizwan Priyatna
NIM: 145150200111105

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2018

PENGESAHAN

IMPLEMENTASI ALGORITME GRAIN UNTUK PENGAMANAN DATA REKAM MEDIS

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Yoga Rizwan Priyatna
NIM: 145150200111105

Skripsi ini telah diuji dan dinyatakan lulus pada
18 Desember 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T, M.Sc.	Mahendra Data, S.Kom., M.Kom
NIP: 19831228 201803 2 002	NIK: 2015038611171001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

i

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 18 Desember 2018

Yoga Rizwan Priyatna

NIM: 145150200111105

ii

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh. Alhamdulillah segala puji syukur saya haturkan kepada Allah SWT atas limpahan rahmat dan hidayahnya penulisan skripsi saya yang berjudul 'IMPLEMENTASI ALGORITME GRAIN UNTUK PENGAMANAN DATA REKAM MEDIS' dapat terselesaikan. Shalawat serta salam senantiasa saya curahkan untuk Baginda Rasulullah Muhammad SAW dan para sahabat.

Bantuan, bimbingan, serta dorongan dari banyak pihak memberikan semangat kepada saya dalam penyelesaian penulisan skripsi ini. Oleh karenanya, melalui tulisan ini saya ingin menyampaikan ucapan terima kasih kepada :

1. Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing pertama yang telah memberikan bimbingan, saran, motivasi, dan pengarahan dalam penyusunan skripsi ini.
2. Bapak Mahendra Data, S.Kom., M.Kom selaku dosen pembimbing kedua yang telah memberikan bimbingan penulisan, saran, pengarahan, serta koreksi dalam penyusunan skripsi ini.
3. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D., Bapak Ir. Heru Nurwasito, M.Kom., Bapak Drs. Mardji, M.T, dan Bapak Edy Santoso, S.Si., selaku Dekan, Wakil Dekan I, Wakil dekan II, dan Wakil Dekan III Fakultas Ilmu Komputer Universitas Brawijaya.
4. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. dan Bapak Agus Wahyu Widodo, S.T, M.Sc selaku Ketua Jurusan Teknik Infomatika, dan Kepala Program Studi Teknik Informatika Fakultas Ilmu komputer Universitas Brawijaya.
5. Ibu Tutik Sri Mulyati dan Ayah Hairul Umam yang sudah memberikan dukungan berupa moril dan materiil, kasih sayang, serta doa yang tiada henti untuk kelancaran penyusunan skripsi ini.
6. Seluruh Dosen serta segenap karyawan Fakultas Ilmu Komputer Universitas Brawijaya yang telah memberikan ilmu bermanfaat dan membantu dalam menyelesaikan skripsi ini.
7. Yosi Syafira, Afro Aransa Sahat, Dimas Gufron Ash-Shidiq, dan Furqoni Nurul Ummah terima kasih untuk waktu, semangat, serta dukungan berupa nasihat yang luar biasa.
8. Sahabat saya : Anton Firdaus, Paul Manason, Galang Perdana, Sastra Ginata, Gogot Alam, Annam Rosyadi, Iskar Maulana, dan Hilman Adi terima kasih untuk bantuan, dukungan, serta masukan yang sangat membantu dalam penyusunan skripsi ini.
9. Teman-teman KOPMA SQUAD yang luar biasa dan banyak Memotivasi saya untuk segera menyelesaikan skripsi ini.

10. Semua pihak yang telah memberikan bantuan baik secara langsung maupun tidak langsung dalam penyusunan skripsi ini.

Malang, 16 Januari 2018

Penulis

rizwan.yoga12@gmail.com

ABSTRAK

Rekam medis merupakan catatan medis seorang pasien yang memuat informasi penting seperti data pribadi dan riwayat pengobatan pada sebuah rumah sakit yang bersifat rahasia. Data rekam medis pasien pada rumah sakit masih banyak yang menggunakan media kertas yang memungkinkan dokumen hilang, rusak, serta duplikasi data. Pada kasus seorang pasien dirujuk dari rumah sakit satu ke rumah sakit lain untuk keperluan penanganan medis lebih lanjut penggunaan media kertas dapat melemahkan status kerahasiaan informasi yang terdapat pada rekam medis. Penggunaan sistem informasi yang memuat fungsi kriptografi berupa enkripsi dan dekripsi data dapat mengurangi kebocoran kerahasiaan data pada saat proses pertukaran rekam medis. Algoritme Grain v1 digunakan pada sistem untuk melakukan enkripsi dan dekripsi untuk melindungi kerahasiaan data sedangkan untuk menjamin keutuhan data pada saat pengiriman menggunakan teknik *hashing* dengan algoritme SHA-3 224. Pada penelitian ini penulis menggunakan algoritme *Diffie-Hellman Key Exchange* untuk bertukar kunci antara pengirim dan penerima agar dapat melakukan enkripsi dan dekripsi data rekam medis pasien. Pengujian yang dilakukan pada penelitian ini menunjukkan bahwa arsitektur algoritme Grain v1 yang dibuat penulis dan *library* SHA-3 224 pada penelitian ini sudah sesuai dengan arsitektur pembuatnya melalui uji *test vector.* Uji kinerja sistem pada penelitian ini menggunakan parameter waktu yang diperlukan sistem untuk melakukan enkripsi dan dekripsi data yang dikirim. Algoritme Grain v1 membutuhkan waktu enkripsi dengan rata-rata 0,0077450 detik dan rata-rata waktu yang diperlukan untuk satu kali proses dekripsi adalah sebesar 0,0145436 detik.

Kata Kunci : Rekam Medis, Kriptografi, Grain v1, SHA-3, *Diffie-Hellman Key Exchange*

ABSTRACT

Medical record is a patient record that contains crucial information such as confidential personal data and medical history on a hospital. Medical record of patients in the hospital still use paper as the medium that allows the document to be lost, damaged, and duplicated. In a case where a patient is referred from one hospital to another for further medical treatment, the use of paper media can reduce the confidentiality status of the information. Information systems that contain cryptographic functions in the form of encryption and decryption of data can reduce data leakage confidentiality during the medical record axchange process. Grain *v1 algorithm is used in the system to perform encyption and decryption which is used to protect data confidentiality, while in order to ensure data integrity in sending process, hashing technique with SHA-3 22 algorithm is used. In this research, the writer used Diffie-Hellman Key Exchange algorithm to exchange keys between the sender and receiver in order to be able to encrypt and decrypt the patient's medical record data. The examination of this research indicated that the* Grain *v1 algorithm created by the writer and the library SHA-3 224 were suitable with the writer architecture through test vector examination. System performance testing in this research used time parameters needed by the system to encrypt and decrypt data sent.* Grain *v1 algorithm required encryption time with an average of 0.0077450 seconds and the average time needed for the decryption process was 0.0145436 seconds.*

Keywords *: Medical record, Cryptography,* Grain *v1, SHA-3, Diffie-Hellman key exchange*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Pemanfaatan teknologi informasi untuk membuat sebuah aplikasi yang dapat digunakan sebagai media penyimpanan data-data rekam medis pasien dapat mempercepat pelayanan di rumah sakit. Menurut Oktariano (2016) beberapa rumah sakit masih menggunakan cara manual dalam melakukan penyimpanan, pengolahan serta penyajian data rekam medis pasien. Penggunaan cara manual seperti memakai media kertas untuk penyajian data rekam medis pasien mempunyai kelemahan seperti duplikasi data, hilangnya dokumen *hard copy,* serta melemahkan status kerahasiaan dokumen rekam medis pasien yang disebabkan tidak teraturnya dalam penyimpanan dokumen serta seringnya terjadi kesalahan yang disebabkan oleh *human error*. Kelemahan pada penggunaan cara manual dapat menyebabkan kesalahan fatal yang bisa membahayakan pasien. Penggunaan kertas atau kartu untuk pendataan rekam medis pasien menimbulkan beberapa masalah yang menyebabkan kesalahan dan menghambat waktu yang diperlukan tenaga medis untuk melakukan diagnosis pada pasien (Bintari, et al., 2017). Sistem Informasi rumah sakit (SIRS) yang memuat fungsi rekam medis pasien sudah diterapkan pada beberapa lingkup rumah sakit untuk mempermudah pekerjaan pegawai rumah sakit seperti dokter, apoteker, dan perawat untuk melakukan penanganan terhadap pasien. SIRS yang digunakan masih pada dalam lingkup satu rumah sakit saja, tetapi dapat dikembangkan menjadi SIRS yang dapat menghubungkan rumah sakit untuk mempermudah layanan kesehatan.

Pada kasus dimana sebuah rumah sakit tidak mampu menangani pasien dengan penyakit tertentu, memungkinkan pasien untuk dirujuk pada rumah sakit lain yang memiliki sumber daya serta peralatan medis yang lebih canggih. Proses pemindahan pasien antar rumah sakit juga melibatkan rekam medis dimana segala pengobatan yang dilakukan pada rumah sakit sebelumnya tertulis pada rekam medis pasien yang kemudian dikirim bersama pasien ke rumah sakit yang dirujuk. Penggunaan cara rekam medis manual seperti saat ini dapat menyebabkan data yang berada didalam rekam medis pasien rentan terhadap manipulasi serta kebocoran kerahasiaan data pada saat pengiriman dokumen ke rumah sakit yang dirujuk.

Penggunaan SIRS yang memiliki fitur pertukaran data rekam medis pasien antar rumah sakit dapat menimbulkan ancaman pada keamanan informasi seperti celah pada kerahasiaan, integritas data dan autentikasi data yang dikirim. Peretasan informasi dan data rekam medis pasien oleh pihak tidak bertanggung jawab menjadi ancaman keamanan pada sistem. Untuk mengurangi kemungkinan peretasan data, dapat dilakukan proses kriptografi pada *file* yang dikirim. Algoritme kriptografi yang dapat digunakan adalah algoritme yang masih belum diketahui celahnya dan belum bisa didekripsi oleh pihak yang tidak memiliki akses

pada kunci enkripsi. Algoritme Grain v1 merupakan salah satu algoritme enkripsi data yang dibuat untuk memenuhi standar *project eSTREAM* dan digunakan sebagai algoritme pengganti sebelumnya yang sudah tidak dapat digunakan karena sudah dapat ditemukan kelemahannya. Algoritme Grain v1 dirancang untuk memiliki kecepatan tinggi dan menggunakan sumberdaya yang rendah (Shahid, et al., 2017). Integritas data pada saat pengiriman pesan juga perlu diperhatikan untuk menjamin data yang dikirim tidak mengalami perubahan pada saat proses pengiriman. Proses yang dapat dilakukan untuk menjamin integritas sebuah data adalah proses *hashing.* Menurut Fakhrusy (2016) Algoritme yang dapat digunakan untuk melakukan proses *hashing* adalah SHA-3 sebagai pengganti algoritme SHA-1 karena algoritme SHA-1 sudah mengalami kolisi. SHA-3 merupakan algoritme pemenang *hash competition* yang diresmikan oleh NIST *(National Institute of Standard Technology)* pada tahun 2015 sebagai algoritme *hashing* alternatif pengganti algoritme SHA-1.

Peningkatan kualitas layanan kesehatan seperti rumah sakit yang memanfaatkan SIRS dapat menjadi lebih baik apabila pada SIRS terdapat fitur pengiriman berkas rekam medis pasien yang memerlukan rujukan ke rumah sakit lain. Penggunaan kriptografi sebagai solusi keamanan informasi rekam medis pasien pada saat pengiriman data menjadikan dasar dari penelitian skripsi ini. Oleh karena itu, "Implementasi Algoritme Grain Untuk Pengamanan Data Rekam Medis" diambil sebagai topik dalam penyusunan skripsi ini.

1.2 Rumusan masalah

1. Bagaimana hasil implementasi algoritme Grain v1 dan SHA-3 224 sebagai pengaman data rekam medis pasien?
2. Bagaimana validasi *keystream* algoritme Grain v1 dengan menggunakan *test vector*?
3. Bagaimana kinerja algoritme Grain v1 pada proses enkripsi dan dekripsi data rekam medis?

1.3 Tujuan

1. Membangun sebuah sistem yang dapat mengamankan kerahasiaan dan keutuhan data rekam medis dengan menggunakan algoritme Grain v1 dan SHA-3 224 pada saat melakukan pertukaran data.
2. Mengetahui hasil validasi *keystream* algoritme Grain v1 dengan *test vector.*
3. Mengetahui hasil kinerja algoritme Grain v1 pada proses enkripsi dan dekripsi data rekam medis.

1.4 Manfaat

Penelitian ini sebagai media yang dapat menggambarkan sebuah metode yang bisa digunakan dalam upaya peningkatan kualitas layanan rumah sakit dengan memberikan layanan aplikasi yang dapat bertukar informasi rekam medis pasien

antar rumah sakit. Penulisan skripsi ini diharapkan dapat menjadi acuan pustaka untuk penulisan dan penelitian selanjutnya.

1.5 Batasan masalah

1. Sistem yang dibangun hanya bisa melakukan enkripsi dan dekripsi data berupa *string* yang kemudian disimpan pada basis data.
2. Data yang digunakan untuk proses pengujian adalah data *dummy* yang bukan merupakan data asli dan akurat dari pihak rumah sakit manapun.
3. Panjang masukan *key* untuk proses produksi *keystream* tidak bisa lebih dari 80 bit dan masukan IV tidak bisa lebih dari 64 bit.
4. IV dan *key* ditentukan secara *default* pada kode sumber belum dapat melakukan masukan untuk IV dan *key* secara manual oleh pengguna sistem.

1.6 Sistematika pembahasan

Penulisan skripsi ini terdiri dari beberapa bab untuk membantu pembaca dalam memahami sistematika pembahasan isi dalam skripsi. Uraian bab yang terdapat dalam skripsi ini terdiri dari :

BAB I : PENDAHULUAN

Pada bab ini akan dijelaskan apa masalah utama yang akan diangkat dalam pengerjaan penelitian dan membahas alasan pemilihan algoritme Grain v1 dan SHA-3 sebagai pengaman pengiriman data rekam medis pasien rumah sakit. Bab ini berisi Latar belakang masalah, Rumusan Masalah, Tujuan, Manfaat, Batasan Masalah, dan Sistematika Pembahasan.

BAB II : LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan berisi Tinjauan pustaka pada penelitian yang sudah ada sebelumnya yang berhubungan dengan penelitian ini sebagai pendukung kepustakaan. Bab ini juga berisi uraian dan pembahasan tentang teori, konsep, model, metode, atau sistem dari literatur ilmiah, yang berkaitan dengan algoritme Grain v1, SHA-3, serta literatur yang dapat menjawab pertanyaan pada penelitian ini.

BAB III : METODOLOGI PENELITIAN

Pada bab ini menjelaskan langkah-langkah yang akan ditempuh untuk melakukan penelitian. Pada bab ini terdapat penjelasan tentang cara (metode) yang lebih spesifik dalam penyelesaian masalah.

BAB IV : ANALISIS KEBUTUHAN DAN PERANCANGAN

Bab ini akan menjelaskan tahapan analisis kebutuhan dari sistem yang akan dibangun agar sesuai dengan tujuan yang ingin dicapai

dalam penelitian ini. Pada bab analisis kebutuhan juga berisi perancangan dari sistem dan hal lain yang berhubungan dengan topik penelitian.

BAB V : IMPLEMENTASI

Bab implementasi membahas proses implementasi dari dasar teori yang telah dipelajari sesuai analisis dan perancangan sistem, serta pengujian hasil implementasinya. Pengujian yang dilakukan mencakup uji vektor, dan pengujian dengan parameter kecepatan produksi *keystream* algoritme serta konsumsi *memory* dari sistem yang dibangun.

BAB VI : PENGUJIAN

Bab pengujian berisi serangkaian proses pengujian terhadap sistem untuk memastikan sistem yang dibangun memiliki kesesuaian dengan analisis kebutuhan dan perancangan. Pada bab ini pengujian terhadap sistem dilakukan dengan beberapa cara yang sesuai untuk mengetahui kinerja sistem yang sudah dibangun.

BAB VII : PENUTUP

Bagian ini memuat kesimpulan dari hasil penelitian yang dilakukan terkait dengan sistem yang mengimplementasikan algoritme Grain v1 dan SHA-3 pada pengamanan pengiriman data rekam medis. Pada bab penutup juga memuat saran terhadap yang dapat dilakukan untuk keperluan penyempurnaan penelitian ini jika ada penelitian selanjutnya yang berhubungan dengan topik penelitian ini.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Algoritme Grain v1 diperkenalkan sebagai algoritme *stream cipher* baru yang dirancang untuk perangkat keras dengan *memory* dan konsumsi daya yang rendah. Algoritme ini terdiri dari dua *shift register* dan fungsi non linier yang memiliki fitur untuk meningkatkan kecepatan pada sisi *hardware.* Kunci yang dihasilkan oleh algoritme Grain v1 adalah sebanyak 80 bit dan belum pernah ada serangan yang dapat melakukan pencarian kunci secara acak dengan cepat untuk menembus sistem keamanan algoritme ini. Perancangan algoritme Grain v1 juga berupaya untuk menggantikan algoritme enkripsi yang sudah ada sebelumnya yaitu algoritme E0 dan A5/1 (Hell, et al., 2006).

Pada *paper* yang memperkenalkan algoritme Grain v1 yang ditulis oleh Martin Hell dan kawan-kawan mengimplementasikan dengan kecepatan enkripsi data 1bit/*clock*, sedangkan algoritme *stream cipher* yang berorientasi pada enkripsi kata dapat mengenkripsi dengan kecepatan 1 kata/*clock*. Alasan mengapa algoritme Grain melakukan proses enkripsi hanya 1bit/*clock* karena algoritme ini dirancang untuk perangkat keras dengan konsumsi daya dan *memory* yang relatif kecil sehingga proses enkripsi dengan algoritme ini dapat berjalan lebih ringan dan cepat. Martin Hell mengatakan dalam papernya jika algoritme Grain v1 pada dasarnya memang dirancang dengan implementasi dasar enkripsi 1bit/*clock* namun bukan tidak mungkin jika perangkat keras yang digunakan untuk mengenkripsi data menggunakan algoritme ini memiliki sumber daya dan kecepatan yang lebih canggih maka algoritme Grain v1 dapat mengenkripsi data dengan kecepatan 16bit/*clock*.

Algoritme Grain dirancang khusus untuk digunakan pada perangkat keras yang minim sumberdaya dengan memanfaatkan gerbang logika berjumlah sesedikit mungkin namun tetap memperhatikan tingkat keamanan yang tinggi. Penggunaan gerbang logika dengan jumlah sesedikit mungkin untuk menghasilkan *keystream* dan *ciphertext* menjadikan Grain sebagai algoritme pengamanan yang baik untuk diterapkan pada perangka keras. Grain memang difokuskan untuk algoritme pengaman perangkat keras, namun algoritme ini masih memungkinkan untuk diterapkan pada perangkat lunak. Selain algoritme Grain masih banyak algoritme lain yang dirancang khusus untuk diterapkan pada perangkat lunak dengan efisiensi yang tinggi dibandingkan algoritme Grain yang diterapkan pada perangkat lunak. Oleh karena itu algoritme Grain tidak dapat dibandingkan kinerjanya dengan algoritme lain sebagai pengaman yang memang dirancang khusus untuk mengamankan perangkat lunak.

Tabel 2.1 Tinjauan Pustaka

Judul Penelitian	Metodologi	Hasil Penelitian	Kelemahan
PERANCANGAN DAN IMPLEMENTASI REKAM MEDIS PASIEN POLI UMUM DI RUMAH SAKIT RIMBO MEDICA MENGGUNAKAN PHP DAN MySQL. SCIENTIA JOURNAL, Volume 4, p. 317.	Penelitian ini berjenis penelitian kualitatif yang dilakukan dengan cara pengamatan dan studi kasus terhadap sistem rekam medis pasien di rumah sakit rimbo medica untuk menggali dan menentukan kebutuhan sistem yang akan dibangun.	Penelitian dapat meningkatkan efisiensi pekerjaan petugas sekitar 61,4% dan mempermudah managemen rumah sakit yang berkaitan dengan rekam medis pasien.	Sistem rekam medis yang dibangun belum dapat terintegrasi antar rumah sakit untuk bertukar data rekam medis jika terjadi kasus pasien rujukan. Belum adanya sistem keamanan informasi yang memadahi dapat meningkatkan resiko serangan pada sistem rekam medis yang dibangun
Implementasi *HMAC-SHA-3-Based One Time* Password pada Skema *Two-Factor Authentication*	Menerapkan Algoritme SHA-3 pada proses skema *two - factor authentication* untuk menggantikan algoritme SHA-1 yang sudah mengalami kebocoran keamanan	Algoritme SHA-3 dapat diterapkan pada skema *two – factor authentication* menggantikan SHA-1 namun dengan *cost* lebih tinggi karena algoritme SHA-3 memiliki tingkat kerumitan yang lebih tinggi dibandingkan dengan algoritme SHA-1	Perlu peningkatan keamanan pada fungsi HMAC-SHA-3 karena pergantian digit *key* sekecil apapun seharusnya dapat menghasilkan *sequence* pada HOTP yang drastis.

2.2 Sistem Informasi

Sistem informasi terdiri dari dua kata yaitu sistem dan informasi. Arti dari kata sistem merupakan sekumpulan unit yang membentuk jaringan dengan pekerjaan dan tugas yang sama untuk mencapai suatu tujuan. Informasi sendiri berarti suatu kumpulan data yang sudah diolah sehingga dapat memberikan arti dan nilai pada pembaca. Dari definisi masing-masing kata dapat disimpulkan bahwa sistem informasi berarti suatu sistem yang mengolah data untuk mendapatkan informasi yang dapat disajikan kepada penerimanya (Andalia & Setiawan, 2015). Sistem informasi sudah banyak digunakan untuk menunjang kinerja suatu instansi atau perusahaan yang bergerak pada bidang jasa seperti rumah sakit karena dapat meningkatkan efektivitas, keakuratan data, dan mempercepat pekerjaan.

Sistem informasi yang menyediakan layanan rekam medis digital (*paperless*) sudah banyak digunakan pada beberapa instansi kesehatan seperti rumah sakit dan fasilitas umum pelayanan kesehatan untuk mempermudah dan meningkatkan kinerja sebuah rumah sakit. Menurut Herman (2017) perkembangan sistem informasi sebagai penunjang kegiatan operasional sebuah rumah sakit mendukung munculnya berbagai aplikasi yang dapat meningkatkan kualitas pelayan kepada pasien. Penggunaan sistem informasi rumah sakit memungkinkan penyimpanan dokumen rekam medis pasien pada sebuah basis data yang dimiliki rumah sakit tersebut. Dokumen rekam medis yang tersimpan pada basis data rumah sakit ini akan terintegrasi pada setiap unit yang dimiliki oleh rumah sakit seperti laboratorium, apotek dan unit yang bersangkutan untuk mempermudah proses pelayanan kesehatan pasien. SIRS yang digunakan beberapa rumah sakit saat ini masih memakai koneksi intranet dimana dokumen atau data dan juga rekam medis pasien yang tersimpan pada basis data rumah sakit hanya bisa diakses oleh rumah sakit atau instansi kesehatan itu sendiri.

2.3 Rekam Medis

Rekam medis pasien menurut Undang Undang Praktik Kedokteran (2014) Pasal 46 ayat (1) adalah sebuah berkas yang berisi catatan dan dokumen tentang identitas pasien, pemeriksaan, pengobatan, tindakan dan pelayanan lain yang telah diberikan kepada pasien. Sedangkan menurut Peraturan Menkes No.749a/Menkes/Per/ XII/1989 (1989) rekam medis adalah berkas yang berisi catatan dan dokumen tentang identitas pasien, pemeriksaan, pengobatan, tindakan dan pelayanan lain kepada pasien pada sarana pelayanan kesehatan.

Data riwayat kesehatan pasien pada dunia kesehatan sangat penting untuk dokumentasi detail riwayat pengobatan selama menjalani proses pengobatan atau perawatan. Menurut Angga (2015) dokumen rekam medis pasien berisi beberapa data seperti identitas pribadi, riwayat pengobatan, serta diagnosis riwayat penyakit yang diderita pasien sehingga rekam medis bersifat sangat rahasia dan hanya pihak tertentu dari rumah sakit yang boleh mengetahui informasi yang ada pada rekam medis pasien. Tidak hanya sebagai status kesehatan seorang pasien, rekam medis juga sebagai bukti tercatat mengenai diagnosis penyakit yang diderita pasien serta riwayat pelayanan medis yang pernah didapat pasien selama menjalani pengobatan atau perawatan medis.

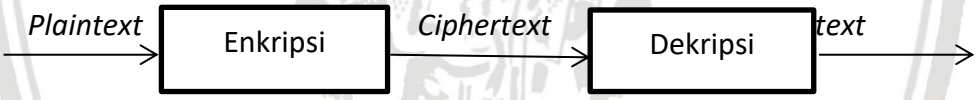
2.3.1 Kegunaan Rekam medis

Kegunaan rekam medis dapat ditinjau dari tujuh aspek berbeda, antara lain aspek administrasi, aspek medis, aspek hukum, aspek keuangan, aspek penelitian, aspek pendidikan, dan aspek dokumentasi. Menurut Ditjen Yankes (1993) : 10 kegunaan rekam medis pasien ditinjau dari aspek administrasi merupakan dokumen yang berisi tindakan dan tanggung jawab tenaga medis kepada pasien untuk mencapai tujuan pelayanan kesehatan. Perencanaan penangan atau perawatan kesehatan pasien tertulis pada rekam medis menjadikan rekam medis memiliki kegunaan pada aspek medis. Rekam medis juga memiliki kegunaan dari

aspek hukum karena bernilai hukum yang dapat digunakan sebagai barang bukti untuk menegakkan hukum. Dari aspek keuangan, berkas rekam medis dapat digunakan sebagai aspek keuangan karena memiliki informasi keuangan pasien. Berkas rekam medis juga dapat digunakan sebagai aspek penelitian dan nilai pendidikan yang menyangkut dengan ilmu pengetahuan dibidang kesehatan dengan menggunakan informasi kronologis pengobatan kepada pasien sebagai referensi pengetahuan. Ditinjau dari aspek dokumentasi, berkas rekam medis berguna sebagai laporan pertanggungjawaban dan laporan rumah sakit karena rekam medis memiliki informasi tentang riwayat penanganan kesehatan kepada pasien.

2.4 Kriptografi

Menurut Wiejaya (2016) Kritografi memiliki definisi sebagai ilmu untuk mengamankan informasi dengan cara enkripsi dan dekripsi seperti data atau pesan. Enkripsi merupakan cara untuk mengubah *plaintext* (pesan asli) menjadi *ciphertext* (pesan yang terkunci) sedangkan dekripsi adalah kebalikan proses enkripsi. Dalam melakukan proses enkripsi dan dekripsi melibatkan dua hal yaitu algoritme yang dipakai dalam enkripsi maupun dekripsi yang biasa disebut *ciphertext* dan *key* sebagai kunci untuk membuka atau mentransformasikan algoritme menjadi *plaintext* sehingga pesan yang sudah dienkripsi dapat dibaca kembali dan memberikan sebuah informasi dari pengirim kepada penerima pesan melalui proses dekripsi.



Gambar 2.1 Konsep Kriptografi
Sumber : (Hidayat & Afrianto, 2017)

Algoritme pada kriptografi atau *cipher* sendiri merupakan sebuah fungsi matematika yang digunakan untuk menyandikan *plaintext* sehingga menghasilkan keluaran berupa pesan acak yang tidak bisa dibaca apabila tidak didekripsi, sedangkan kunci atau *key* yang digunakan untuk proses dekripsi berupa deretan bit (Wiejaya, 2016). Kriptografi memakai algoritme tertentu untuk melakukan proses enkripsi dan dekripsi dengan tujuan agar pesan hanya bisa dibaca oleh pengirim dan penerima pesan, namun banyaknya pelaku kejahatan *cyber* membuat algoritme kriptografi tidak bisa dipakai terus menerus dikarenakan algoritme tersebut sudah diketahui celahnya yang membuat pesan atau data dapat dibaca oleh pihak tidak bertanggung jawab.

2.4.1 Tujuan Kriptografi

Kriptografi sebagai ilmu yang mempelajari teknik pengaman informasi menggunakan fungsi matematika mempunyai tujuan yaitu untuk mengamankan data atau informasi dari pihak-pihak yang tidak memiliki izin atas data atau

informasi tersebut. Tujuan mendasar kriptografi adalah melaksanakan tiga aspek utama dalam keamanan informasi yaitu *Confidentiality, Integrity* dan *Availability* (Candra, et al., 2014). *Confidentiality* atau kerahasiaan merupakan aspek utama dalam keamanan informasi dimana kerahasiaan sebuah informasi atau pesan harus terjaga dari pihak-pihak yang tidak memiliki i*z*in atau otoritas pada informasi atau pesan yang tersandi. *Integrity* atau keutuhan merupakan aspek dimana sebuah informasi atau pesan harus terjamin keutuhannya, tidak pernah dirubah atau dimodifikasi pada saat proses pengiriman menuju pihak yang memiliki wewenang terhadap informasi atau pesan tersandi. *Availability* atau ketersediaan merupakan aspek dimana informasi atau pesan harus tetap tersedia bila diperlukan sewaktu-waktu oleh pihak yang memiliki izin atau otoritas terhadap informasi atau pesan tersandi tersebut.

2.4.2 Jenis Kriptografi

Algoritme yang digunakan untuk keperluan enkripsi dan dekripsi pesan atau informasi pada kriptografi pasti memiliki kelebihan dan kekurangannya sendiri. Kriptografi sendiri terbagi menjadi dua, yaitu kriptografi dengan kunci *simetris* dan kriptografi dengan kunci *asimetris*. Perbedaan mendasar dari kriptografi simetris dan asimetris terdapat pada kunci yang digunakan pada saat proses enkripsi dan dekripsi. Pada kriptografi kunci simetris, pengirim dan penerima pesan menggunakan sebuah kunci yang sama untuk melakukan proses enkripsi dan dekripsi. Kunci ini bersifat rahasia dan tidak boleh diketahui pihak lain selain pengirim dan penerima pesan atau informasi (Sugiarto, 2017).

Penggunaan *private key* atau kriptografi kunci simetris memiliki keunggulan di aspek kecepatan karena pada proses enkripsi dan dekripsi menggunakan kunci yang sama antara pengirim dan penerima pesan maka proses enkripsi dan dekripsi akan berjalan lebih cepat sehingga cocok digunakan untuk melakukan proses enkripsi data yang sangat besar. Setelah ditemukan algoritme menggunakan *private key cryptography,* studi lebih lanjut dilakukan untuk memenuhi kebutuhan metode kriptografi yang lebih aman dari sebelumnya kemudian ditemukanlah kriptografi asimetris, dimana kunci yang digunakan pada saat proses enkripsi dan dekripsi sama sekali berbeda. Pada kriptografi asimetris pengirim dan penerima pesan masing-masing memili dua buah kunci, kunci *public* dan kunci *private.* Proses enkripsi dan dekripsi pada kriptografi asimetris melibatkan dua buah kunci yang dimiliki pengirim dan penerima pesan. Kunci *public* digunakan pada saat proses enkripsi dan bersifat tidak rahasia yang dapat diketahui semua orang, sedangkan kunci *private* digunakan pada saat proses dekripsi dan bersifat rahasia yang hanya diketahui oleh pihak yang bertukar informasi atau pesan. Meskipun kunci *public* bisa diketahui oleh siapapun, akan sangat sulit untuk menemukan kunci *private* yang digunakan seseorang melalui kunci *public* yang sudah diketahui. Kelebihan dari kriptografi asimetris ini akan menghasilkan keamanan yang lebih tinggi karena menggunakan kunci *public* dan *private* yang lebih rumit untuk dipecahkan oleh pihak lain yang tidak bertanggung jawab contoh dari algoritme kriptografi asimetris adalah RSA dan ECC (Candra, et al., 2014).

2.5 Algoritme Grain v1

Perancangan, pengimplementasian dan evaluasi keamanan pada *stream cipehr* terus dikembangkan untuk mengatasi masalah keamanan. Salah satu algoritme *stream cipher* yang dapat digunakan karna sulit ditembus dan memiliki tingkat keamanan tinggi adalah algoritme Grain v1 *cipher.* Sebagai Institusi yang mengatur keamanan data, *National Institute of Standard Technology* (NIST) memilih algoritme Grain v1 *cipher* sebagai algoritme keamanan data pada sisi *hardware* dan menggantikan algoritme sebelumnya yaitu A5/1. Algoritme Grain dirancang untuk memiliki kecepatan tinggi dengan konsumsi sumber daya yang rendah pada *hardware*. Saat ini algoritme Grain memiliki tiga versi yaitu Grain *cipher* V0, V1, dan 128. Pada algoritme Grain v1 menggunakan *key* 80 bit, *Initialization Vector* (IV) 64 bit, dan 160 bit *cycle*. Sedangkan Grain v1 128 menggunakan *key* 128 bit, IV 96 bit, serta 256 *cycle,* namun secara arsitektur dan cara kerja, baik Grain v1 dan 128 memiliki kesamaan (Hell, et al., 2006). Pada arsitektur algoritme Grain v1 menggunakan tiga blok utama yaitu, *Non-Linier Feedback Shift register* (NFSR), *Linier Feedback Shift register* (LFSR), dan keluaran. Secara umum semua versi dari Grain *cipher* memiliki cara kerja yang sama namun terdapat perbedaan pada struktur NFSR, *h(x)* dan *zi*, dimana jumlah *zi* yang dikembalikan pada NFSR dan LFSR setiap versi memiliki jumlah yang berbeda (Nurrohmah, et al., 2017).

2.5.1 Arsitektur Algoritme Grain v1

Beberapa algoritme kriptografi seperti Grain v1*, Trivium,* dan *Mickey* menggunakan LFSR sebai blok utama. LFSR sendiri merupakan sebuah *shift register* yang menggunakan operasi *exclusive-or* (XOR) pada bit keluaran untuk dijadikan sebagai masukan pada *state* berikutnya. Pada arsitektur Grain v1*,* LFSR digunakan untuk menjamin periode minum untuk *keystream* dan menjaga keseimbangan keluaran. *Initialization Vector* (IV) dari LFSR biasa disebut dengan istilah *seed* dimana nilai yang dihasilkan register ini akan ditentukan oleh *state* sekarang atau *state* sebelumnya, oleh karena itu keluaran dari register memiliki keterbatasan dan pasti akan membentuk sebuah siklus yang berulang. Siklus yang berulang ini akan membentuk urutan bit yang sangat panjang dan tampak acak karena LFSR memiliki fungsi umpan balik yang baik (Herlambang, 2010). Pada NFSR terjadi proses *masked,* dimana masukan dari NFSR ditutupi dengan keluaran dari LFSR sehingga keluaran dari NFSR menjadi seimbang. NFSR memproses masukan bersama dengan *filter* nonlinier yang menyebabkan cipher menjadi tidak linier dan tidak mengeluarkan keluaran yang memiliki siklus berulang.

Pada arsitektur Grain v1*,* LFSR dan NFSR masing-masing memiliki ukuran sebesar 80 bit. Keduanya memiliki *key* 80 bit, IV sebesar 64 bit. LFSR dinotasikan dengan $S_i, S_{i+1}, S_{i+2}, \dots, S_{i+78}, S_{i+79}$ dan hasil *feedback* dari LFSR dinotasikan dengan $f(x)$ sedangkan NFSR dinotasikan dengan $b_i, b_{i+1}, b_{i+2}, \dots, b_{i+78}, b_{i+79}$ dan hasil *feedback* NFSR dinotasikan dengan $g(x)$ yang merupakan polinomial dari 80 bit (Bokhari, 2014). Persamaan notasi LFSR dituliskan dengan persamaan 1 berikut :

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80} \quad (1)$$

Untuk menghilangkan kemungkinan keluarnya nilai ambigu, maka dilakukan fungsi update pada LFSR dengan notasi pada persamaan 2 berikut :

$$S_{i+80} = S_{i+62} + S_{i+38} + S_{i+23} + S_{i+13} + S_i \quad (2)$$

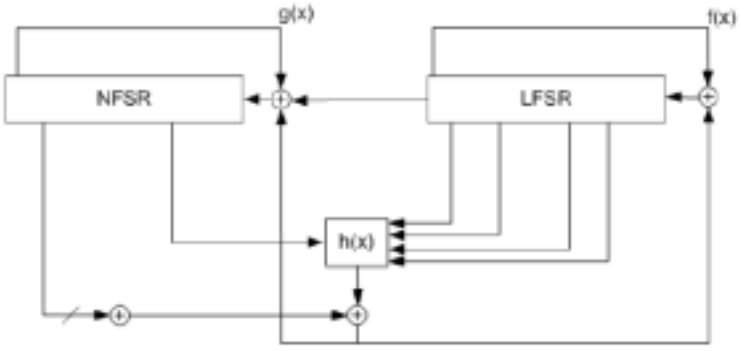
Persamaan notasi NFSR dituliskan dengan persamaan 3 berikut :

$$\begin{aligned} g(x) = 1 &+ x^{17} + x^{20} + x^{28} + x^{35} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + \\ &x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + \\ &x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + \\ &x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59} \end{aligned} \quad (3)$$

Sekali lagi untuk menghilangkan kemungkinan keluarnya nilai ambigu, maka dilakukan fungsi update pada NFSR dengan notasi pada persamaan 4 berikut :

$$\begin{aligned} b_{i+80} = s_i &+ b_{i+63} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + \\ &b_{i+21} + b_{i+15} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + \\ &b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + \\ &b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + \\ &b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + \\ &b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21} \end{aligned} \quad (4)$$

Berikut adalah arsitektur algoritme Grain *cipher* yang digambarkan pada gambar 2.2



Gambar 2.2 Arsitektur Grain **v1**
Sumber : (Hell, et al., 2006)

Gambar 2.2 menjelaskan tentang cara kerja Grain v1 *cipher* secara umum dimana isi dari kedua register, yaitu LFSR dan NFSR merepresentasikan *state* dari *cipher.* Fungsi $h(x)$ merupakan fungsi *Boolean* yang masukannya berasal dari lima variabel yang diambil dari LFSR dan NFSR. Fungsi $h(x)$ bertugas sebagai penyeimbang dan fungsi ini memiliki derajat aljabar 3 dengan kemungkinan nonlinieritas tertinggi adalah 12. Fungsi $h(x)$ dinotasikan dengan persamaan 5 berikut :

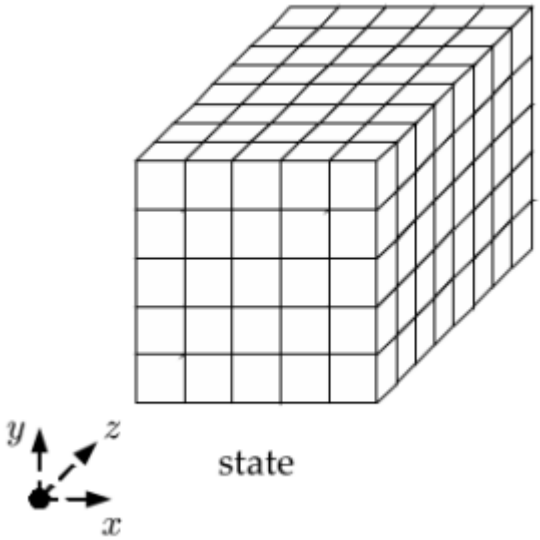
$$\begin{aligned} h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + \\ x_2x_3x_4 \end{aligned} \quad (5)$$

dimana variabel x_0, x_1, x_2, x_3, dan x_4 berhubungan dengan masing - masing posisi *tap* S_{i+3}, S_{i+25}, S_{i+46}, S_{i+64}, dan b_{i+63}. Kemudian keluaran dari fungsi $h(x)$ akan di XOR-kan dengan keluaran NFSR sehingga menghasilkan *keystream* (Hell, et al., 2006).

2.6 *Secure Hash Algorithm-3* (SHA-3)

Secure Hash Algorithm-3 atau yang biasa disingkat dengan SHA-3 merupakan algoritme pemenang pada lomba yang diadakan NIST untuk menggantikan algoritme MD5, SHA-0, dan SHA-1 yang sudah tidak dapat digunakan karena telah ditemukan celah keamanannya. Algoritme SHA-3 juga dikenal dengan nama algoritme Keccak yang dirancang oleh Guido Bertoni, Joan Daemen, Michael Peeters, dan Gilles van Assche. Algoritme SHA-3 menjadi pemenang lomba dengan alasan tingkat keamanan dan kerumitan algoritme yang tidak terlalu tinggi sehingga dapat diaplikasikan pada sisi *hardware* maupun *software* (Fakhrusy, 2016). Pada algoritme SHA-3 memungkinkan keluaran dan blok perhitungan ditentukan sendiri sesuai kebutuhan oleh pengguna, namun biasanya SHA-3 menggunakan sebuah *draft* yang dikeluarkan oleh NIST sebagai standar baru fungsi *hash* pada tahun 2014 yang bernama FIPS 202 (Romine, 2015).

Karena bertujuan untuk menggantikan algoritme sebelumnya yaitu SHA-0 dan SHA-1, algoritme SHA-3 memiliki kelebihan dimana keluaran yang dihasilkan algoritme ini dapat memiliki ukuran yang beragam mulai dari 244, 256, 384, dan 512 bit sehingga algoritme ini dapat menahan serangan *brute force attack* lebih baik dari algoritme terdahulunya (Kurniawan, et al., 2017).



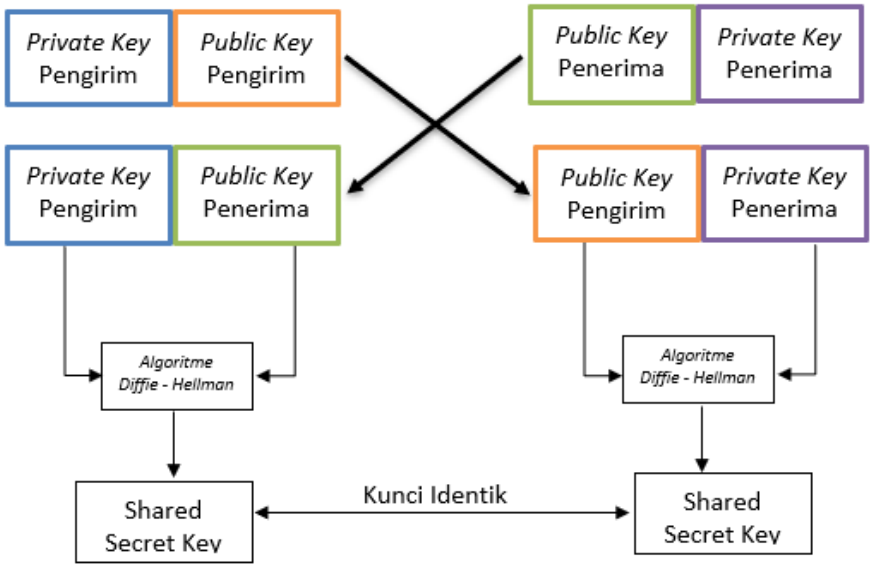
Gambar 2.3 *State* algoritme SHA-3
Sumber : (Fakhrusy, 2016)

Gambar 2.3 menjelaskan tentang *state* dari algoritme SHA-3 yang digambarkan dengan sebuah *array* tiga dimensi dimana pada dua sumbunya bernilai *array* 5x5

dan salah satu sumbu yang lain bernilai bervariasi. *Array* pada algoritme ini dapat dinotasikan sebagai b dan w dengan nilai $b = 25w$. Nilai notasi w pada *array* adalah b € 1, 2, 4, 8, 16, 32, dan 64 dengan nilai w = 64 sebagai nilai *default* yang biasa digunakan. Pada gambar 2.3 setiap blok pada *state* berisi nilai bit. Jika mengacu pada standar operasi nilai w = 64 bit, maka proses perhitungan dilakukan pada setiap nilai w-bit.

2.7 *Diffie – Hellman Key Exchange* (DHKE)

Diffie – Hellman Key Exchange (DHKE) merupakan sebuah metode untuk memproduksi dan bertukar kunci rahasia antara dua pihak yang akan melakukan pertukaran file rahasia. Pada saat proses pertukaran, jalur yang digunakan tidak dapat diinterupsi oleh pihak ketiga sebagai penyerang yang ingin mengetahui kunci rahasia meskipun jalur yang digunakan bukan merupakan jalur yang tidak aman. DHKE memiliki sebuah fitur dimana dua orang yang melakukan pertukaran *key* tidak saling mengirim informasi pada saat pembuatan *key,* tetapi memproduksi *key* secara bersamaan yang membuat DHKE aman dari interupsi pihak ketiga atau serangan yang biasa disebut *Man In The Middle* (MITM). Penggunaan DHKE dapat diandalkan karena pengguna dapat menggunakan teknik ini untuk mengenkripsi dengan *public key* dari penerima dan kemudian mulai mengenkripsi data selanjutnya dengan *private key* pengirim. Meskipun data atau pesan yang dikirim telah direkam ataupun dianalisis oleh MITM, tidak ada cara untuk mencari tahu *private key* dari pengirim dan penerima pesan (Durairajan & Saravanan, 2014).



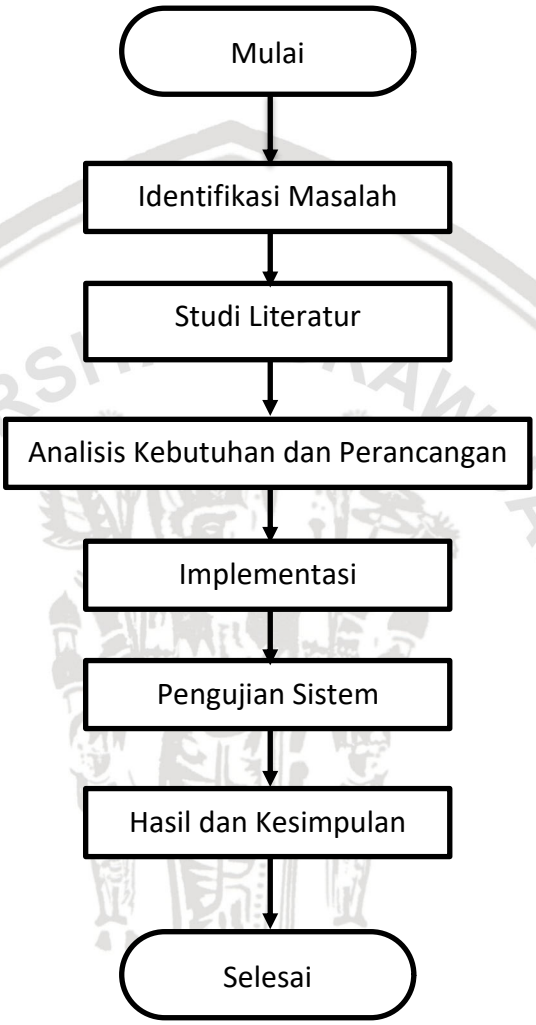
Gambar 2.4 Mekanisme pertukaran kunci algoritme *Diffie – Helman*
Sumber : (Durairajan & Saravanan, 2014)

Pada gambar 2.4 menjelaskan mekanisme pertukaran kunci algoritme DHKE dimana pada saat pembentukan *shared key* melibatkan *private key* dan *public key* pengirim dan penerima pesan. *Public key* pengirim maupun penerima pesan digunakan untuk membangkitkan *shared key* dengan kalkulasi dengan *private key*

yang dimiliki masing-masing penerima dan pengirim pesan. Pembentukan *shared key* algoritme DHKE melibatkan dua variabel umum yaitu α dan q dimana q adalah sebuah bilangan prima yang dipilih oleh penerima dan pengirim pesan dan α adalah akar pangkat dari q yang nilainya kurang dari q.

BAB 3 METODOLOGI PENELITIAN

Penjelasan mengenai cara atau metode yang ditempuh dalam penelitian ini yang diantaranya adalah identifikasi masalah, studi literatur, analisis dan perancangan sistem, implementasi, pengujian serta kesimpulan dari penelitian akan dibahas pada bab ini.



Gambar 3.1 Diagram Alir Tahapan Metodologi Penelitian

3.1 Identifikasi Masalah

Sistem Informasi pada beberapa rumah sakit sudah diterapkan untuk meningkatkan kinerja, mempercepat waktu pekerjaan petugas rumah sakit, serta mengurangi tingkat kekeliruan data. Namun sistem informasi pada rumah sakit saat ini masih bersifat lokal atau hanya dalam lingkup satu rumah sakit itu saja. Penelitian ini akan membahas pembuatan aplikasi rekam medis yang dapat digunakan beberapa rumah sakit untuk melakukan pertukaran dokumen rekam medis pasien. Kerahasiaan rekam medis pasien harus tetap terjaga pada saat

proses pengiriman dari rumah sakit satu ke rumah sakit yang lain. Oleh karena itu pada aplikasi yang akan dibangun diterapkan keamanan informasi berupa kriptografi dengan menggunakan algoritme Grain v1 sebagai metode enkripsi dan dekripsi data serta menggunakan algoritme SHA-3 sebagai metode *hashing* dan pengaman dokumen rekam medis pasien.

3.2 Studi Literatur

Dalam melakukan penelitian dilakukan kajian literatur untuk memahami dan mencari solusi permasalahan. Kajian literatur yang dilakukan berupa pencarian referensi yang diperoleh dari jurnal ilmiah, buku, dan pendapat pembimbing skripsi serta diskusi dengan teman-teman mahasiswa lain. Beberapa hal yang perlu dilakukan saat pengkajian literatur ini adalah :

1. Memahami konsep rekam medis pasien rumah sakit
2. Memahami konsep keamanan informasi dan kriptografi
3. Memahami dan mempelajari tentang cara kerja algoritme Grain v1 dan algoritme SHA-3
4. Memahami cara melakukan uji kinerja pada sistem yang dibangun

3.3 Analisis Kebutuhan dan Perancangan

Pada tahap ini yang dilakukan adalah mendefinisikan dan menentukan kebutuhan yang diperlukan dalam implementasi sistem seperti perancangan algoritme yang digunakan, perancangan sistem, dan perancangan pengujian. Proses pencarian dan analisis kebutuhan dan perancangan dapat dilakukan dengan konsultasi dengan tenaga ahli dibidang rekam medis dan konsultasi dengan dosen pembimbing skripsi untuk menentukan analisis kebutuhan pada keamanan sistem.

Perancangan algoritme Grain v1 dan SHA-3 yang akan diterapkan pada sistem rekam medis, membutuhkan proses perhitungan manual untuk menghasilkan *keystream* dan melakukan *test vector* manual. Pada perancangan sistem, dibahas tentang penjelasan langkah-langkah operasi serta prosedur yang perlu dilakukan dalam pengiriman dan pengaman data rekam medis pasien. Proses perancangan pengujian adalah menentukan jenis pengujian keamanan pada sistem dan algoritme pada saat proses pengiriman data rekam medis.

3.4 Implementasi

Pada tahap ini, dilakukan implementasi dari analisis kebutuhan sistem yang sudah ditentukan pada bagian sebelumnya. Implementasi meliputi pembuatan program aplikasi rekam medis dengan mengimplementasikan keamanan informasi menggunakan algoritme Grain v1 dan algoritme SHA-3 serta metode pertukaran kunci enkripsi dan enkripsi dengan menggunakan algoritme DHKE. Implementasi sistem meliputi penjelasan mengenai kebutuhan spesifikasi perangkat keras dan perangkat lunak yang akan digunakan untuk menjalankan

sistem. Pembuatan dan penerapan proses enkripsi dan dekripsi data dilakukan pada tahap implementasi sistem dengan berorientasi pada tahap analisis kebutuhan dan perancangan yang sudah dilakukan sebelumnya.

3.5 Pengujian

Pada tahap ini dilakukan pengujian sistem yang bertujuan untuk mengetahui apakah hasil implementasi sistem sudah sesuai dengan analisis kebutuhan dan perancangan keamanan sistem pada tahap sebelumnya. Skema pengujian yang akan dilakukan adalah menguji performa dari algoritme Grain v1 sebagai algoritme enkripsi untuk menjaga kerahasiaan data pada saat proses pengiriman dan algoritme SHA-3 sebagai algoritme *hashing* yang bertujuan untuk menjaga keutuhan atau integritas data. Parameter yang digunakan untuk melakukan pengujian adalah parameter waktu yang diperlukan untuk melakukan proses enkripsi dan dekripsi.

Pengujian kerahasiaan data rekam medis dilakukan untuk memastikan bahwa data yang dikirim tidak dapat dibaca oleh pihak yang tidak bertanggung jawab atau *hacker* yang ingin mendapatkan data rekam medis pasien. Pengujian integritas atau keutuhan data bertujuan untuk menjaga data dari penyerang yang ingin mengubah atau memodifikasi isi serta informasi dari data yang dikirim.

3.6 Hasil dan Kesimpulan

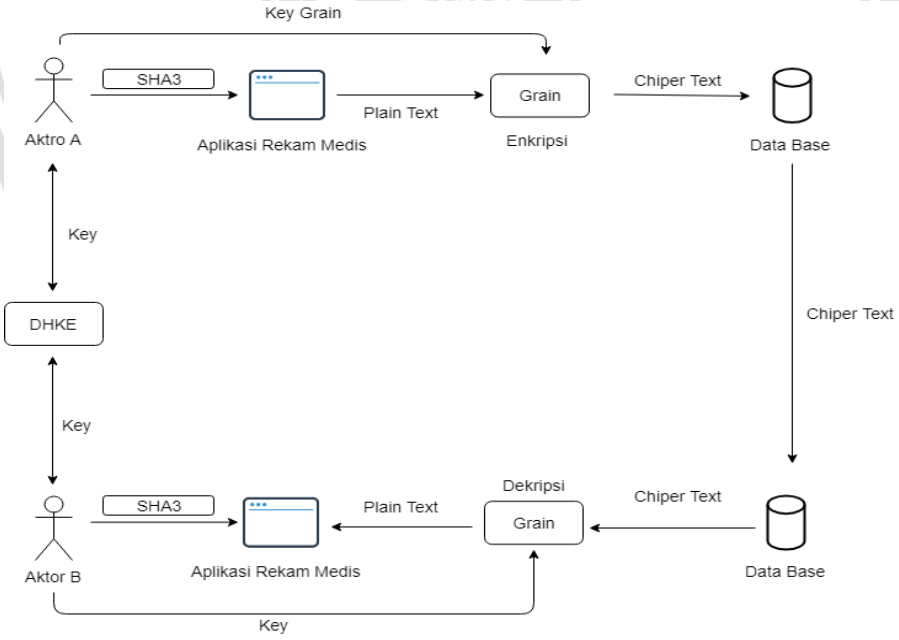
Pada tahap ini menjabarkan analisis dan hasil dari pengujian yang telah dilakukan sebelumnya kemudian menarik suatu kesimpulan sesuai dengan rumusan masalah yang ada, serta memberikan saran untuk penelitian selanjutnya dengan topik yang terkait.

BAB 4 ANALISIS DAN PERANCANGAN

Pada bab ini membahas tentang analisis kebutuhan sistem serta perancangan yang meliputi perancangan sistem, perancangan algoritme, dan perancangan pengujian terhadap sistem yang dibangun. Proses perancangan ini adalah sebagai acuan untuk melakukan implementasi sistem yang akan dibangun agar sistem dapat menjalankan fungsi sesuai analisis kebutuhan.

4.1 Gambaran Umum Sistem

Sistem yang akan dibangun adalah aplikasi berbasis *desktop* yang terhubung pada basis data yang menyimpan data serta informasi tiruan atau rekayasa dari data rekam medis pasien sebuah rumah sakit. Sistem yang dibangun dapat mengamankan data pada basis data rumah sakit dengan cara melakukan enkripsi data yang dimasukkan ataupun diperbarui pada basis data. Pada saat proses pengiriman, data yang dikirim berupa data *ciphertext* hasil dari enkripsi data menggunakan algoritme Grain v1, sedangkan untuk menjaga keutuhan data dari pihak pengirim sampai ke pihak penerima, sistem menggunakan algoritme SHA-3. Algoritme Grain v1 sendiri membutuhkan dua buah masukan yaitu *key* dan IV untuk menghasil *keystream* yang digunakan untuk proses enkripsi dan dekripsi data. *Key* dan IV ini harus dimiliki kedua pihak yang melakukan pertukaran data rekam medis supaya data dapat dienkripsi pada pihak pengirim dan dapat didekripsi pada pihak penerima. Sistem yang dibangun juga memiliki fungsi pertukaran *key* dan IV dengan menggunakan algoritme DHKE.



Gambar 4.1 Gambaran Umum Sistem

Pada gambar 4.1 merupakan penjelasan gambaran umum sistem dimana pada sistem ini aktor akan melihat tampilan halaman *login* dan melakukan *login* dengan

menggunakan *username* dan *password.* Setelah aktor memasukkan *username* dan *password*, sistem akan mengubah *password* yang dimasukkan oleh aktor sebagai masukan untuk algoritme SHA-3. Setelah proses *hashing* dilakukan selanjutnya sistem akan memeriksa apakan nilai *hash* dari *password* sama dengan nilai *hash* yang ada pada basis data. Jika *username* dan nilai *hash password* ditemukan maka aktor akan diteruskan ke halaman utama dimana aktor dapat melihat, memasukkan, menghapus dan mencari data informasi rekam medis pasien. Sistem yang akan dibangun memiliki dua buah *key* untuk proses enkripsi dan dekripsi data rekam medis pasien. Pada proses yang dilakukan pada lingkup rumah sakit A seperti melihat, memasukkan, menghapus dan mencari data informasi rekam medis pasien, aktor A sebagai admin rekam medis atau dokter pada rumah sakit A menggunakan *key* 1 untuk mengamankan basis data pada rumah sakit A saja. Pada saat memasukkan data rekam medis pasien akan memerlukan proses enkripsi data menggunakan algoritme Grain v1 untuk kemudian dimasukkan kedalam basis data rumah sakit A, sedangkan untuk menampilkan data rekam medis pasien memerlukan proses dekripsi untuk mengubah *ciphertext* dari basis data menjadi *plaintext* supaya dapat dibaca.

Untuk proses pengiriman data rekam medis dari rumah sakit A ke rumah sakit B, aktor A akan menginisialisasi *key2* yang khusus dibuat untuk enkripsi dan dekripsi data rekam medis pasien pada saat melakukan proses pengiriman. Sebelum dikirim, data dari basis data akan didekripsi dengan *key1* kemudian dienkripsi kembali dengan menggunakan *key2. Key2* inilah yang akan ditukar antara aktor A dan aktor B dengan menggunakan algoritme DHKE. Setelah aktor B sudah memiliki *key2,* data rekam medis akan diproses untuk menciptakan *ciphertext* melalui proses enkripsi menggunakan algoritme Grain v1 yang bertujuan untuk menjaga kerahasiaan data pada saat proses pengiriman. Setelah data yang berupa *ciphertext* masuk ke basis data rumah sakit B, aktor B akan melakukan proses dekripsi dengan menggunakan *key2* pada algoritme Grain v1 untuk mengubah *ciphertext* menjadi *plaintext* supaya dapat dibaca kembali oleh aktor B.

4.2 Kebutuhan Sistem

Pada subbab kebutuhan sistem akan dijabarkan kebutuhan apa saja yang diperlukan untuk membangun sistem yang meliputi kebutuhan fungsional sistem, kebutuhan perangkat keras dan kebutuhan perangkat lunak untuk menjalankan sistem. Kebutuhan sistem harus terpenuhi agar sistem dapat menjalankan fungsi dengan baik.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan dari sistem yang harus ada dan dapat dilakukan sebagai tugas utama dari sistem yang akan dibangun. Beberapa kebutuhan fungsional dari sistem antara lain :

1. Sistem harus menghasilkan *keystream* yang akan di-XOR kan dengan *plaintext* sehingga dapat menciptakan *ciphertext.*

2. Sistem dapat melakukan proses enkripsi dan dekripsi dengan hasil *plaintext* yang sama pada saat data sebelum dan sesudah dikirim.
3. Sistem dapat melakukan pertukaran *key* antara kedua pihak yang melakukan pertukaran data rekam medis pasien.
4. Sistem dapat melakukan proses *hashing* untuk menjamin data tetap utuh dan tidak mengalami perubahan ketika proses pengiriman.

4.2.2 Kebutuhan Perangkat Keras

Sistem membutuhkan perangkat keras dengan spesifikasi sebagai berikut untuk mencapai kinerja optimal. Perangkat keras yang digunakan adalah sebagai berikut laptop dengan spesifikasi :

1. Model : ASUS A456U
2. Sistem Operasi : Windows 10 Pro 64-bit
3. Prosesor : Intel Core i5 – 7200U
4. *RAM* : 4 Gb
5. Jumlah : 1

4.2.3 Kebutuhan Perangkat Lunak

Perangkat lunak yang dibutuhkan untuk membangun sistem pada penelitian ini menggunakan perangkat lunak *open source.* Berikut adalah spesifikasi perangkat lunak yang digunakan untuk membangun sistem :

1. NetBeans IDE 8.0.1

   NetBeans digunakan untuk menulis dan membangun *source code* program algoritme Grain v1*,* SHA-3, dan DHKE.

2. XAMPP Control Panel v3.2.2

   XAMPP digunakan untuk membuat basis data yang menyimpan data hasil masukan dari sistem.

3. Wire Shark Versi 2.0.1

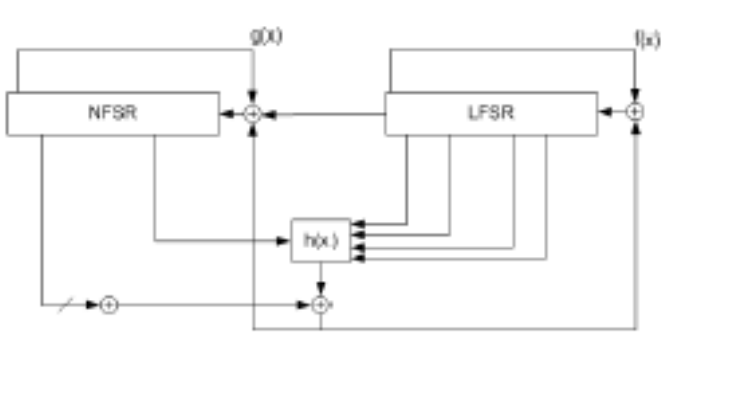
   Wire Shark merupakan *tools* yang digunakan untuk melakukan *capture* pada protokol TCP pada saat proses pengujian serangan *sniffing.*

4.3 Arsitektur Algoritme

Perancangan algoritme bertujuan untuk menjelaskan langkah-langkah bagaimana algoritme Grain v1 yang digunakan dalam sistem membentuk *keystream* dan dapat melakukan proses enkripsi dan dekripsi data atau informasi rekam medis pasien sehingga bisa memberikan keamanan informasi pada sistem. Pada subbab perancangan algoritme ini hanya akan membahas bagaimana algoritme Grain v1 untuk menghasilkan *keystream,* karena pada penelitian ini

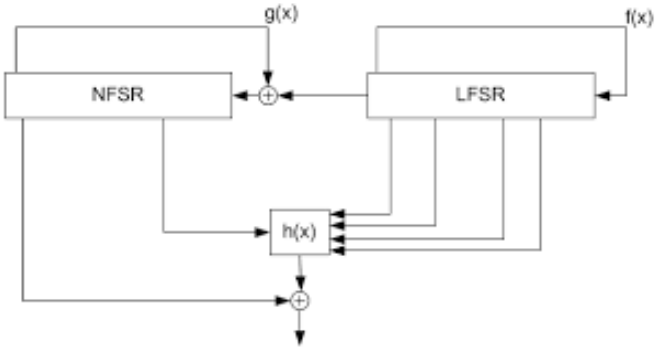
algoritme SHA-3 dan algoritme DHKE menggunakan *library* yang diunduh pada laman *https://github[dot]com/pannous/Diffie-Hellman.*

Algoritme Grain v1 memiliki langkah-langkah untuk membentuk *keystream* dimana langkah pertama adalah mengisi indeks *array* LFSR dan NFSR. LFSR merupakan sebuah *shift register* yang memiliki panjang 80 bit dan akan diisi oleh IV sebagai masukan. IV sendiri merupakan sebuah deretan angka biner yang ditentukan secara acak oleh *user,* namun untuk melakukan *test vector* nilai *default* dari IV adalah angka biner 1 sebanyak 64 bit dan angka biner 0 sebanyak 16 bit. NFSR juga merupakan sebuah *shift register* yang memproses masukan berupa deretan angka biner sebanyak 80 bit. NFSR memiliki masukan yang disebut dengan *key* yang juga dapat ditentukan secara acak oleh user, namun nilai *default* dari *key* untuk melakukan *test vector* adalah angka biner 0 sebanyak 80 bit. Setelah LFSR dan NFSR diisi oleh IV dan *key,* langkah selanjutnya adalah inisialisasi kunci yang diperoleh dari meng-XOR-kan nilai beberapa indeks dari LFSR dan NFSR sebanyak 160 kali *clock. Keystream* dari algoritme Grain v1 bisa diperoleh setelah melakukan proses inisialisasi kunci sebanyak 160 kali *clock. Clock* ke-161 dan seterusnya merupakan *keystream* yang dihasilkan oleh algoritme Grain v1.



Gambar 4.2 Inisialisasi Kunci Algoritme Grain v1
Sumber : (Hell, et al., 2006)

Gambar 4.2 menjelaskan skema proses inisialisasi kunci Grain v1 yang dilakukan sebanyak 160 kali *clock* sebelum memproduksi *keystream* (Hell, et al., 2006).



Gambar 4.3 Produksi *Keystream*
Sumber : (Hell, et al., 2006)

Pada gambar 4.3 merupakan skema produksi *keystream* algoritme Grain v1 dimana proses ini akan dijalankan setelah mengerjakan proses inisialisasi kunci sebanyak 160 kali *clock. Keystream* akan dihasilkan mulai dari *clock* ke-161 sampai sepanjang *plaintext* yang akan dienkripsi (Hell, et al., 2006). Algoritme Grain v1 akan bernilai valid apabila hasil dari *test vector* menghasilkan keluaran *keystream* yang sama dengan keluaran *keystream* pada *paper* yang dibuat oleh pencipta algoritme Grain v1. *Key* dan IV disesuaikan dengan *key* dan IV pada *paper* asli pencipta algoritme Grain v1 agar *keystream* yang dihasilkan sama. Perbandingan hasil *test vector* pada *paper* pencipta algoritme Grain v1 dan *test vector* pada penelitian ini adalah sebagai berikut:

Hasil *keystream* dari algoritme Grain v1 yang dilakukan oleh penciptaanya

```
Key:        00000000000000000000
IV:         0000000000000000
Keystream:  7b978cf36846e5f4ee0b
```

Gambar 4.4 *Test Vector*

Sumber : (Hell, et al., 2006)

Pada gambar 4.4 terlihat bahwa *Key* yang digunakan bernilai 0 sebanyak 80 bit dan IV bernilai 0 sebanyak 64 bit pertama kemudian bernilai 1 pada 16 bit sisanya. *Key* dan IV yang dimasukkan pada LFSR dan NFSR akan mengalami proses inisialisasi kunci sebanyak 160 kali *clock* dan pembentukan *keystream* setelah *clock* ke 161 sampai *clock* ke 240 sehingga menghasilkan *keystream* sebanyak 80 bit. Pada gambar 4.4 juga terlihat bahwa *keystream* yang terbentuk berupa angka hexadecimal sebanyak 20 digit dan bila diubah ke bentuk biner maka akan menghasilkan *keystream* 01111011100101111000110011110011011010000100 01101110010111110100111011000001011.

Hasil *keystream* dari algoritme Grain v1 yang digunakan pada penelitian ini dengan menggunakan nilai *Key* dan IV yang sama seperti pada *paper* penciptanya dan keluaran yang dikeluarkan program adalah sebagai berikut :

```
LFSR = 00000000000000000000000000000000000000000000000000000000000000001111111111111111
NFSR = 00000000000000000000000000000000000000000000000000000000000000000000000000000000
01111011100101111000110011110011011010000100011011100101111101001110111000001011
Keystream = BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 4.5 Hasil *Output* Program

Dari gambar 4.5 dapat dilihat bahwa keluaran program yang dibangun menghasilkan *keystream* 01111011100101111000110011110011011010000100 01101110010111110100111011000001011 dimana hasil *keystream* ini sama dengan hasil *keystream* dari *paper* pencipta algoritme Grain v1.

Tahapan proses algoritme Grain v1 untuk menghasilkan *key*stream adalah diawali dengan pengisian nilai LFSR dengan nilai IV dan pengisian NFSR dengan nilai *key.* Kemudian menjalakan proses mencari nilai *h(x)*, nilai *h(x)* yang dikeluarkan program adalah sebagai berikut :

```
hx = 00000000000000000001111111111100000010000001111111111000000111000010001111100000110
     10111101101000101111111101101101100110111000100100010100001100001000100001000110000
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 4.6 Hasil Perhitungan nilai *h(x)*

Pencarian nilai *h(x)* dilakukan setiap kali *clock* dengan meng-XOR-kan beberapa nilai dari indeks dari LFSR dan NFSR pada saat proses inisialisasi kunci. Setiap *clock* pada proses inisialisasi kunci akan menghasilkan 1 bit nilai *h(x)*, oleh karena itu banyaknya nilai *h(x)* akan sebanyak 160 bit seperti yang ditunjukkan gambar 4.6.

```
zi = 00000000000000000001111111111100000010000000000000000100110010000101011100111001 0
     11010111011100101100000101110101001100110110101011100111011001110111010010000111 1
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 4.7 Hasil perhitungan nilai *Zi*

Pada gambar 4.7 merupakan perhitungan manual dari nila *zi* dimana nilai *zi* didapat dari proses meng-XOR-kan beberapa nilai dari indeks NFSR dan di-XOR-kan lagi dengan masing-masing nilai dari *h(x).* Jumlah nilai *zi* sama dengan nila *h(x)* karena *zi* masih merupakan rangkaian proses inisialisai kunci.

```
fx = 00111111111111000000011111100000000001001110000000001101101000100110011110100110110
     01001110101101000011000110111100110001010011010001001110111111011000100101101101 0
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 4.8 Hasil Perhitungan nilai *f(x)*

Gambar 4.8 menunjukkan perhitungan nilai *f(x)* yang diperoleh dari meng-XOR-kan beberapa nilai indeks dari LFSR dan di-XOR-kan lagi dengan masing-masing nilai *zi.* Hasil dari perhitungan nilai *f(x)* nantinya akan dimasukkan kembali pada indeks LFSR ke-79 yang digunakan untuk proses selanjutnya yaitu proses produksi *keystream.*

```
gx = 00000000000000000001111111111100000011111111111110011101111001011001101111100101011
     01101101011110110000010110011001100100011000101000101111000001111001011100100110
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 4.9 Hasil Perhitungan nilai *g(x)*

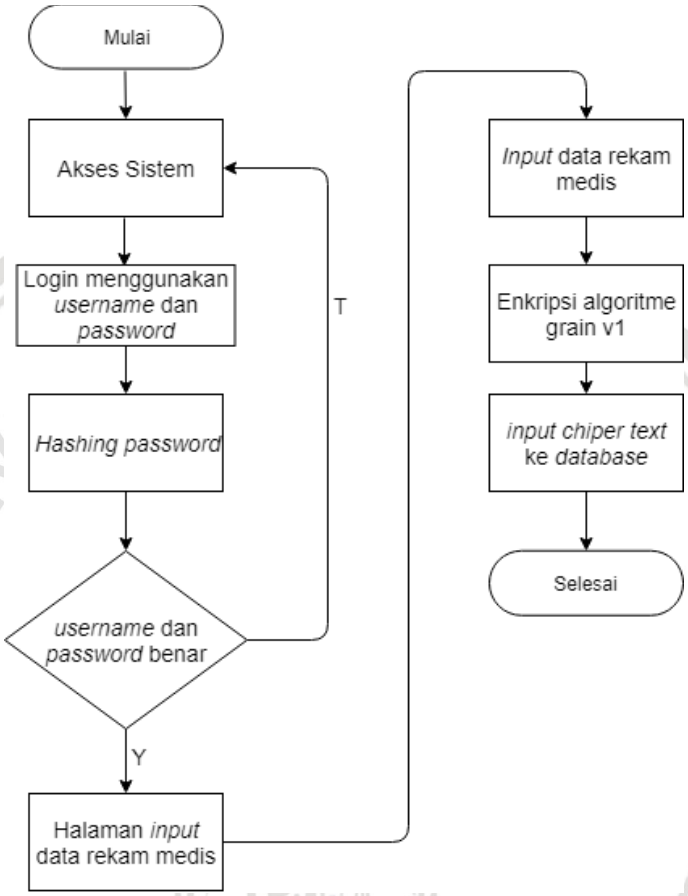
Gambar 4.9 menunjukkan perhitungan nilai *g(x)* yang diperoleh dari meng-XOR-kan beberapa nilai indeks dari NFSR dan di-XOR-kan lagi dengan masing-masing nilai *zi.* Hasil dari perhitungan nilai g(x) nantinya akan dimasukkan kembali pada indeks NFSR ke-79 yang digunakan untuk proses selanjutnya yaitu proses produksi *keystram.*

Setalah semua nilai *h(x), Zi, f(x),* dan *g(x)* ditemukan pada proses inisialisasi kunci dengan melakukan *clock* sebanyak 160 kali dan menggeser nilai register LFSR dan NFSR pada setiap *clock* selanjutnya adalah melakukan proses pembentukan *keystream* dimana prosesnya sama seperti proses inisialisasi kunci namun menggunakan nilai indeks LFSR dan NFSR yang sudah diproses pada tahap inisialisasi kunci. Proses pembentukan *keystream* dilakukan sebanyak 80 kali *clock*

untuk menghasilkan 80 bit *keystream* karena setiap *clock* hanya akan memproduksi 1 bit *keystream.*

4.4 Perancangan Sistem

Sistem yang akan dibangun memiliki alur kerja yang akan dijelaskan pada subbab perancangan sistem. Penjelasan alur kerja sistem akan digambarkan dengan *flowchart.*



Gambar 4.10 *Flowchart* Sistem Aplikasi Rekam Medis

Pada gambar 4.4 merupakan gambaran langkah-langkah yang dikerjakan oleh sistem dalam melakukan memasukkan dan pengamanan data rekam medis pasien. Proses yang dilakukan oleh pengirim ketika mengakses sistem dimaulai dari akses aplikasi yang dilakukan oleh pengirim data rekam medis. Pengirim data rekam medis akan diminta untuk melakukan l*ogin* dengan memasukkan *username* dan *password* sebagai langkah *autentifikasi*. *username* dan *password* yang dimasukkan pengirim akan diperiksa dengan mencocokkan *username* dan *password* yang ada pada basis data jika *username* dan *password* ditemukan atau benar, maka pengirim akan masuk ke halaman utama aplikasi, namun jika *username* dan *password* tidak ditemukan maka pengirim data akan dikembalikan ke halaman *login* untuk mengulangi proses *login.* Halaman utama sistem berisi beberapa *field* untuk mengisi data dan informasi rekam medis pasien. Pengirim dapat memasukkan data, dan jika sudah melakukan proses *memasukkan* data

yang sudah dimasukkan pengirim akan melalui proses enkripsi data oleh algoritme Grain v1 dan proses *hashing* yang dilakukan oleh algoritme SHA-3. Keluaran dari proses enkripsi dan *hashing* akan dikembalikan ke basis data.

4.5 Perancangan Pengujian

Perancangan pengujian bertujuan untuk menjelaskan metode pengujian terhadap sistem yang sudah dibangun untuk mengetahui performa dari algoritme Grain v1, SHA-3, dan DHKE. Metode pengujian sistem meliputi pengujian validasi *keystream,* pengujian hasil enkripsi dan dekripsi data rekam medis pasien, pengujian performa sistem sebelum dan sesudah dilengkapi algoritme pengaman data rekam medis pasien, pengujian terhadap serangan *sniffing,* dan pengujian *brute force attack*. penjelasan mengenai metode pengujian pada sistem adalah sebagai berikut :

1. Pengujian validasi *keystream* dilakukan dengan cara menguji *keystream* dengan *test vector* untuk mengetahui apakah *keystream* yang dihasilkan algoritme Grain v1 sama persis dengan *keystream* yang dihasilkan oleh arsitektur pembuatnya.
2. Pengujian validasi enkripsi dan dekripsi dilakukan pada sisi penerima dan pengirim data rekam medis. Pengujian ini dilakukan dengan cara mencocokkan *plaintext* yang dikirim dan diterima. *Plaintext* yang dikirim harus sama dengan *plaintext* yang diterima oleh penerima data.
3. Pengujian fungsional sistem adalah untuk mengetahui apakah ada kesalahan pada sistem dalam mengolah masukan yang diberikan. Cara pengujian fungsional adalah dengan menjalankan sistem dan memberikan masukan untuk diproses sehingga sistem dapat memberikan keluaran.
4. Pengujian kinerja sistem dilakukan dengan menggunakan parameter waktu untuk mengetahui waktu yang dibutuhkan oleh sistem untuk memproduksi melakukan proses enkripsi dan dekripsi dengan variasi IV dan *key* algoritme Grain sebanyak 1024 variasi dengan panjang *plaintext* sebanyak 80bit sesuai dengan *test vector*. Pengujian performa juga meliputi pengujian sistem sebelum dan sesudah dilengkapi dengan algoritme Grain sebagai metode enkripsi dan dekripsi data.
5. Pengujian serangan *sniffing* dengan menggunakan *tools* Wireshark untuk mengetahui dan memastikan apakah data yang dikirim ke basis data dan data yang ditukarkan antar rumah sakit sudah terenkripsi dengan cara melakukan proses *capture* pada protokol TCP. Proses *capture* pada protokol TCP dilakukan karena koneksi yang digunakan untuk bertukar data pada penelitian ini adalah protokol TCP.
6. Pengujian *brute force attack* atau serangan brutal pada sistem adalah dengan melakukan simulasi serangan dengan cara menggunakan *tools* yang bernama Rainbowcrack. Serangan dengan *tools* Rainbowcrack bertujuan untuk mengetahui hasil dekripsi *ciphertext* yang dilakukan

dengan mencoba satu persatu kemungkinan kata atau kalimat hasil dekripsi *ciphertext*.

BAB 5 IMPLEMENTASI

Bab implementasi berisi pembahasan tentang bagaimana cara menerapkan dan membuat sistem yang akan dibangun berdasarkan analisis kebutuhan dan perancangan yang sudah dibahas pada bab sebelumnya. Sistem yang dibangun berupa aplikasi yang dapat mengirim data rekam medis dengan keamanan informasi untuk menjamin kerahasiaan data pada saat proses pengiriman menggunakan algoritme Grain v1. Data yang dikirimkan juga dijamin integrasi atau keutuhannya dari pengirim sampai pada penerima dengan proses *hashing* dari algoritme SHA-3, serta pertukaran *key* antar aktor menggunakan algoritme DHKE. Sistem dibangun menggunakan Bahasa pemrograman java yang mengimplementasikan algoritme Grain v1, SHA-3, dan DHKE dimana untuk algoritme SHA-3 dan DHKE memakai *library.*

Tahap implementasi akan menjelaskan *coding* dan cara mengimplementasikan algoritme Grain v1, SHA-3, dan DHKE dengan menggunakan Bahasa pemrograman java dimana untuk algoritme SHA-3 menggunakan *library* yang diunduh dari https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk15on/1.59 sedangkan untuk algoritme DHKE didapat dari unduhan pada situs https://github.com/pannous/Diffie-Hellman. Algoritme Grain v1 digunakan dengan tujuan untuk mengamankan informasi data rekam medis pasien pada saat proses pengiriman dengan cara enkripsi dan dekripsi untuk menjamin kerahasiaan data. Data rekam medis yang dikirim harus sama pada saat sebelum dan sesudah dikirim dan untuk menjamin keutuhan data pada saat pengiriman, diperlukan proses *hashing* dengan menggunakan algoritme SHA-3. Algoritme DHKE digunakan untuk proses pertukaran *key* antar aktor sebagai kunci untuk proses enkripsi dan dekripsi algoritme Grain v1.

5.1 Algoritme Grain v1

Algoritme Grain v1 *cipher* merupakan algoritme keamanan data pada sisi perangkat keras yang menggantikan algoritme sebelumnya yaitu A5/1. Algoritme Grain v1 dirancang untuk memiliki kecepatan tinggi dengan konsumsi sumber daya yang rendah pada perangkat keras. Algoritme Grain v1 menggunakan *key* 80 bit, *Initialization Vector* (IV) 64 bit, dan 160 bit *cycle*.

Algoritme Grain v1 menggunakan dua buah *shift register* yaitu LFSR dan NFSR sebai blok utama. LFSR sendiri merupakan sebuah *shift register* yang menggunakan operasi *exclusive-or* (XOR) pada bit keluaran untuk dijadikan sebagai masukan pada *state* berikutnya. NFSR adalah *shift register* dimana masukan dari NFSR ditutupi dengan keluaran dari LFSR sehingga keluaran dari NFSR menjadi seimbang.

5.1.1 *Linear Feedback Shift register* (LFSR)

LFSR merupakan salah satu *shift register* yang digunakan pada algoritme Grain v1 yang memiliki panjang 80 bit. Pada keadaan *default* register ini akan diisi sebanyak 64 bit dari nilai IV *(Initial Value)* dan 16 bit sisanya diisi dengan angka 1. Pada suatu kondisi dimana nilai IV yang kurang dari 64 bit, maka LFSR akan diisi dari indeks ke-0 sampai sejumlah IV yang ada dan untuk mencapai 64 bit, maka dari indeks terakhir sampai indeks ke-63 akan diisi angka 0. Berikut adalah *source code* untuk mengisi nilai LFSR pada algoritme Grain v1:

Tabel 5.1 *Source Code* isiLFSR

```
Algoritme 1 : fungsi isiLFSR
1       public static void isiLFSR(String masukan) {
2           String a = hexToBinary(masukan);
3           if (a.length() <= 64) {
4               System.out.println("IV = " + a);
5               for (int i = 0; i < a.length(); i++) {
6                   lfsr[i] = Byte.parseByte(String.valueOf(a.charAt
                    (i)));
7               }
8               if (a.length() < 64) {
9                   for (int i = a.length(); i < 64; i++) {
10                      lfsr[i] = 0;
11                  }
12              }
13              for (int i = 64; i < 80; i++) {
14                  lfsr[i] = 1;
15              }
16          } else {
17              System.out.println("IV Terlalu Panjang, Proses Tidak
    dapat Dilanjutkan");
18      }
```

Pada tabel 5.1 merupakan *source code* dari proses pengisian indeks dari LFSR. *Method isiLFSR* berfungsi untuk menginisialisasi isi dari LFSR dengan mengisikan 80 bit nilai IV ke indeks LFSR. Jika isian IV berjumlah 64 bit, maka indeks ke-64 sampai indeks ke-79 akan diisi dengan nilai 1 sebagai isian default. Jika IV kurang dari 64 bit maka antara indeks terakhir isian IV sampai indeks ke-64 akan diisi dengan nilai 0. Jika IV lebih dari 64 bit maka proses tidak dapat dilanjutkan.

5.1.2 *Non-Linear Feedback Shift register* (NFSR)

NFSR juga merupakan register yang digunakan pada algoritme Grain v1 dengan panjang 80 bit. Nilai dari NFSR didapat dari *key* yang diisikan oleh aktor. Jika *key* yang dimasukkan kurang dari 80 bit, maka dari indeks terakhir isian *key* akan diisi dengan nilai *default* yaitu angka 0 sampai mencapai indeks ke-79.

Pada tabel 5.2 merupakan *source code* dari proses pengisian indeks dari NFSR. *Method isiNFSR* berfungsi untuk menginisialisasi isi dari NFSR dengan mengisikan 80 bit nilai *key* ke indeks NFSR. Jika masukan *key* berjumlah kurang dari 80 bit, maka antara indeks terakhir masukkan *key* sampai indeks ke-79 akan diisi dengan nilai 1. Jika IV lebih dari 64 bit maka proses tidak dapat dilanjutkan. Berikut adalah *source code* untuk mengisi nilai NFSR pada algoritme Grain v1:

Tabel 5.2 *Source Code* isiNFSR

```
Algoritme 2 : Fungsi isiNFSR
1   public static void isiNFSR(String masukan) {
2           String a = hexToBinary(masukan);
3           if (a.length() <= 80) {
4               System.out.println("Key = " + a);
5               for (int i = 0; i < a.length(); i++) {
6                   nfsr[i] = Byte.parseByte(String.valueOf(a.charAt
                    (i)));
7               }
8               if (a.length() < 80) {
9                   for (int i = a.length(); i < 80; i++) {
10                      nfsr[i] = 0;
11                  }
12              }
13          } else {
14              System.out.println("Key Terlalu Panjang, Proses Tidak
    dapat Dilanjutkan");
15          }
16          System.out.println(Arrays.toString(nfsr));
17      }
```

5.1.3 Inisialisasi Kunci

Sebelum algoritme Grain v1 memproduksi *keystream* terdapat sebuah proses yang disebut inisialisasi kunci dimana LFSR dan NFSR diproses dengan meng-XORkan beberapa isi dari indeksnya untuk selanjutnya digunakan untuk memproduksi *keystream.* Inisialisasi kunci ini dilakukan sebanyak 160 kali *cycle* untuk mengacak isi dari register LFSR dan NFSR. Banyak *cycle* dari proses inisialisasi kunci ini mengacu pada pencipta algoritme Grain. Berikut adalah *source code* untuk proses inisialisasi kunci pada algoritme Grain v1:

Tabel 5.3 *Source Code* Inisialisasi Kunci Grain **v1**

```
Algoritme 3 : Fungsi inisialisasi
1   public static void inisialisasi() {
2           for (int i = 0; i < 160; i++) {
3               hx = (lfsr[25] ^ nfsr[63] ^ (lfsr[3] & lfsr[64])
4                       ^ (lfsr[46] & lfsr[64])
5                       ^ (lfsr[64] & nfsr[63])
6                       ^ (lfsr[3] & lfsr[25] & lfsr[46])
7                       ^ (lfsr[3] & lfsr[46] & lfsr[64])
8                       ^ (lfsr[3] & lfsr[46] & nfsr[63])
9                       ^ (lfsr[25] & lfsr[46] & nfsr[63])
10                      ^ (lfsr[46] & lfsr[64] & nfsr[63]));
11              zi = (nfsr[1] ^ nfsr[2] ^ nfsr[4] ^ nfsr[10] ^ nfsr[31]
    ^ nfsr[43] ^ nfsr[56] ^ hx);
12
13              fx = (lfsr[62] ^ lfsr[51] ^ lfsr[38] ^ lfsr[23] ^
    lfsr[13] ^ lfsr[0]) ^ zi;
14
15              gx = lfsr[0] ^ nfsr[62] ^ nfsr[60]
16                      ^ nfsr[52] ^ nfsr[45] ^ nfsr[37]
17                      ^ nfsr[33] ^ nfsr[28] ^ nfsr[21]
18                      ^ nfsr[14] ^ nfsr[9] ^ nfsr[0]
19                      ^ (nfsr[63] & nfsr[60]) ^ (nfsr[37] &
    nfsr[33])
20                      ^ (nfsr[15] & nfsr[9]) ^ (nfsr[60] & nfsr[52]
    & nfsr[45])
21                      ^ (nfsr[33] & nfsr[28] & nfsr[21])
22                      ^ (nfsr[63] & nfsr[45] & nfsr[28] & nfsr[9])
```

```
23                      ^ (nfsr[60] & nfsr[52] & nfsr[37] & nfsr[33])
24                      ^ (nfsr[63] & nfsr[60] & nfsr[21] & nfsr[15])
25                      ^ (nfsr[63] & nfsr[60] & nfsr[52] & nfsr[45]
    & nfsr[37])
26                      ^ (nfsr[33] & nfsr[28] & nfsr[21] & nfsr[15]
    & nfsr[9])
27                      ^ (nfsr[52] & nfsr[45] & nfsr[37] & nfsr[33]
    & nfsr[28] & nfsr[21]) ^ zi;
28
29              List<Byte> data_list = Arrays.asList(lfsr);
30
31              Collections.rotate(data_list, -1);
32
33              data_list.set(79, (byte) fx);
34
35              List<Byte> data_list1 = Arrays.asList(nfsr);
36
37              Collections.rotate(data_list1, -1);
38              data_list1.set(79, (byte) gx);
39          }
40      }
```

Pada tabel 5.3 menjelaskan *source code* untuk proses inisialisasi kunci algoritme Grain v1. Pada saat melakukan proses inisialisasi kunci pada algoritme Grain v1, dilakukan 160 kali perulangan atau *clock* pada LFSR dan NFSR yang bertujuan untuk mengacak isi atau *state* dari LFSR serta NFSR sebelum algoritme memproduksi *keystream.* Inisialisasi kunci dilakukan dengan mencari nilai yang diperlukan seperti mencari nilai *filter function (hx), Z function (zi), fx,* dan *gx. Filter function* didapat dari proses meng-XOR-kan beberapa nilai LFSR dan satu nilai NFSR, sedangkan *Z function* didapat dari proses meng-XOR-kan beberapa nilai indeks dari NFSR dan hasil dari *filter function.*

Pada saat melakukan perulangan sebayak 160 kali *clock* untuk proses inisialisasi kunci, isi dari LFSR dan NFSR juga digeser sebanyak 160 kali. Setiap kali melakukan pergeseran, indeks ke-79 dari LFSR dan NFSR akan kosong, indeks kosong ini kemudian diisi dengan nilai *fx* untuk LFSR dan nilai *gx* untuk NFSR. Nilai *fx* diperoleh dari meng-XOR-kan beberapa nilai indeks LFSR dan nilai *Z function,* sedangkan nilai *gx* diperoleh dari meng-XOR-kan beberapa nilai indeks NFSR, satu indeks LFSR dan nilai dari *Z function.*

5.1.4 Produksi *Keystream*

Tahap produksi *keystream* dilakukan setelah melalui proses inisialisasi kunci. Pada tahap ini langkah-langkah yang ditempuh untuk memproduksi *keystream* hampir sama dengan proses inisialisasi kunci. Perbedaan antara proses inisialisasi kunci dan proses pembentukan *keystream* terdapat pada penggunaan nilai dari *filter function* dan *Z function.* Nilai dari *filter function* masih digunakan untuk mencari nilai *Z function,* namun hasil dari pencarian nilai *Z function* ini tidak akan digunakan untuk mencari nilai *fx* dan *gx* melainkan akan menjadi *keystream* dari algoritme Grain v1.

Pada tabel 5.4 merupakan *source code* dari proses pembentukan *keystream* algoritme Grain v1. Proses pembentukan *keystream* ini juga dilakukan dengan cara melakukan perulangan atau *clock* untuk menggeser isi indeks dari LFSR dan NFSR namun jumlah perulangan atau *clock* nya tidak sebanyak 160 kali namun

perulangan dilakukan sebanyak bit dari *plaintext* yang akan dienkripsi. Nilai dari *filter function* digunakan untuk menentukan nilai *Z function,* dari nilai *Z function* inilah *keystream* didapatkan. Jika pada proses inisialisasi kunci nilai *fx* dan *gx* di-XOR-kan dengan nilai *Z function* untuk mencari masukkan indeks ke-79 LFSR dan NFSR dalam satu kali *clock*, maka pada proses ini nilai *fx* dan *gx* tidak di-XOR-kan dengan nilai *Z function.* Berikut adalah *source code* untuk proses pembentukan *keystream* pada algoritme Grain v1:

Tabel 5.4 *Source Code* Pembentukan *Keystream*

Algoritme 4 : Fungsi *keystream*

```
1   public static void keystream() {
2           inisialisasi();
3           String keystreamm = "";
4           for (int i = 0; i < plain.length(); i++) {
5               hx = lfsr[25] ^ nfsr[63]
6                       ^ (lfsr[3] & lfsr[64])
7                       ^ (lfsr[46] & lfsr[64])
8                       ^ (lfsr[64] & nfsr[63])
9                       ^ (lfsr[3] & lfsr[25] & lfsr[46])
10                      ^ (lfsr[3] & lfsr[46] & lfsr[64])
11                      ^ (lfsr[3] & lfsr[46] & nfsr[63])
12                      ^ (lfsr[25] & lfsr[46] & nfsr[63])
13                      ^ (lfsr[46] & lfsr[64] & nfsr[63]);
14
15              zi = (nfsr[1] ^ nfsr[2] ^ nfsr[4] ^ nfsr[10] ^ nfsr[31]
    ^ nfsr[43] ^ nfsr[56]) ^ hx;
16
17              keystream[i] = (byte) zi;
18
19              fx = (lfsr[62] ^ lfsr[51] ^ lfsr[38] ^ lfsr[23] ^
    lfsr[13] ^ lfsr[0]);
20
21              gx = lfsr[0] ^ nfsr[62] ^ nfsr[60]
22                      ^ nfsr[52] ^ nfsr[45] ^ nfsr[37]
23                      ^ nfsr[33] ^ nfsr[28] ^ nfsr[21]
24                      ^ nfsr[14] ^ nfsr[9] ^ nfsr[0]
25                      ^  (nfsr[63]  &  nfsr[60])  ^  (nfsr[37]  &
    nfsr[33])
26                      ^ (nfsr[15] & nfsr[9]) ^ (nfsr[60] & nfsr[52]
    & nfsr[45])
27                      ^ (nfsr[33] & nfsr[28] & nfsr[21])
28                      ^ (nfsr[63] & nfsr[45] & nfsr[28] & nfsr[9])
29                      ^ (nfsr[60] & nfsr[52] & nfsr[37] & nfsr[33])
30                      ^ (nfsr[63] & nfsr[60] & nfsr[21] & nfsr[15])
31                      ^ (nfsr[63] & nfsr[60] & nfsr[52] & nfsr[45]
    & nfsr[37])
32                      ^ (nfsr[33] & nfsr[28] & nfsr[21] & nfsr[15]
    & nfsr[9])
33                      ^ (nfsr[52] & nfsr[45] & nfsr[37] & nfsr[33]
    & nfsr[28] & nfsr[21]);
34
35              List<Byte> data_list = Arrays.asList(lfsr);
36
37              Collections.rotate(data_list, -1);
38
39              data_list.set(79, (byte) fx);
40
41              List<Byte> data_list1 = Arrays.asList(nfsr);
42
43              Collections.rotate(data_list1, -1);
44
45              data_list1.set(79, (byte) gx);
```

```
46              keystreamm += keystream[i];
47          }
48          System.out.print(stringBinaryToHex(keystreamm));
49      }
```

5.1.5 Enkripsi

Setelah mendapatkan nilai dari *keystream,* tahap selanjutnya adalah proses enkripsi *plaintext* menjadi *ciphertext.* Proses enkripsi dilakukan dengan cara meng-XOR-kan nilai bit biner dari *plaintext* dengan nilai *keystream* yang sudah dibentuk. Berikut adalah *source code* untuk proses enkripsi pada algoritme Grain v1 :

Tabel 5.5 *Source Code* Enkripsi

```
Algoritme 5 : Fungsi encrypt
1     public static String encrypt(String input) {
2           String result = stringToBinary(input);
3           String result_string = "";
4           Byte[] result_array = new Byte[result.length()];
5           Byte[] result_xor_array = new Byte[result.length()];
6
7           for (int i = 0; i < result_array.length; i++) {
8               result_array[i]    =    Byte.parseByte(String.valueOf
    (result.charAt(i)));
9               result_xor_array[i]    =    (byte)    (keystream[i]    ^
    result_array[i]);
10              result_string += result_xor_array[i];
11          }
12          return stringBinaryToHex(result_string);
13      }
```

Pada tabel 5.5 merupakan *source code* dari proses enkripsi dimana *method encrypt* akan menerima masukan berupa *string plaintext* dan merubah *plaintext* menjadi bit biner. Setiap bit biner dari *plaintext* akan di-XOR-kan dengan setiap bit biner dari *keystream.* Keluaran dari *method encrypt* ini adalah *ciphertext.*

5.1.6 Dekripsi

Dekripsi merupakan proses kebalikan dari enkripsi, dimana *ciphertext* hasil enkripsi akan diubah menjadi *plaintext* kembali dengan cara meng-XOR-kan setiap bit biner *ciphertext* dengan setiap bit biner *keystream.* Berikut adalah *source code* untuk proses dekripsi pada algoritme Grain v1 :

Tabel 5.6 *Source Code* Dekripsi

```
Algoritme 6 : Fungsi decrypt
1     public static String decrypt(String cipher, String keystream) {
2           String a = hexToBinary(cipher);
3           Byte[] a_array = new Byte[a.length()];
4           String b = hexToBinary(keystream);
5           Byte[] b_array = new Byte[b.length()];
6           String plain_binary = "";
7           String plain = "";
8           Byte[] hasil1 = new Byte[a.length()];
9
10          for (int i = 0; i < a_array.length; i++) {
11              a_array[i]  =  Byte.parseByte(String.valueOf(a.charAt
    (i)));
12          }
13
14          for (int i = 0; i < b_array.length; i++) {
```

```
15              b_array[i]  =  Byte.parseByte(String.valueOf(b.charAt
    (i)));
16          }
17
18          for (int i = 0; i < a.length(); i++) {
19              hasil1[i] = (byte) (b_array[i] ^ a_array[i]);
20              plain_binary += hasil1[i];
21          }
22
23          for (int i = 0; i <= plain_binary.length() - 8; i += 8) {
24              int k = Integer.parseInt(plain_binary.substring(i, i
    + 8), 2);
25              plain += (char) k;
26          }
27
28          return plain;
29      }
```

Pada tabel 5.6 merupakan *source code* dari proses dekripsi, dimana proses ini merupakan kebalikan dari proses enkripsi yang mengembalikan *ciphertext* menjadi *plaintext* agar dapat dibaca kembali. *Method decrypt* pada tabel 5.6 memiliki masukan berupa dua tipe data *string* yaitu *string ciphertext* dan *string keystream.* Proses dekripsi dilakukan dengan cara meng-XOR-kan setiap bit biner dari *ciphertext* dengan setiap bit biner dari *keystream.* Keluaran dari *method decrypt* ini adalah berupa *plaintext.*

5.2 Algoritme SHA-3

Algoritme SHA-3 yang diimplementasikan pada sistem ini menggunakan *library* yang ditulis dengan menggunakan Bahasa pemrograman java. Algoritme SHA-3 dipanggil dengan masukan *password* aktor untuk melakukan proses *hashing* dengan tujuan menjamin data yang dikirim oleh aktor A sama persis dengan data yang diterima oleh aktor B. . Berikut adalah *source code* untuk proses pemanggilan fungsi *hashing* dengan menggunakan algoritme SHA-3 dengan *digest* 224 bit :

Tabel 5.7 *Source Code* Pemanggilan Algoritme SHA-3 224 bit

```
Algoritme 7 : Fungsi runSHA-3
1       public static String runSHA-3(String key) {
2           String input = key;
3           String SHA-3 = "";
4           SHA-3.DigestSHA-3 digestSHA-3 = new SHA-3.Digest224();
5           byte[] digest = digestSHA-3.digest(input.getBytes());
6
7           SHA-3 = Hex.toHexString(digest);
8           return SHA-3;
9       }
```

Pada tabel 5.7 menjelaskan tentang *source code* dari proses pemanggilan fungsi algoritme SHA-3 dengan keluaran 224 bit. *Method run*SHA-3 pada tabel 5.7 memilik masukan bertipe *string* yang merupakan *key* masukan algoritme SHA-3. Keluaran dari *method run*SHA-3 ini berupa *digest* dari proses *hashing.*

5.3 Algoritme *Diffie – Hellman Key Exchange* (DHKE)

Algoritme *Diffie – Hellman Key Exchange* (DHKE) merupakan salah satu algoritme yang digunakan untuk proses pertukaran kunci dengan menggunakan *private key* dan *public key* dari masing-masing aktor. DHKE memiliki sebuah fitur

dimana dua orang yang melakukan pertukaran *key* tidak saling mengirim informasi pada saat pembuatan *key,* tetapi memproduksi *key* secara bersamaan yang membuat DHKE aman dari interupsi pihak ketiga atau serangan yang biasa disebut *Man In The Middle* (MITM).

Pada sistem yang dibangun, algoritme DHKE digunakan untuk bertukar *key* yang digunakan untuk proses enkripsi data rekam medis pasien yang akan dikirim. Potongan program algoritme DHKE dibagi menjadi dua bagian, yaitu *source code* untuk *client* dan server. Sistem operasi Windows 10 akan bertindak sebagai server dan sistem operasi Windows XP akan bertindak sebagai *client.* Berikut ini adalah *source code* dari algoritme DHKE pada bagian *server* dan *client.*

Tabel 5.8 *Source Code* Algoritme DHKE

Algoritme 8 : Fungsi sserverDHKE	
1	public class sserverDHKE {
2	
3	public static void main(String[] args) throws IOException {
4	//server
5	ServerSocket serversocket = new ServerSocket(2929);
6	System.out.println("--Menunggu Koneksi.......");
7	Socket socket = serversocket.accept();
8	System.out.println("--Koneksi Client telah
9	disetujui.......");
10	
11	DataInputStream din = new DataInputStream(socket.getInput Stream());
12	DataOutputStream dout = new DataOutputStream(socket. getOutputStream());
13	BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
14	System.out.println("--DHKE");
15	BigInteger secretB;
16	BigInteger generatorValue, primeValue, publicA, publicB, shared;
17	String str;
18	primeValue = new BigInteger("330273671788511023796232738497605972489");
19	
20	generatorValue = new BigInteger("17");
21	
22	System.out.println("Menunggu Client mengirim Public *Key* Client..");
23	str = din.readUTF();
24	
25	publicA = new BigInteger(str);
26	System.out.println("Public *key* Client = " + publicA);
27	
28	System.out.print("Input Secret *Key* Server (integer) = ");
29	Scanner inSc = new Scanner(System.in);
30	str = inSc.next();
31	secretB = new BigInteger(str);
32	
33	publicB = generatorValue.modPow(secretB, primeValue);
34	System.out.println("Public *key* Server = " + publicB);
35	dout.writeUTF(publicB.toString());
36	dout.flush();
37	
38	shared = publicA.modPow(secretB, primeValue);
39	
40	System.out.println("Shared *key* = " + shared);
41	

```
42          }
43
44          //client
45          System.out.println("--Menghubungi Server.......");
46          Socket socket=new Socket("192.168.56.102",3333);
47          System.out.println("--Server telah terkoneksi.......");
48          DataInputStream din=new DataInputStream(socket.getInput
49          Stream());
50          DataOutputStream dout = new DataOutputStream(socket.
51          getOutputStream());
52          BufferedReader br = new BufferedReader(new InputStream
53          Reader(System.in));
54
55          System.out.println("--DHKE");
56          BigInteger secretA;
57          BigInteger generatorValue, primeValue, publicA, publicB,
58          shared;
59          String str;
60
61          //p
62          primeValue = new BigInteger("330273671788511023796232738
63          497605972489");
64          //g
65          generatorValue = new BigInteger("17");
66
67          System.out.print("Input Secret Key Client (integer) = ");
68          str = inSc.next();
69          secretA = new BigInteger(str);
70
71          publicA = generatorValue.modPow(secretA, primeValue);
72          System.out.println("Public key Client = "+publicA);
73          dout.writeUTF(publicA.toString());
74          dout.flush();
75
76          System.out.println("Menunggu Client mengirim Public Key
77          Server..");
78          str = din.readUTF();
79
80          publicB = new BigInteger(str);
81          System.out.println("Public key Server = "+publicB);
82
83          shared = publicB.modPow(secretA,primeValue);
84
85          System.out.println("Shared key = "+shared);
86      }
```

Pada tabel 5.8 merupakan *source code* dari algoritme DHKE pada sistem operasi Windows 10 yang berperan sebagai server atau penerima *key* yang dikirim oleh *client* di sistem operasi Windows XP. Skema pertukaran kunci menggunakan algoritme DHKE adalah dengan memulai koneksi antar kedua sistem operasi dengan *Socket* yang memiliki *port* 2929. *Port* 2929 inilah yang akan menjadi jalur pertukaran *key.* Jika koneksi berhasil dibuat, maka server akan menunggu *client* untuk mengirimkan *public key* nya. Kemudian *client* akan mengirimkan *public key* nya dimana *key* yang dikirim ini tidak bersifat rahasia dan dapat di ketahui oleh orang lain. Setelah server menerima *public key* dari *client* langkah selanjutnya adalah menghitung *shared key* dengan nilai *g* = 17 dan nilai *p* = 33027 36717885110237962327384976059724 89. Nilai *g* dan *p* ini bersifat rahasia yang hanya boleh diketahui oleh masing-masing pengirim sebagai *secret key*. *Shared key* yang sudah terbentuk selanjutnya akan digunakan sebagai *key* untuk proses enkripsi dan dekripsi algoritme Grain v1.

BAB 6 PENGUJIAN

Pada bab pengujian akan dilakukan beberapa metode untuk melakukan pengujian terhadap performa sistem berkaitan dengan kinerja algoritme Grain v1, SHA-3, dan DHKE. Pengujian akan dilakukan sesuai dengan perancangan pengujian yaitu melakukan pengujian *test vector,* pengujian validasi enkripsi dan dekripsi, pengujian performa dengan parameter waktu pembentukan *keystream,* dan pengujian serangan *sniffing.* Pada bab pengujian juga dituliskan tujuan dari masing-masing metode pengujian dan analisis hasil pengujian untuk memastikan tingkat keamanan informasi pada sistem yang dibangun.

6.1 Pengujian Validasi *Test Vector*

Test vector merupakan pengujian dengan cara memasukkan beberapa data masukan pada sistem untuk diproses dengan tujuan untuk menguji sistem. *Test vector* juga merupakan sebuah metode untuk melakukan pengujian perangkat lunak yang berhubungan dengan verifikasi dan validasi keluaran. Tujuan dari pengujian *test vector* adalah untuk memastikan apakan algoritme Grain v1 dan SHA-3 yang dibangun sudah berjalan dan mempunyai fungsionalitas yang sama seperti algoritme yang dibuat oleh penciptanya. *Test vector* dapat memastikan apakah desain algoritme yang dibuat sudah sama persis dengan desain yang dibuat oleh pencipta algoritme Grain v1 dan SHA-3.

6.1.1 Prosedur Pengujian

Prosedur yang dilakukan untuk *test vector* pada algoritme Grain v1 dan SHA-3 adalah dengan memasukkan beberapa masukan data sesuai dengan masukan *test vector* yang ditentukan oleh pencipta masing-masing algoritme. Setelah melakukan masukan data dan melakukan proses pada setiap algoritme, hasil keluaran dari masing-masing algoritme dibandingkan dengan hasil *test vector* pencipta algoritme Grain v1 dan SHA-3. *Test vector* akan bernilai valid apabila keluaran dari *test vector* sistem yang dibangun sama persis dengan keluaran *test vector* yang dilakukan oleh pencipta algoritme. Pada algoritme Grain v1, yang diuji *test vector adalah keystream* yang dihasilkan dengan masukan IV dan *key default* untuk *test vector*. Pada algoritme SHA-3 *test vector* dilakukan pada hasil produksi *digest* dengan masukan *default test vector* untuk algoritme SHA-3.

6.1.2 Hasil dan Analisis *Test Vector* Algoritme Grain v1

Test vector untuk algoritme Grain v1 adalah sebagai berikut :

1.	*Key*	: 00000000000000000000
	IV	: 0000000000000000
	Keystream	: 7b978cf36846e5f4ee0b
2.	*Key*	: 0123456789abcdef1234
	IV	: 0123456789abcdef

Keystream : 42b567ccc65317680225

Hasil produksi *keystream* dari algoritme Grain v1 yang digunakan pada sistem yang dibangun :

1. *Key* : 00000000000000000000
   IV : 0000000000000000
   Keystream : 7b978cf36846e5f4ee0b

2. *Key* : 0123456789abcdef1234
   IV : 0123456789abcdef
   Keystream : 42b567ccc65317680225

Dari pengujian *test vector* yang dilakukan menunjukkan bahwa hasil produksi *keystream* sistem yang dibangun sama persis dengan hasil uji *keystream* yang dilakukan oleh pembuat algoritme Grain v1 sehingga *source code* pada sistem dapat dikatakan valid.

6.1.3 Hasil dan Analisis *Test Vector* Algoritme SHA-3

Test vector untuk algoritme SHA-3 dilakukan dengan mencocokkan hasil produksi *digest source code library* dengan hasil *test vector* pada laman *https://www.di-mgt.com.au/sha_testvectors.html.* Berikut ini adalah hasil *test vector* untuk algoritme SHA-3 :

1. Pesan: “abc”
   SHA-3: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad 73b46fdf
2. Pesan: “” (*string* kosong)
   SHA-3: 6b4e03423667dbb7 3b6e15454f0eb1ab d4597f9a1b078e3f 5b5a6bc7
3. Pesan: "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
   SHA-3: 8a24108b154ada21 c9fd5574494479ba 5c7e7ab76ef264ea d0fcce33

Hasil produksi digest dari algoritme SHA-3 yang dihasilkan oleh sistem yang dibangun :

a) Pesan: “abc”
   SHA-3: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad 73b46fdf
b) Pesan: “” (*string* kosong)
   SHA-3: 6b4e03423667dbb7 3b6e15454f0eb1ab d4597f9a1b078e3f 5b5a6bc7
c) Pesan: "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
   SHA-3: 8a24108b154ada21 c9fd5574494479ba 5c7e7ab76ef264ea d0fcce33

Uji *test vector* yang dilakukan sebanyak tiga kali percobaan menunjukkan bahwa hasil produksi *digest* pencipta algoritme SHA-3 sama persis dengan hasil produksi *digest* sistem yang dibangun, sehingga *source code library* yang digunakan pada sistem yang dibangun ini bernilai valid.

6.2 Pengujian Validasi Enkripsi dan Dekripsi

Pengujian validasi enkripsi dan dekripsi berhubungan dengan keberhasilan kerja algoritme Grain v1. Pada pengujian algoritme Grain v1 dapat dikatakan valid ketika *plaintext* hasil dekripsi sama persis dengan *plaintext* ketika sebelum dienkripsi. Pengujian validasi ini dapat juga dijadikan sebagai salah satu acuan untuk menentukan apakah perhitungan dan desain algoritme yang ada pada sistem sudah benar atau belum. Hasil pengujian validasi ini harus menampilkan *plaintext* yang sama ketika sebelum enkripsi dan sesudah dekripsi. Tujuan dari pengujian validasi enkripsi dan dekripsi ini adalah untuk memastikan bahwa fungsi enkripsi yang ada pada desain algoritme Grain v1 yang dibangun memiliki kesesuaian dengan fungsi dekripsinya.

6.2.1 Prosedur Pengujian

Pengujian validasi enkripsi dan dekripsi dilakukan dengan mengisikan IV dan *key* pada algoritme Grain v1 untuk menghasilkan *keystream*. *Keystream* yang sudah diproduksi akan digunakan untuk proses enkripsi *plaintext* dengan panjang 80 bit. Hasil enkripsi yang berupa *ciphertext* akan diubah menjadi *plaintext* kembali dengan menggunakan fungsi dekripsi. *Plaintext* hasil dekripsi dan enkripsi kemudian dibandingkan, jika *plaintext* sebelum dienkripsi dan sesudah didekripsi sama persis, maka pengujian bernilai valid.

6.2.2 Hasil dan Analisis Validasi Enkripsi dan Dekripsi

Pada pengujian validasi ini masukan *plaintext* memiliki panjang 80 bit yang dienkripsi dengan *keystream* dari hasil proses produksi dengan IV dan *key* yang sudah dimasukkan. *Ciphertext* hasil enkripsi kemudian didekripsi dengan *keystream* yang dihasilkan IV dan *key* untuk dibandingkan dengan *plaintext* sebelum enkripsi. Berikut hasil uji validasi enkripsi dan dekripsi :

Tabel 6.1 Enkripsi *Plaintext*

No	*Plaintext*	*Keystream*	*Ciphertext*	Status
1	0111001101110101 0111001001100001 0110001001100001 0111100101100001	1010100011011110 1001011011010110 1000100000000011 1000111001100010	1101101110101011 1110010010110111 1110101001100010 1111011100000011 1101101010000100	Valid
2	0111001101110101 0111001001100001 0110001001100001 0111100101100001	1000001111111000 1101011111110011 1010001111100001 1111110000001110 1001111111111111	1111000010001101 1010010110010010 1100000110000000 1000010101101111	Valid

Pada tabel 6.1 merupakan perhitungan manualisasi nilai biner dari *plaintext, keystream,* dan *ciphertext* yang dihasilkan oleh algoritme Grain v1. Pada pengujian ini *plaintext* yang digunakan adalah kata 'surabaya' dengan nilai bit sepanjang 80 bit. *Ciphertext* yang dihasilkan merupakan hasil dari XOR nilai biner *plaintext* dan nilai biner *keystream.* IV yang digunakan pada pengujian pertama dan kedua

bernilai string ‘invinity’ sedangkan *key* pada pengujian pertama bernilai string ‘invinity’ dan pada pengujian kedua bernilai string ‘pemenang’.

Tabel 6.2 Dekripsi *Ciphertext*

No	*Ciphertext*	*Keystream*	*Plaintext*	Status
1	1101101110101011 1110010010110111 1110101001100010 1111011100000011 1101101010000100	1010100011011110 1001011011010110 1000100000000011 1000111001100010	0111001101110101 0111001001100001 0110001001100001 0111100101100001	Valid
2	1111000010001101 1010010110010010 1100000110000000 1000010101101111	1000001111111000 1101011111110011 1010001111100001 1111110000001110 1001111111111111	0111001101110101 0111001001100001 0110001001100001 0111100101100001	Valid

Pada tabel 6.2 merupakan perhitungan manual proses dekripsi *ciphertext.* Hasil perhitungan biner dari *ciphertext* di-XOR-kan dengan biner dari *keystream* untuk mendapatkan nilai biner dari *plaintext.* Setelah mendapatkan nilai biner dari *plaintext* selanjutnya biner diubah ke bentuk string agar dapat dibaca kembali. Pada tabel 6.2 dapat dibandingkan jika hasil dekripsi *ciphertext* bernilai sama dengan nilai biner *plaintext* sebelum didekripsi, dengan itu pengujian validasi enkripsi dan dekripsi bernilai valid.

6.3 Pengujian Fungsional

Pada pengujian fungsional dilakukan dengan teknik *black box testing* dimana teknik pengujian ini berfokus pada keluaran hasil dari respon masukan pada sistem dalam hal ini pengujian mengesampingkan bagaimana sebuah proses dijalankan oleh sistem. Tujuan dari pengujian fungsional dengan menggunakan teknik *black box testing* adalah untuk mengetahui fungsi yang dapat dijalankan sistem ketika mendapat respon masukan sebagai proses mendapatkan keluaran. *Black box testing* dilakukan dengan menjalankan sistem untuk mengetahui apakah masih ada *bug* atau *error* dari program. Pengujian fungsional dapat bernilai valid ketika sistem dapat memberikan keluaran sebagai respon dari masukan yang diberikan sesuai dengan perancangan.

6.3.1 Prosedur Pengujian

Pengujian fungsional dilakukan dengan cara memberikan masukan pada sistem untuk mendapatkan keluaran atau respon dari proses yang dijalankan. Keluaran dari sistem kemudian diamati dan disesuaikan dengan perancangan sistem yang sudah dilakukan pada bab sebelumnya. Pengujian fungsional dilakukan menggunakan lima parameter pengujian yaitu : uji *keystream* algoritme Grain v1, uji masukan IV dan *key,* uji enkripsi dan dekripsi Grain v1, pengujian *digest* algoritme SHA-3, dan pengujian pertukaran *key* algoritme DHKE.

6.3.2 Hasil dan Analisis

Pada pengujian fungsionalitas dilakukan beberapa uji terhadap sistem untuk mengetahui keluaran sebagai hasil proses masukan sistem. Berikut adalah tabel hasil pengujian fungsional pada sistem yang dibangun :

Tabel 6.3 Pengujian Fungsional

No	*Test Name*	Hasil yang diharapkan	Hasil	Jenis Pengujian	Status
1	Pengujian hasil *keystream*	Sistem dapat menghasilkan *keystream* sebanyak jumlah bit biner *plaintext*	Sistem dapat menghasilkan *keystream* sepanjang bit biner *plaintext*	*Black Box*	Valid
2	Pengujian masukan IV dan *key*	Sistem dapat menerima IV sebanyak kurang dari sama dengan 64 bit dan *key* sebanyak kurang dari sama dengan 80 bit	Sistem dapat menerima IV kurang dari sama dengan 64 bit dan dapat menerima masukan *key* sebanyak kurang dari sama dengan 80 bit	*Black Box*	Valid
3	Pengujian enkripsi dan dekripsi *plaintext*	Sistem dapat melakukan enkripsi *plaintext* dan dapat mengembalikan *ciphertext* menjadi *plaintext* melalui proses dekripsi.	Sistem dapat melakukan enkripsi *plaintext* dan mengembalikan *ciphertext* menjadi *plaintext* menggunakan proses dekripsi.	*Black Box*	Valid
4	Pengujian *digest* SHA-3	Sistem dapat memproduksi *digest* untuk keperluan proses *hashing*	Sistem dapat memproduksi *digest* untuk proses *hashing*	*Black Box*	Valid
5	Pengujian pertukaran *key*	Sistem dapat bertukar *key* antar client dengan menggunakan DHKE	Sistem dapat bertukar *shared key* dengan menggunakan algoritme DHKE	*Black Box*	Valid

Dari tabel 6.3 dapat diketahui bahwa pengujian fungsional dengan teknik *black box* menghasilkan keluaran yang sesuai dengan perancangan sistem sehingga pengujian fungsional ini dapat dikatakan valid. Pada pengujian uji *keystream* pada algoritme Grain v1, memberikan hasil bahwa sistem dapat menghasilkan *keystream* sebanyak panjang bit *plaintext* yang dimasukkan. Pengujian masukan IV dan *key* bernilai valid karena untuk memproduksi *keystream* yang sesuai dengan *test vector,* sistem harus dapat dimasukkan IV sebanyak kurang dari sama dengan

64 bit dan *key* sebanyak kurang dari sama dengan 80 bit. Pengujian enkripsi dan dekripsi dilakukan untuk mengetahui apakah sistem dapat melakukan proses dekripsi maupun enkripsi sesuai pengujian validasi enkripsi dan dekripsi. Pengujian *digest* SHA-3 dilakukan untuk memastikan bahwa *library* yang dipakai pada penelitian ini adalah sebuah *library* yang bernilai valid karena dapat menghasilkan *digest* yang benar dan dibuktikan melalui *test vector.* Pengujian pertukaran *key* menggunakan DHKE dilakukan dengan menggunakan dua sistem operasi yang berperan sebagai *client* dan *server* yang sedang bertukar *key.*

6.4 Pengujian Performa

Pengujian performa pada penelitian ini dilakukan dengan menghitung dan mencatat waktu hasil eksekusi sistem. Pengujian performa sistem dibagi menjadi dua bagian, bagian pertama menghitung waktu eksekusi program algoritme Grain dengan variasi masukan IV dan *key* namun *plaintext* sama. Bagian kedua menghitung waktu eksekusi sistem sebelum dan sesudah dilengkapi dengan algoritme Grain untuk proses enkripsi dan dekripsi. Pengujian performa bertujuan untuk mengetahui kinerja sistem dengan cara membandingkan waktu eksekusi sistem pada saat sebelum dan sesudah dilengkapi dengan algoritme Grain.

6.4.1 Prosedur Pengujian

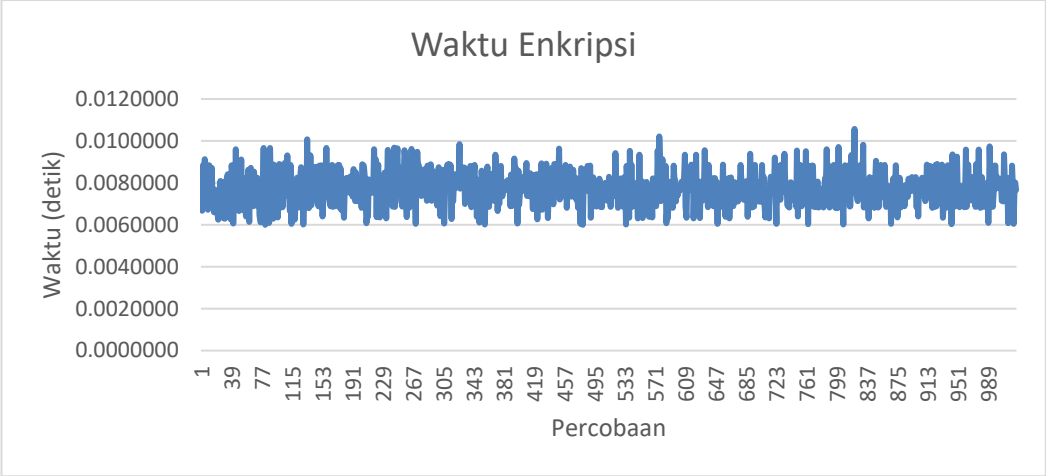
Pengujian performa dilakukan dengan menjalankan dua prosedur sebagai berikut :

1. Prosedur pertama pada pengujian performa pada penelitian ini adalah dengan melakukan pencatatan waktu eksekusi proses enkripsi dan dekripsi algoritme Grain dengan variasi *key* dan IV sebanyak 1024 variasi dengan *plaintext* yang sama yaitu berupa *string* yang bernilai '1234567890'. Pemilihan string sebanyak sepuluh digit bertujuan untuk menghasilkan *plaintext* sepanjang 80 bit sesuai dengan pengujian *test vector* yang dilakukan oleh pencipta algoritme Grain.
2. Prosedur kedua adalah dengan melakukan pencatatan waktu eksekusi sistem yang melibatkan basis data dengan melakukan enkripsi dan dekripsi yang dimulai dari parameter ke-0 sampai dengan parameter ke-9 sebanyak 40 kali percobaan.

6.4.2 Hasil dan Analisis

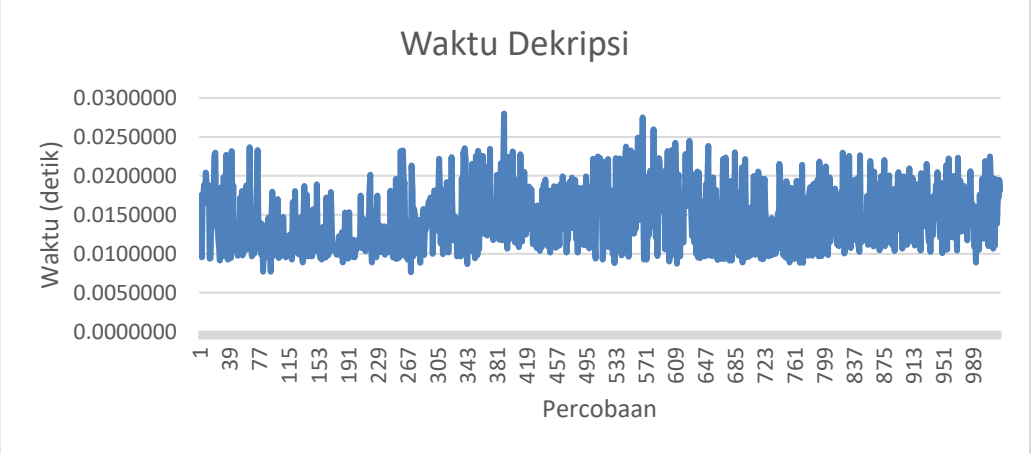
Hasil dan Analisis pada pengujian performa pada penelitian ini disajikan dalam bentuk grafik untuk mempermudah membaca hasil pengujian. Gambar 6.1 merupakan grafik waktu hasil eksekusi dari proses enkripsi algoritme Grain dengan masukkan *key* dan IV sebanyak 1024 variasi. Hasil dari pengujian performa menunjukkan bahwa waktu rata-rata yang diperlukan oleh algoritme Grain untuk mengenkripsi 80 bit *plaintext* adalah sebesar 0,0077450 detik dan waktu terlama yang tercatat untuk mengenkripsi adalah sebesar 0,0105720 detik sedangkan untuk waktu tercepat sebesar 0.006 detik. Variasi waktu yang tercatat diakibatkan

kondisi pemakaian *memory* perangkat yang digunakan pada saat pengujian juga fluktuatif.



Gambar 6.1 Grafik Waktu Eksekusi Enkripsi

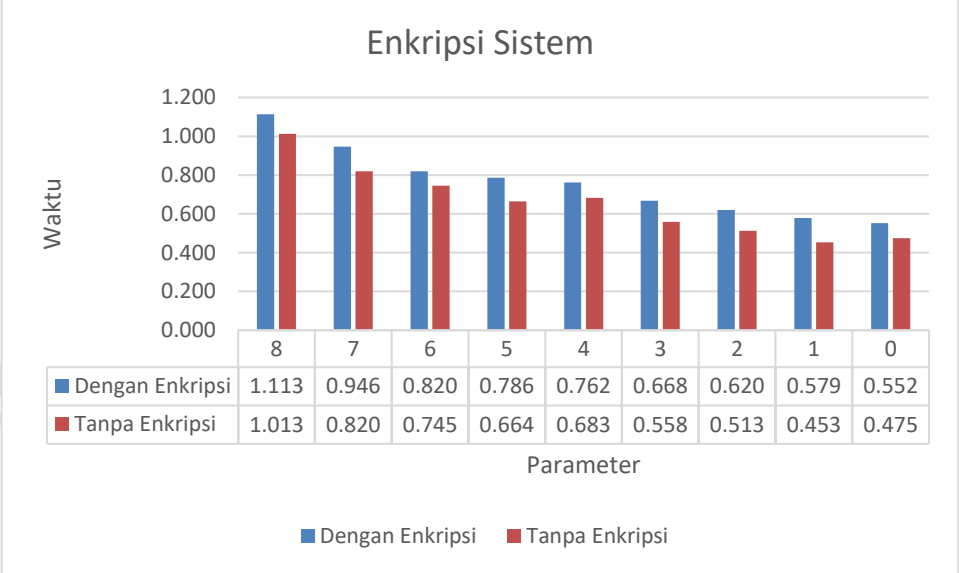
Pada gambar 6.2 adalah grafik yang menunjukkan waktu eksekusi dekripsi algoritme Grain dengan *key* dan IV 1024 variasi. Hasil pengujian performa dekripsi menunjukkan rata-rata waktu yang diperlukan untuk mendekripsi 80bit *plaintext* adalah sebesar 0,0145436 detik. Waktu terlama yang tercatat pada proses dekripsi adalah sebesar 0,0279942 detik sedangkan waktu tercepat adalah 0,0076205 detik. Selisih waktu yang terjadi antara proses enkripsi dan dekripsi terjadi karena pada proses dekripsi dilakukan dua tahap yaitu mengenkripsi data kemudian didekripsi agar dapat dibaca kembali yang juga menyebabkan waktu hasil proses dekripsi bervariasi.



Gambar 6.2 Grafik Waktu Eksekusi Dekripsi

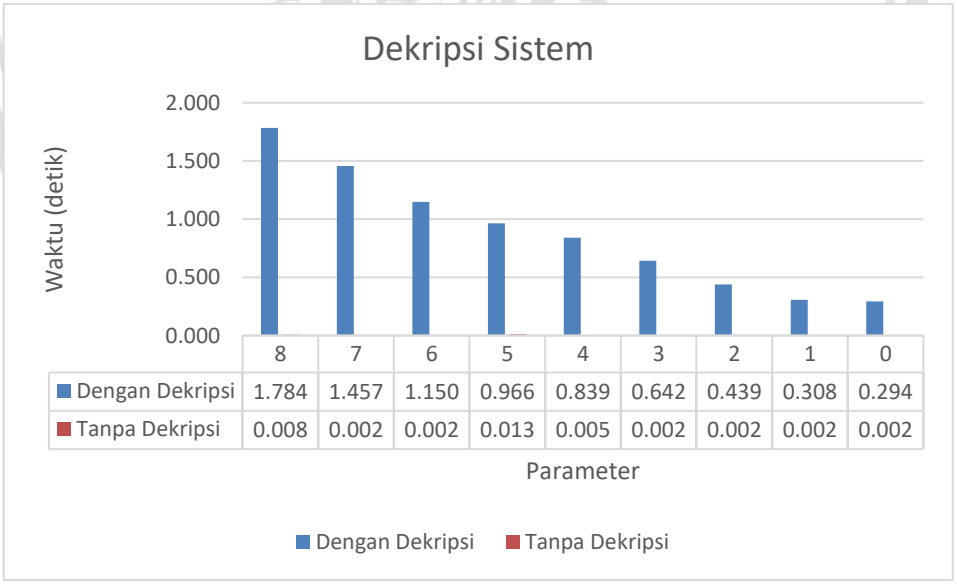
Gambar 6.3 merupakan grafik yang menunjukkan hasil pengujian performa dengan skenario ke-dua yang dilakukan dengan mencatat waktu hasil eksekusi sistem dengan menggunakan enkripsi dan tanpa enkripsi. Pengujian skenario ke-dua ini dilakukan menggunakan sembilan parameter yang terdapat pada basis data sistem. Hasil pencatatan waktu menunjukkan bahwa waktu eksekusi sistem lebih cepat saat sistem belum dilengkapi dengan proses enkripsi pada basis data

dibandingkan dengan pada saat sistem sudah dilengkapi proses enkripsi. Perbedaan waktu eksekusi sistem pada saat sesudah dan sebelum dilengkapi dengan proses enkripsi terjadi karena proses enkripsi pada basis data memerlukan waktu lebih lama untuk menjalankan proses menghasilkan *keystream* dan kalkulasi pada algoritme Grain. Banyaknya parameter yang dienkripsi berbanding lurus dengan waktu yang diperlukan sistem untuk menjalankan eksekusi. Semakin banyak parameter yang dienkripsi maka waktu yang dibutuhkan sistem untuk menjalankan sekali eksekusi juga akan semakin lama.



	8	7	6	5	4	3	2	1	0
Dengan Enkripsi	1.113	0.946	0.820	0.786	0.762	0.668	0.620	0.579	0.552
Tanpa Enkripsi	1.013	0.820	0.745	0.664	0.683	0.558	0.513	0.453	0.475

Gambar 6.3 Grafik Waktu Eksekusi Sistem Enkripsi



	8	7	6	5	4	3	2	1	0
Dengan Dekripsi	1.784	1.457	1.150	0.966	0.839	0.642	0.439	0.308	0.294
Tanpa Dekripsi	0.008	0.002	0.002	0.013	0.005	0.002	0.002	0.002	0.002

Gambar 6.4 Grafik Waktu Eksekusi Sistem Dekripsi

Gambar 6.4 merupakan grafik hasil pencatatan waktu pengujian performa sistem dengan menjalankan skenario ke-dua yang dilakukan dengan mencatat waktu eksekusi sistem pada saat sesudah dan sebelum dilengkapi proses dekripsi. Pengujian performa skenario ke-dua ini menggunakan sembilan parameter dari

basis data yang didekripsi. Hasil pencatatan waktu menunjukkan bahwa waktu eksekusi sistem jauh lebih cepat pada saat sistem tidak dilengkapi proses dekripsi dibanding dengan sistem yang dilengkapi proses dekripsi basis data. Perbedaan waktu eksekusi yang signifikan ini dikarenakan pada proses dekripsi dilakukan perubahan *ciphertext* menjadi *plaintext* dimana proses ini melibatkan kalkulasi pada algoritme Grain. Waktu eksekusi sistem pada saat sesudah dan sebelum dilengkapi proses dekripsi berbanding lurus dengan banyaknya parameter yang didekripsi, semakin banyak parameter yang didekripsi maka waktu yang diperlukan sistem dalam sekali eksekusi juga akan semakin lama.

6.5 Pengujian *Sniffing*

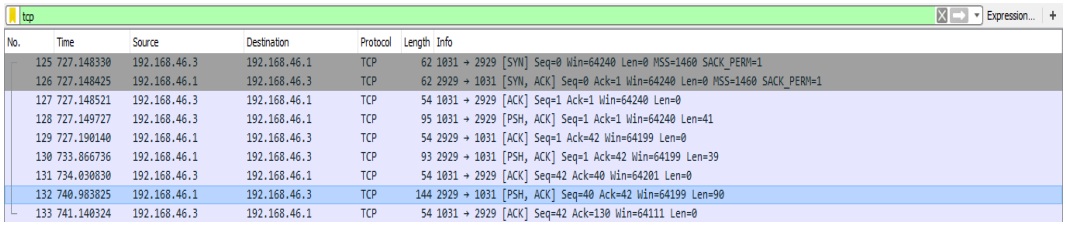
Sniffing merupakan salah bentuk serangan pada jaringan komputer dengan cara menyadap paket yang dikirimkan melalui jaringan internet. *Sniffing* dilakukan untuk mengetahui lalu lintas data yang lewat pada jaringan komputer sehingga *attacker* dapat mengetahui isi dari paket yang dikirimkan. Pada pengujian *sniffing* ini dilakukan aktifitas *capture* paket data yang dilewatkan sistem pada saat proses pengiriman antar sistem operasi yang berperan sebagai *server* dan *client* menggunakan *tools* Wireshark v2.0.1 pada protokol TCP. Tujuan pengujian *sniffing* adalah untuk memastikan bahwa paket yang dikirimkan melalui protokol TCP sudah dalam bentuk *ciphertext* dan tidak dapat dibaca oleh *sniffer.*

6.5.1 Prosedur Pengujian

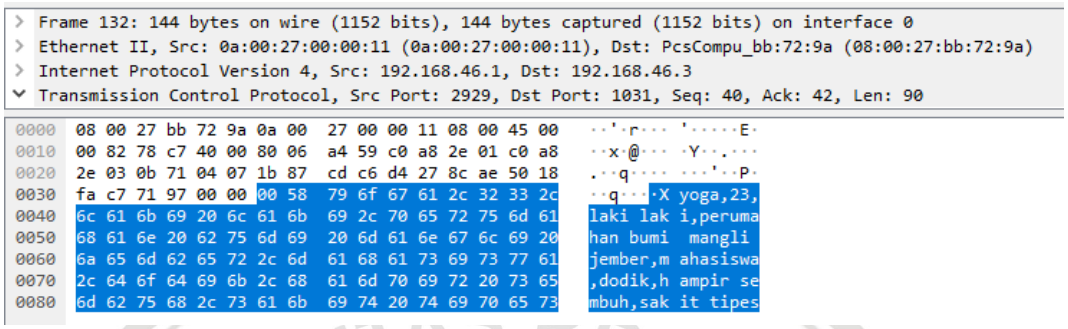
Pengujian *sniffing* dilakukan dengan cara melakukan simulasi pengiriman data melalui protokol TCP antar dua sistem operasi. Simulasi dilakukan sebanyak dua kali dimana simulasi pertama dilakukan tanpa menggunakan algoritme Grain sebagai pengaman sedangkan pada simulasi kedua, Wireshark akan melakukan *capture* paket data pada protokol TCP yang dilengkapi algoritme Grain untuk melihat semua data yang dilewatkan protokol TCP. Pada saat simulasi pengiriman dilakukan, *tools* Wireshark akan meng-*capture* semua data yang melewati protokol TCP terasuk data yang dikirim oleh *server* dan *client.* Hasil *capture* protokol TCP akan dilihat dan melihat informasi *capture* apakah data yang terekam masih berbentuk *plaintext* atau sudah melalui proses enkripsi sehingga yang terekam adalah data *ciphertext.* Pengujian *sniffing* ini akan bernilai valid jika data yang terekam pada *tools* Wireshark sudah berbentuk *ciphertext.*

6.5.2 Hasil dan Analisis

Gambar 6.5 merupakan hasil *capture* protokol TCP menggunakan *tools* Wireshark pada saat simulasi pengiriman data dilakukan. Pada gambar 6.1 disajikan hanya paket-paket yang melalui protokol TCP saja karena data yang dikirimkan dilewatkan melalui protokol TCP. Untuk mengetahui apakah data yang dikirimkan sudah terenkripsi atau belum dilakukan pembukaan informasi detail dari salah satu paket yang terekam. Berikut adalah gambar hasil *capture tools wireshark* pada protokol TCP pada saat simulasi pengiriman data :

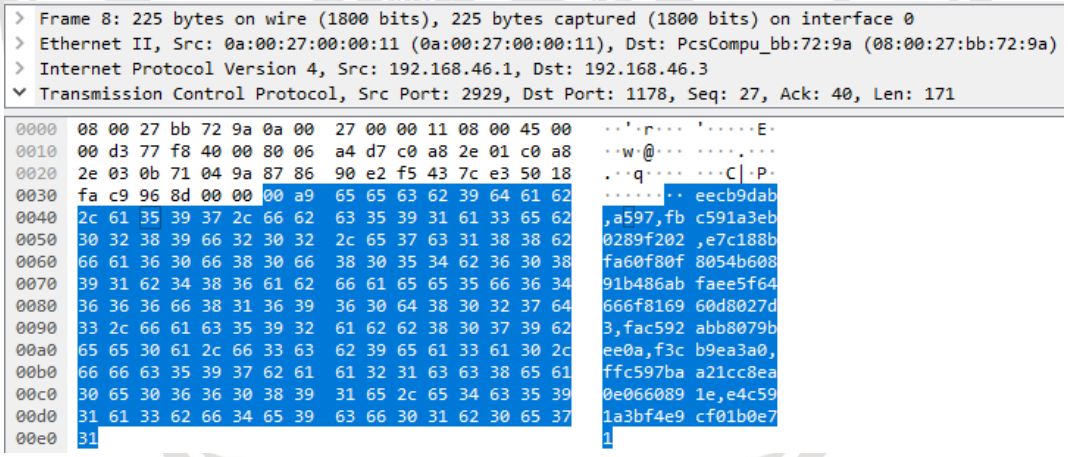


Gambar 6.5 Filter Protokol TCP



Gambar 6.6 *Capture* Paket Pada Pengiriman Tanpa Grain

Pada gambar 6.6 merupakan salah satu *publish Message* dimana pada paket *publish* berisi beberapa informasi seperti informasi panjang paket, *src port* (port asal)*, Dst port* (port tujuan)*,* dan *message.* Terlihat bahwa pada bagian *message* pesan yang terekam Wireshark masih dapat dibaca oleh *sniffer* karena pesan yang dikirim masih berupa *plaintext* dan belum dienkripsi oleh algoritme Grain.



Gambar 6.7 *Capture* Paket Pada Pengiriman dengan Grain

Gambar 6.7 merupakan *capture* salah satu paket data yang dilewatkan pada protokol TCP dimana *message* pada paket itu berisi sederet huruf dan angka tidak beraturan atau *ciphertext* hasil dari enkripsi algoritme Grain. Gambar 6.7 membuktikan bahwa setelah menggunakan algoritme Grain sebagai pengaman sistem, para *attacker* atau *sniffer* hanya akan dapat membaca *ciphertext* ketika merekam pesan yang dikirimkan server ke *client.* Informasi *ciphertext* yang terekam pada gambar 6.7 merupakan hasil enkripsi berupa *hexadecimal* dari variabel nama, usia, jenis kelamin pasien, alamat, pekerjaan, gejala dan keterangan rekam medis pasien.

6.6 Pengujian *Brute Force Attack*

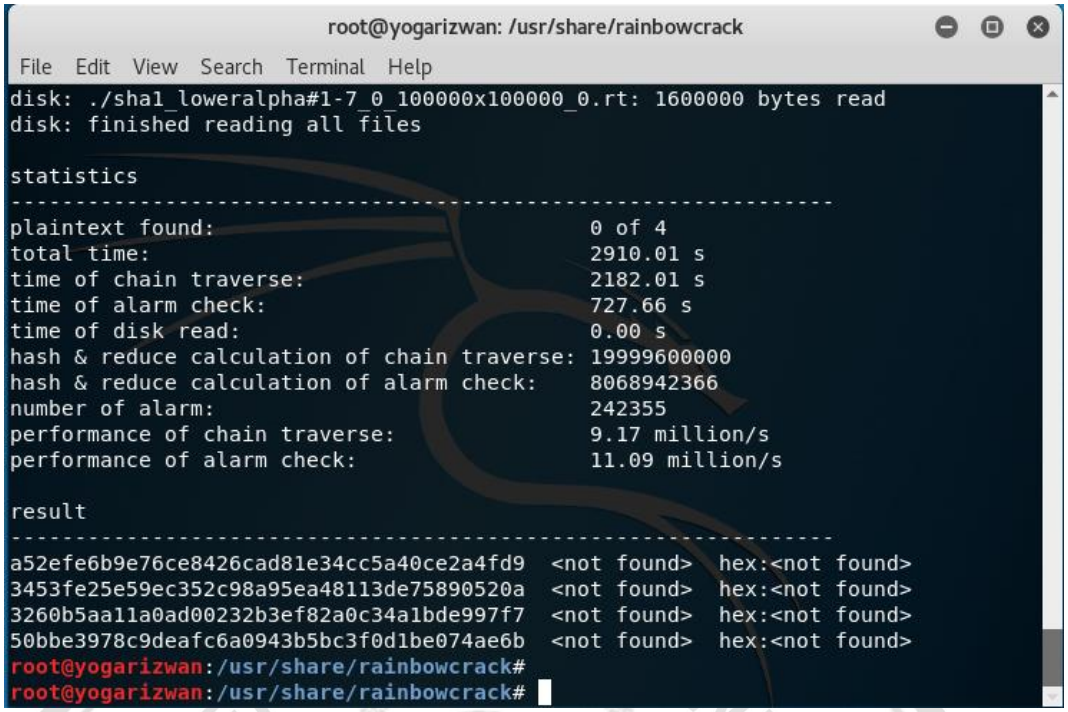
Brute force attack atau serangan brutal pada sistem bertujuan untuk mencoba membuka secara acak hasil dekripsi dari *ciphertext. Brute force attack* pada pengujian ini menggunakan perangkat lunak pihak ketiga yang bernama Rainbowcrack. Perangkat lunak ini akan mencoba satu persatu kemungkinan kata atau kalimat yang merupakan hasil dekripsi *ciphertext.* Apabila ditemukan kecocokan dari kata yang dicoba oleh perangkat lunak Rainbowcrack, maka perangkat lunak akan mengeluarkan hasil dekripsi *ciphertext.* Tujuan dilakukan pengujian *brute force attack* adalah menjamin apakah hasil dekripsi dari algoritme Grain sebagai pengaman sistem tidak dapat dibobol dengan mudah oleh penyerang yang mencoba untuk mencuri informasi data rekam medis pada saat proses pengiriman.

6.6.1 Prosedur Pengujian

Pengujian *brute force attack* dilakukan dengan menggunakan perangkat lunak pihak ketiga pada sistem operasi Kali Linux yang bernama Rainbowcrack. Perangkat lunak Rainbowcrack akan mencoba menyerang sebanyak empat *ciphertext* hasil enkripsi algoritme Grain. Rainbowcrack akan mencoba satu persatu kata pada *file* bernama *wordlists* yang berisi kumpulan kata yang memiliki kemungkinan menjadi hasil dekripsi dari *ciphertext* yang diserang. Jumlah dari kata yang ada pada *file wordlists* adalah sebanyak 100000 kata. Hasil dari pengujian ini berupa status berhasil atau tidaknya Rainbowcrack dalam mencari *plaintext* dari hasil enkripsi algoritme Grain. Parameter waktu pengujian juga dicatat untuk mengetahui berapa lama waktu yang diperlukan untuk melakukan *brute force attack* dengan 100000 kata pada *ciphertext* yang dihasilkan algoritme Grain.

6.6.2 Hasil dan Analisis

Gambar 6.8 memuat informasi dari pengujian *brute force attack* yang dilakukan pada empat *ciphertext* hasil enkripsi algoritme Grain. Terlihat pada gambar 6.8 bahwa perangkat lunak Rainbowcrack tidak dapat menemukan *plaintext* dari keempat *ciphertext* yang diserang. Waktu yang diperlukan Rainbowcrack untuk melakukan *brute force attack* dengan 100000 *wordlists* pada empat *ciphertext* adalah 2910.01 detik atau selama 48,5 menit. Dari hasil ini dapat diketahui bahwa *ciphertext* yang dihasilkan oleh algoritme Grain tidak dapat ditembus oleh serangan *brute force attack,* hal ini karena perangkat lunak Rainbowcrack tidak memiliki logika dan dapat menghitung *keystream* yang dibuat oleh algoritme Grain. Hasil dari pengujian *brute force attack* memuat semua informasi pada saat pengujian dilakukan pada gambar 6.8 berikut :



Gambar 6.8 Hasil Pengujian *Brute force attack*

BAB 7 PENUTUP

7.1 Kesimpulan

Penelitian yang telah dilakukan menghasilkan beberapa kesimpulan mengenai implementasi algoritme Grain v1 dan SHA-3 pada pengamanan data rekam medis.

1. Algoritme Grain v1 dan SHA-3 224 dapat diterapkan pada sistem untuk menjaga kerahasiaan dan keutuhan data pada saat proses pengiriman data rekam medis. Algoritme Grain v1 dapat menghasilkan *keystream* sebanyak bit *string* dari *plaintext* dengan melakukan proses masukan nilai IV sepanjang 64 bit dan *key 80 bit.* Nilai *keystream* yang dihasilkan akan sepanjang bit dari *string plaintext* yang kemudian setiap bitnya di-XOR-kan dengan nilai bit *plaintext,* kemudian disimpan pada basis data. Algoritme SHA-3 dapat di implementasikan pada sistem dimana *password* dari aktor akan diubah menjadi nilai *hash* yang kemudian dicocokkan dengan nilai *hash* password yang sudah tersimpan didalam basis data sebelumnya.
2. Hasil pengujian validasi *keystream* dengan melakukan uji *test vector* yang ditulis oleh pencipta algoritme Grain v1 menunjukkan bahwa *keystream* yang dihasilkan oleh sistem bernilai sama.
3. Kinerja dari algoritme Grain v1 pada proses enkripsi dan dekripsi dilakukan dengan mengenkripsi dan mendekripsi string sepanjang 80 bit. Pengujian performa algoritme Grain v1 menghasilkan waktu enkripsi dengan rata-rata 0,0077450 detik dan waktu tercepat pada proses enkripsi adalah sebesar 0,006 detik sedangkan waktu terlama adalah 0,0105720 detik. Waktu dekripsi yang dihasilkan pada saat pengujian mencatat rata-rata waktu yang diperlukan untuk satu kali proses dekripsi adalah sebesar 0,0145436 detik dan waktu tercepat untuk proses dekripsi adalah sebesar 0,0076205 detik sedangkan waktu terlama adalah sebesar 0,0279942 detik. Pengujian performa sistem yang melibatkan basis data menunjukkan bahwa waktu eksekusi akan lebih cepat apabila sistem tidak dilengkapi proses enkripsi dan dekripsi.

7.2 Saran

Pembuatan dan pengimplementasian algoritme Grain v1 dan SHA-3 244 pada sistem yang telah dibangun memiliki beberapa saran yang dapat dilakukan untuk kepentingan pengembangan sistem selanjutnya.

1. Sistem yang dibangun pada penelitian ini mengimplementasikan algoritme Grain v1 dan SHA-3 sebagai pengaman pengiriman data. Pada sistem ini memuat fungsi enkripsi dan dekripsi *text.* Untuk penelitian selanjutnya dapat penambahan fungsi untuk enkripsi data selain *text.*

2. Untuk menambah tingkat keamanan sistem, masukan *key* dan IV untuk sistem selanjutnya dapat dilakukan secara manual

DAFTAR PUSTAKA

Andalia, F. & Setiawan, B. E., 2015. PENGEMBANGAN SISTEM INFORMASI PENGOLAHAN DATA PENCARI KERJA PADA DINAS SOSIAL DAN TENAGA KERJA KOTA PADANG. *Jurnal Ilmiah Komputer dan Informatika (KOMPUTA),* Volume 4.

Angga, D., 2015. *RANCANG BANGUN SISTEM INFORMASI REKAM MEDIK RAWAT JALAN DI RUMAH SAKIT UMUM MITRA MULIA HUSADA BANDAR JAYA KABUPATEN LAMPUNG TENGAH.* Bandar Lampung: Universitas Lampung.

Bintari, R., Budiwati, S. D. & Tambunan, T. D., 2017. APLIKASI REKAM MEDIK DAN PEMESANAN OBAT BERBASIS WEB. *e-Proceeding of Applied Science,* Volume Vol. 3 No.2, p. 695.

Bokhari, M. U., 2014. A Detailed Analysis of Grain family of Stream. *I.J. Computer Network and Information Security,* pp. 34-40.

Candra, B., Wahyudi, J. & Hermawansyah, 2014. PENGEMBANGAN SISTEM KEAMANAN UNTUK TOKO ONLINE BERBASIS KRIPTOGRAFI AES MENGGUNAKAN BAHASA PEMROGRAMAN PHP DAN MYSQL. *Jurnal Media Infotama,* Volume 11, pp. 31 - 40.

Ditjen Yankes : 10 , 1993. *Ditjen Yankes 1993 : 10.* Jakarta: s.n.

Durairajan, M. S. & Saravanan, R., 2014. Biometrics Based Key Generation using Diffie Hellman Key Exchange for Enhanced Security Mechanism. *International Journal of ChemTech Research,* Volume 6, pp. 4359-4365.

Fakhrusy, M., 2016. *Implementasi HMAC-SHA-3-Based One Time Password pada Skema Two-Factor Authentication,* Bandung: Fakultas Teknik Mesin dan Dirgantara Institut Teknologi Bandung.

Hell, M., Johansson, T. & Meier, W., 2006. Grain - A Stream Cipher for Constrained Environments.

Herlambang, S., 2010. Studi dan Analisis Grain Cipher.

Herman, S., Fajrillah, A. A. N. & Andreswari, R., 2017. ENTERPRISE ARCHITECTURE DESIGN OF HOSPITAL. *Jurnal Rekayasa Sistem & Industri,* Volume 4, p. 37.

Hidayat, A. D. & Afrianto, I., 2017. Sistem Kriptografi Citra Digital Pada Jaringan Intranet Menggunakan Metode Kombinasi Chaos Map Dan Teknik Selektif. *ULTIMATICS,* Volume IX, pp. 59 - 66.

Kurniawan, F., Kusyanti, A. & Nurwasito, H., 2017. Analisis dan Implementasi Algoritma SHA-1 dan SHA-3 pada Sistem Autentikasi Garuda Training Cost. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* Volume 1, pp. 803-812.

Nurrohmah, A., Kusyanti, A. & Primananda, R., 2017. Implementasi Algoritme Grain V1 Dan 128 Bit Pada Arduino Mega 2560. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* Volume 2, pp. 1436-1445.

Oktariano, A., 2016. PERANCANGAN DAN IMPLEMENTASI REKAM MEDIS PASIEN POLI UMUM DI RUMAH SAKIT RIMBO MEDICA MENGGUNAKAN PHP DAN MySQL. *SCIENTIA JOURNAL,* Volume 4, p. 317.

PERATURAN MENTERI KESEHATAN REPUBLIK INDONESIA, 1989. *No.749a/Menkes/Per/ XII/1989.* Jakarta: s.n.

Romine, C. H., 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *National Institute of Standards and Technology.*

Shahid, S., Kusyanti, A. & Primananda, R., 2017. Implementasi Algoritme Grain V1 Dan 128 Bit Pada Raspberry PI. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* Volume 2, pp. 1572-1581.

Sugiarto, A., 2017. *Perancangan Kriptografi Simetris Bebasais pada Pola Gambar Rumah Adat Tongkongan.* Salatiga: Fakultas Teknologi Informasi Universitas Kristen Satya Wacana.

Undang Undang Praktik kedokteran, 2014. *UNDANG-UNDANG REPUBLIK INDONESIA NOMOR 29 Pasal 46 ayat (1),* s.l.: s.n.

Wiejaya, C., 2016. KEAMANAN DATA DENGAN METODE KRIPTOGRAFI KUNCI PUBLIK. *Jurnal TIMES,* Volume V, pp. 11 - 15.