PENGAMANAN DATA PADA MEDIA PENYIMPANAN CLOUD MENGGUNAKAN TEKNIK ENKRIPSI DAN *SECRET SHARING*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Mohammad Fachry
NIM: 135150200111089

PROGAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PENGAMANAN DATA PADA MEDIA PENYIMPANAN *CLOUD* MENGGUNAKAN TEKNIK ENKRIPSI DAN *SECRET SHARING*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Mohammad Fachry
NIM: 135150200111089

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

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T, M.Sc	Kasyful Amron, S.T, M.Sc
NIK: 201102 831228 2 001	NIP: 19750803 200312 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

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

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, 16 April 2018

Mohammad Fachry

NIM: 135150200111089

KATA PENGANTAR

Assalamuaikum Wr. Wb

Puji syukur atas kehadiran Allah SWT yang selalu melimpakan rahmat dan karunia – Nya, sehingga peneliti dapat menyelesaikan skripsi untuk meraih gelar sarjana dengan judul "Pengamanan data pada media penyimpanan *cloud* menggunakan teknik enkripsi dan *secret sharing*". Dalam proses penulisan ini terdapat pihak – pihak yang terlibat, baik bantuan moral maupun materil oleh karena itu penulis mengucapkan rasa terimakasih kepada:

1. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
2. Bapak Heru Nurwarsito, Ir., M.Kom. selaku Wakil Ketua I Bidang Akademik Fakultas Ilmu komputer Universitas Brawijaya.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika Universitas Brawijaya.
4. Bapak Agus Wahyu Widodo, S.T, M.Cs. selaku Ketua Prodi Teknik Informatika Universitas Brawijaya.
5. Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing I yang telah memberikan ilmu, motivasi, saran, waktu, serta membantu proses penyusunan penulisan laporan.
6. Bapak Kasyful Amron, S.T, M.Sc selaku dosen pembimbing II yang telah memberikan ilmu, motivasi, saran, waktu serta membantu penulisan laporan.
7. Bapak dan Ibu dosen Fakultas Ilmu Komputer yang telah memberikan ilmu untuk mendukung penyelesaian skripsi ini, serta para karyawan Fakultas Ilmu Komputer yang telah membantu dalam proses administrasi skripsi.
8. Kedua orang tua tercinta Bapak Aswan dan Ibu Fadhillah, serta adik-adik yang telah memberikan doa, motivasi dan dukungan baik secara moril dan materi kepada penulis untuk menyelesaikan skripsi ini.
9. Teman – teman penulis, Ika, Firly, Raja, St Ananda, Kiki, Pras yang memberikan masukan dalam penulisan skripsi.
10. Seluruh Keluarga besar Selflie dan Nol Derajat Film yang memberikan semangat dukungan serta doa dalam penyelesaian skripsi.

Semoga Allah SWT senantiasa memberikan rahmat dan nikmat-Nya kepada semua pihak, penulis menyadari bahwa masih banyak kekurangan dalam penulisan ini. Penulis berharap adanya kritik dan saran untuk penelitian selanjutnya, semoga skripsi ini dapat bermanfaat bagi pembaca.

Malang, 16 April 2018

Penulis

mfachryss9@gmail.com

ABSTRAK

Cloud computing adalah teknologi yang biasa digunakan untuk melakukan penyimpanan data secara *online*. Dengan hanya terhubung melalui internet, menyimpan maupun mengakses data akan lebih mudah dilakukan. Akan tetapi, teknologi ini juga menimbulkan masalah dalam hal keamanan. Kriptografi adalah bidang ilmu yang mempelajari bagaimana sebuah data dapat diamankan. Untuk itu, penerapan ilmu kriptografi sudah tidak asing lagi dalam sistem berbasis *cloud computing*. Algoritme-algoritme baru serta kombinasi algoritme-algoritme kriptografi terus dikembangkan untuk menjamin aspek-aspek keamanan seperti *confidentiality*, *integrity* dan *availability*. Pada penelitian ini, sebuah sistem dibangun dengan menerapkan 2 metode kriptografi yaitu enkripsi dan *secret sharing* untuk mengamankan data. Metode enkripsi yang diterapkan adalah algoritme Grain dan AES sedangkan metode *secret sharing* yang diterapkan adalah Shamir's Secret Sharing. Data yang diamankan akan disimpan dalam 2 media penyimpanan yang berbeda yaitu *cloud storage* dan memori internal laptop. Berdasarkan pengujian yang dilakukan dapat diketahui bahwa sistem yang dibangun mampu menjalankan fungsi-fungsi keamanan dengan baik.

Kata kunci: kriptografi, cloud computing, enkripsi, secret sharing

ABSTRACT

Cloud computing is a technology that used to store data online. With only internet connection, store and access data can be an easy task. However, cloud computing technology also cause some security problems. Cryptography is a study of how data can be secured. Therefore, applying cryptography in cloud computing environment is not a new thing. New algorithms and combination between them is expanded and studied to ensure security aspects such as confidentiality, integrity and availability. In this research, a system is built with applying 2 methods of cryptography, encryption and secret sharing to ensure data. Data that is ensured will be store at 2 different type of media storage namely cloud storage and laptop's internal storage. The test shows all the security functions work well.

Keywords: cryptography, cloud computing, encryption, secret sharing

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1 PENDAHULUAN

1.1 Latar belakang

Dalam beberapa tahun terakhir, teknologi *cloud computing* semakin sering digunakan. Dengan menggunakan koneksi internet, pengguna *cloud computing* akan lebih mudah dalam hal menyimpan maupun mengakses sebuah data. Akan tetapi, kemudahan yang diberikan ketika menggunakan teknologi *cloud computing* juga menimbulkan masalah dalam hal keamanan (AlJadaani, AlMaliki & AlGhamdi, 2016). Data yang disimpan pada *cloud computing* menjadi tidak aman karena *multi user* saling berbagi infrastruktur fisik yang sama pada tingkatan jaringan dan aplikasi. Hal ini memungkinkan *attacker* atau *malicious insider* mengetahui dan merusak isi sebuah data. Untuk mengatasi masalah keamanan ini maka dapat diterapkan ilmu kriptografi.

Kriptografi adalah bidang ilmu yang mempelajari bagaimana sebuah data dapat diamankan. Kriptografi terus dikembangkan untuk memenuhi aspek-aspek dari keamanan komputer seperti *confidentiality* (kerahasiaan), *integrity* (keabsahan) dan *availability* (ketersediaan) (Ariyus, 2008). Salah satu penelitian yang pernah melakukan penerapan ilmu kriptografi pada sistem berbasis *cloud computing* adalah penelitian yang berjudul "Sistem pengamanan data pada media penyimpanan *cloud* dengan metode *split data distribution*".

Penelitian tersebut melakukan proses *hashing* menggunakan algoritme SHA untuk menjamin keabsahan dari sebuah data. Nilai dari data setelah melakukan proses *hashing* selanjutnya akan dipecah menjadi 2 bagian menggunakan operasi *split*. Setelah itu, masing-masing dari hasil bagian akan disimpan pada 2 *cloud service provider* yang berbeda. Salah satu pengujian yang dilakukan pada penelitian tersebut adalah dengan menggabungkan hasil pecahan menggunakan operasi *merge*, lalu membandingkan hasilnya dengan data sebelum melakukan operasi *split*. Jika kedua data tersebut bernilai sama, maka teknik kriptografi yang digunakan telah berfungsi dengan baik. (Wibowo, 2016). Akan tetapi, dalam penelitian tersebut masih terdapat beberapa kelemahan. Pertama, metode kriptografi yang digunakan kurang sesuai untuk menyimpan data. Metode *hashing* adalah metode dari ilmu kriptografi yang bersifat *one-way function* (Ariyus, 2008). Hal ini berarti, jika data melakukan proses *hashing*, maka nilai awal dari data tidak bisa dikembalikan. Kedua, tidak adanya algoritme khusus untuk memecah data. Data juga hanya dapat dikembalikan dengan mengumpulkan semua bagian pecahan. Masalahnya, jika salah satu bagian pecahan ada yang hilang atau rusak, maka nilai awal dari data tidak bisa dikembalikan. Untuk mengatasi masalah pertama dapat diterapkan teknik enkripsi sedangkan untuk mengatasi masalah kedua dapat diterapkan teknik *secret sharing*.

Teknik enkripsi adalah salah satu metode dari ilmu kriptografi yang memungkinkan sebuah data (*data plain*) menjadi sulit untuk dibaca (*data cipher*) menggunakan sebuah kunci. Dengan ini, pihak yang tidak diinginkan akan sulit untuk membaca informasi. Sedangkan proses kebalikan enkripsi disebut dekripsi. Secara umum, teknik enkripsi dibagi berdasarkan penggunaan kuncinya menjadi algoritme kunci simetris (*symmetric key*) dan algoritme kunci asimetris (*assymetric key*). Algoritme kunci simetris adalah algoritme enkripsi dengan kunci enkripsi dan dekripsi yang sama sedangkan algoritme kunci asimetris adalah algoritme enkripsi yang memiliki kunci enkripsi dan dekripsi yang berbeda. Selain itu, pada algoritme simetris, terdapat 2 macam cara enkripsi yaitu enkripsi berdasarkan bit-bit data (*stream cipher*) dan enkripsi byte data (*block cipher*) (Ariyus, 2008). Salah satu kelompok algoritme *stream cipher* adalah Grain sedangkan salah satu kelompok algoritme *block cipher* adalah AES.

Grain adalah algoritme yang terbilang cepat dan sederhana dalam kelompok *stream cipher* (Hell, Johannson & Meier, 2007). Grain menerima masukan *key* dan *IV* (*Initial Vector*) yang masing-masing memiliki panjang 80 bit dan 64 bit untuk membangkitkan *keystream*. *Keystream* selanjutnya akan melakukan proses enkripsi dengan *data plain* sehingga menghasilkan *data cipher*. Dengan kata lain, proses dari algoritme Grain akan menghasilkan 2 nilai baru yaitu *keystream* sebagai kunci dan *data cipher* sebagai hasil enkripsi dari data.

AES atau *advanced encryption standard* adalah kelompok algoritme *block cipher* yang penggunaannya lebih efisien dibandingkan dengan algoritme *block cipher* lain (Nithya & Sripiya, 2016). AES melakukan enkripsi dengan melalui 4 proses transformasi yaitu *SubBytes*, *ShiftRows*, *MixColumn* dan *AddRoundKey* (Daemen & Rijmen, 2002).

Secret sharing adalah sebuah teknik dalam ilmu kriptografi yang memungkinkan sebuah data dapat dipecah sebanyak jumlah *share* (n) yang ditentukan (Beimel, 2007). *Secret sharing* pertama kali diperkenalkan oleh Shamir dengan skema *threshold*. Skema *threshold* memungkinkan sejumlah *share* sebanyak nilai *threshold* (k) untuk mengembalikan data awal, dimana $k > 1$ dan $n > k$ (Shamir, 1979). Dengan ini, untuk kembali mendapatkan data tidak perlu mengumpulkan semua hasil *share*.

Pada penelitian ini, penulis mengusulkan sebuah sistem yang mampu menerapkan teknik enkripsi dan *secret sharing* untuk menjamin aspek *confidentiality* pada data. *Confidentiality* adalah aspek keamanan yang menjaga informasi atau data dari pihak yang tidak diinginkan (Ariyus, 2008). Data yang diamankan akan melalui 3 proses algoritme yang berbeda. Pertama adalah algortime Grain yang akan menghasilkan nilai *keystream* dan *data cipher*. Selanjutnya, kedua data ini akan melakukan pengamanan dengan menggunakan algoritme AES dan Shamir's Secret Sharing. *Keystream* akan diamankan menggunakan algoritme AES dan nilainya akan

disimpan pada memori internal laptop sedangkan *data cipher* akan memecah datanya menjadi 3 bagian menggunakan Shamir's Secret Sharing dan menyimpan setiap nilainya pada 3 *cloud service provider* yang berbeda.

1.2 Rumusan masalah

Adapun rumusan masalah penelitian ini adalah sebagai berikut:

1. Bagaimana hasil dari penerapan teknik enkripsi dan *secret sharing*?

1.3 Tujuan

Adapun tujuan dari penelitian ini adalah sebagai berikut:

1. Melakukan implementasi dan pengujian terhadap fungsionalitas algoritma enkripsi dan *secret sharing*.

1.4 Manfaat

Manfaat dari penelitian ini adalah memberikan solusi keamanan dalam aspek *confidentiality* dengan menerapkan teknik enkripsi dan *secret sharing* serta menggunakan media penyimpanan *cloud* untuk penyimpanan data.

1.5 Batasan masalah

Adapun batasan masalah dari penelitian ini adalah sebagai berikut:

1. Aplikasi menerapkan teknik enkripsi dan *secret sharing* untuk pengamanan dan penyimpanan data.
2. Aplikasi menggunakan 3 penyedia layanan *cloud* berbeda dalam pembuatannya.

1.6 Sistematika penulisan

Bagian ini akan berisi tahap-tahap sistematis dalam penulisan laporan ilmiah yang dikerjakan. Adapun urutan-urutan penyusunan yang dilakukan adalah sebagai berikut:

BAB 1 PENDAHULUAN

Pada bab ini akan dijelaskan suatu permasalahan yang menjadi landasan dari penelitian. Isi dari bab ini antara lain latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini akan berisi teori-teori ilmiah yang akan membantu penulis dalam mengembangkan penelitiannya. Isi yang direncanakan dalam bab ini terdiri dari teori dasar kriptografi, teknik enkripsi dan *secret sharing* yang meliputi Grain, AES, dan Shamir's Secret Sharing, media penyimpanan *cloud,* serta *tools,* bahasa pemrograman dan aplikasi yang digunakan dalam pembuatan aplikasi.

BAB 3 METODOLOGI PENELITIAN

Bab ini akan menjelaskan tahapan-tahapan yang dilalui penulis dalam melakukan penelitian. Bab ini berisi studi literatur, perancangan sistem, implementasi, pengujan serta kesimpulan.

BAB 4 PERANCANGAN

Bab ini akan menjelaskan tahap perancangan dari sistem yang akan dibangun. Bab ini berisi 2 sub-bab yaitu lingkungan sistem yang menjelaskan kebutuhan fungsional sistem dan perancangan aplikasi yang menjabarkan alur sistem yang dibuat.

BAB 5 IMPLEMENTASI

Bab ini akan berisi hasil implementasi dari rancangan yang telah dijabarkan sebelumnya. Implementasi yang dilakukan adalah menerapkan teknik enkripsi (Grain dan AES) dan *secret sharing* (Shamir's Secret Sharing) untuk mengamankan data serta menggunakan fitur *cloud computing* dan memori internal laptop sebagai media penyimpanan.

BAB 6 PENGUJIAN

Bab ini akan menjelaskan skenario pengujian dan hasil pengujian yang dilakukan. Pengujian yang dilakukan adalah menguji hasil *test vector* untuk menguji validitas *keystream*, hasil proses enkripsi, dekripsi, *split* dan *reconstruct* dalam mengamankan dan mengembalikan data serta menguji fungsi-fungsi pada media penyimpanan *cloud computing* yang diterapkan pada sistem.

BAB 7 PENUTUP

Bab ini akan berisi kesimpulan yang merupakan rangkuman proses dan hasil dari keseluruhan penelitian dan saran yang berguna untuk pengembangan dan perbaikan untuk sistem yang ada.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Penelitian ini menerapkan ilmu kriptografi untuk mengamankan data dan arsitektur *cloud computing* untuk penyimpanan data. Penelitian yang berjudul "Sistem pengamanan data pada media penyimpanan *cloud* dengan metode *split data distribution*" juga menerapkan ilmu kriptografi dan arsitektur *cloud computing*. Penelitian tersebut melakukan proses *hashing* menggunakan algoritme SHA untuk menjamin keabsahan dari sebuah data. Nilai dari data setelah melakukan proses *hashing* selanjutnya akan dipecah menjadi 2 bagian menggunakan operasi *split*. Setelah itu, masing-masing dari hasil bagian akan disimpan pada 2 *cloud service provider* yang berbeda. Salah satu pengujian yang dilakukan pada penelitian tersebut adalah dengan menggabungkan hasil pecahan menggunakan operasi *merge*, lalu membandingkan hasilnya dengan data sebelum melakukan operasi *split*. Jika kedua data tersebut bernilai sama, maka teknik kriptografi yang digunakan telah berfungsi dengan baik. (Wibowo, 2016). Akan tetapi, dalam penelitian tersebut masih terdapat beberapa kelemahan. Pertama, metode kriptografi yang digunakan kurang sesuai untuk menyimpan data. Metode *hashing* adalah metode dari ilmu kriptografi yang bersifat *one-way function* (Ariyus, 2008). Hal ini berarti, jika data melakukan proses *hashing*, maka nilai awal dari data tidak bisa dikembalikan. Kedua, tidak adanya algoritme khusus untuk memecah data. Data juga hanya dapat dikembalikan dengan mengumpulkan semua bagian pecahan. Masalahnya, jika salah satu bagian pecahan ada yang hilang atau rusak, maka nilai awal dari data tidak bisa dikembalikan. Untuk mengatasi masalah pertama dapat diterapkan teknik enkripsi sedangkan untuk mengatasi masalah kedua dapat diterapkan algoritme *secret sharing*.

Ada 2 algoritme enkripsi yang diterapkan pada penelitian ini yaitu Grain dan AES. Algoritme Grain dipilih sebagai algoritme pertama karena Grain merupakan algoritme yang terbilang cepat dan sederhana dalam kelompok *stream cipher* (Hell, Johannson & Meier, 2007). Grain menerima masukan *key* dan *IV* (*Initial Vector*) yang masing-masing memiliki panjang 80 bit dan 64 bit untuk membangkitkan *keystream*. *Keystream* selanjutnya akan melakukan proses enkripsi dengan *data plain* sehingga menghasilkan *data cipher*

AES dipilih sebagai algoritme kedua karena AES merupakan algoritme *block cipher* yang lebih baik dalam masalah waktu, kecepatan dan penggunaan memori dibandingkan dengan beberapa algoritme lain (Nithya & Sripiya, 2016). AES melakukan enkripsi dengan melalui 4 proses transformasi yaitu *SubBytes*, *ShiftRows*, *MixColumn* dan *AddRoundKey* (Daemen & Rijmen, 2002).

Secret sharing adalah algoritme yang mampu memecah data sebanyak jumlah *share* (n) yang ditentukan (Beimel, 2007). *Secret sharing* pertama kali diperkenalkan oleh Shamir dengan skema *threshold*. Skema *threshold* memungkinkan sejumlah *share* sebanyak nilai *threshold* (k) untuk mengembalikan data awal, dimana $k > 1$ dan $n > k$ (Shamir, 1979). Dengan ini, untuk kembali mendapatkan data tidak perlu mengumpulkan semua hasil *share*.

2.2 Dasar Teori

2.2.1 Kriptografi

Kriptografi berasal dari Bahasa Yunani, yaitu *crypto* yang berarti menyembunyikan dan *graphia* yang berarti tulisan. Sehingga kriptografi bisa diartikan sebagai ilmu yang mempelajari bagaimana sebuah pesan/data dapat disembunyikan (Ariyus, 2008). Di era modern, kriptografi berkembang seiring dengan perkembangan komputer. Hal ini tidak lepas karena penggunaan kriptografi sekarang banyak di implementasikan dalam komputer untuk mengamankan data maupun komunikasi. Ada beberapa aspek yang dibutuhkan pada keamanan komputer (Ariyus, 2008) yaitu:

1. *Confidentiality*
   Confidentiality atau aspek kerahasiaan menjamin sebuah pesan/data atau informasi tidak diketahui oleh pihak yang tidak diinginkan dan hanya diketahui oleh pihak tertentu.
2. *Integrity*
   Integrity atau aspek integritas menjamin keutuhan informasi. Informasi harus berasal dari pihak yang benar dan tidak dimodifikasi oleh pihak lain. Contoh dari pengamanan ini adalah dengan menerapkan teknik *hash*.
3. *Authentication*
   Authentication adalah proses validasi yang hanya mengijinkan pihak tertentu untuk dapat mengakses informasi. *Authentication* bisa diterapkan dengan menggunakan *password* atau teknik *digital signature*.
4. *Availability*
   Availabiity adalah aspek yang menjamin ketersediaan sebuah informasi. Informasi harus bisa diakses bila dibutuhkan.
5. *Non-Repudiation*
   Non-Repudiation adalah aspek yang berhubungan dengan pengirim. Aspek ini membuat pengirim tidak bisa mengelak bahwa dialah yang mengirim informasi tersebut.
6. *Privacy*
   Privacy merupakan aspek yang mementingkan hal-hal yang bersifat pribadi.

7. *Authority*
   Authority mengatur hak akses dari pihak-pihak yang dapat mengakses informasi.
8. *Access Control*
   Access Control mengatur masalah akses informasi. Hal ini biasanya berhubungan dengan masalah autentikasi dan *privacy*.

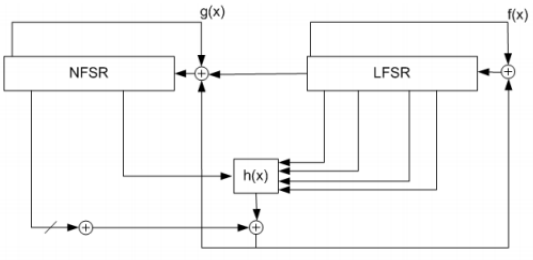
Dalam bidang ilmu kriptografi, dikenal dua istilah yaitu enkripsi dan dekripsi. Enkripsi merupakan proses pengacakan naskah asli (*plain*) menjadi naskah acak yang sulit dibaca (*cipher*) sedangkan dekripsi adalah proses pengembalian *cipher* ke dalam *plain*. Kedua proses ini membutuhkan satu komponen yang disebut kunci. Kunci dan *plain* akan dikombinasikan dengan serangkaian rumus matematika dalam algoritme enkripsi sehingga menghasilkan *cipher*. Kunci yang digunakan untuk proses dekripsi bisa merupakan kunci yang sama saat proses enkripsi (kunci simetris) maupun kunci yang berbeda saat proses enkripsi (kunci asimetris) (Ariyus, 2008). Berdasarkan cara melakukan enkripsi, algoritme enkripsi dibagi menjadi algoritme yang melakukan enkripsi dengan bit-bit data (*stream cipher*) dan algoritme yang melakukan enkripsi dengan blok-blok data (*block cipher*) (Kurniawan, 2004).

Gambar 2.1 Proses enkripsi dan dekripsi

Sumber: (Kromodimoeljo, 2009)

2.2.1.1 Grain

Grain adalah salah satu algoritme kelompok *stream cipher* yang cepat dan sederhana (Hell, Johannson, & Meier, 2007). Grain memiliki 3 versi yaitu Grain v0, v1 dan v128. Grain v1 menerima masukan *key* dan *initial vector* (IV) yang masing-masing memiliki panjang 80 bit dan 64 bit. Desain algoritme ini menggunakan 2 *shift register* yaitu *linear feedback shift register* (LFSR) dan *non-linear feedback shift register* (NFSR) serta sebuah fungsi filter.



Gambar 2.2 Inisialisasi *keystream*

Sumber: (Hell, Johannson, & Meier, 2007)

Untuk mendapatkan *keystream*, ada 2 proses utama yang dilakukan pada algoritme ini. Proses pertama adalah inisialisasi dan proses kedua adalah *generate*. Gambar 2.2 menunjukkan bagaimana Inisialisasi Grain *cipher* bekerja.

LFSR merupakan *state* (kondisi) yang didapat dari inisialisasi IV sepanjang 64bit dan 16bit yang bernilai 1. Isi LFSR dapat dilambangkan dengan $s_i, s_{i+1} \dots, s_{i+79}$. LFSR akan menjalankan fungsi umpan balik $f(x)$ yang merupakan fungsi polynomial berderajat 80. Adapun fungsi $f(x)$ dapat didefinisikan menggunakan Persamaan 2.1:

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80} \quad (2.1)$$

Untuk menghilangkan kemungkinan nilai yang bertumpuk karena melakukan perulangan sebanyak 160 kali, fungsi *update* dari LFSR dapat dinotasikan menggunakan Persamaan 2.2:

$$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i \quad (2.2)$$

NFSR merupakan *state* yang didapat dari inisialisasi *key* sepanjang 80 bit. Isi NFSR dilambangkan dengan $b_i, b_{i+1} \dots, b_{i+79}$. NFSR akan menjalankan fungsi umpan balik $g(x)$. Fungsi ini adalah fungsi *polynomial* berderajat 80. Adapun fungsi $g(x)$ dapat didefinisikan menggunakan Persamaan 2.3:

$$g(x) = 1 + x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59} \quad (2.3)$$

Untuk menghilangkan kemungkinan nilai yang bertumpuk karena melakukan perulangan sebanyak 160 kali, fungsi *update* NFSR dapat dinotasikan menggunakan Persamaan 2.4:

$$b_{i+80} = s_i + b_{i+63} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + b_{i+15} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + b_{i+60}b_{i+52}b_{i+45} + b_{i+38}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{1+9} + b_{i+60}b_{i+52}b_{i+37}b_{1+33} + b_{i+63}b_{i+60}b_{i+21}b_{1+15} + b_{i+63}b_{i+60}b_{i+52}b_{1+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{1+15}b_{i+9} + b_{i+52}b_{i+45}b_{i+37}b_{1+35}b_{i+28}b_{i+21} \quad (2.4)$$

Fungsi *filter* akan menerima masukan dari *state* yang ada pada LFSR dan NFSR. Lima variabel akan diambil sebagai masukan dari *boolean function*, $h(x)$. Variabel tersebut dilambangkan dengan x_0, x_1, x_2, x_3, x_4 dimana nilai tersebut masing-masing merupakan nilai dari posisi $s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}$. Adapun fungsinya dapat didefinisikan menggunakan Persamaan 2.5:

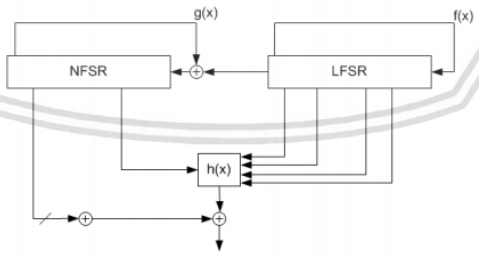
$$h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4 \quad (2.5)$$

Keluaran dari nilai $h(x)$ selanjutnya akan melakukan proses xor terhadap nilai zi. Adapun fungsi zi dapat dinotasikan menggunkan Persamaan 2.6:

$$zi = \sum_{k\epsilon A}^{n} b + h(x) \quad (2.6)$$

$$Dimana\ A = \{1,2,4,10,31,43,56\}$$

Setelah itu, nilai ini akan melakukan operasi xor dengan $f(x)$ dan $g(x)$ secara terpisah. Kedua nilai yang dihasilkan masing-masing akan menggantikan nilai pada indeks terakhir LFSR dan NFSR setelah melakukan penggeseran pada *state* LFSR dan NFSR searah jarum jam sebanyak satu kali. Proses ini akan di-*clock* sebanyak 160 kali sehingga masing-masing *state* dari LFSR dan NFSR akan memiliki *state* baru yang disebut *initial state*.



Gambar 2.3 *Generate keystream*

Sumber: (Hell, Johannson, & Meier, 2007)

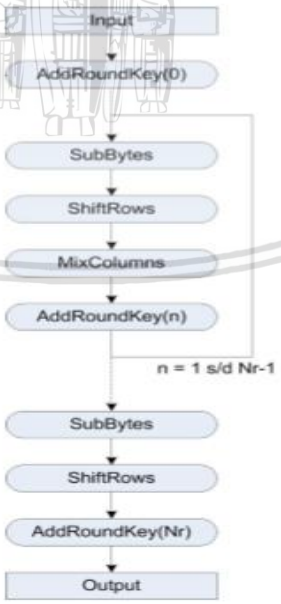
Proses *generate keystream* tidak memiliki banyak perbedaan dari proses inisialisasi. Gambar 2.3 menggambarkan bagaimana proses *generate* bekerja. *State* dari LFSR dan NFSR merupakan *initial state* yang didapat dari proses inisialisasi. Selanjutnya, proses mendapatkan nilai $f(x)$, $g(x)$ dan $h(x)$ sama seperti proses

inisialisasi. Penggeseran indeks tetap dilakukan namun kali ini nilai pada indeks terakhir dari LFSR dan NFSR masing-masing diganti dengan nilai $f(x)$dan $g(x)$. Selain itu, perbedaan *generate* dan inisialisasi terdapat pada saat nilai $h(x)$ yang melakukan proses xor dengan nilai zi. Keluaran dari nilai tersebut akan dijadikan bit-bit dari *keystream* dan proses ini diulang sebanyak jumlah bit yang terdapat pada *plain text* yang ingin diamankan.

Setelah mendapatkan nilai *keystream*, *plain text* akan melakukan operasi xor dengan *keystream* dan menghasilkan *cipher text*. Proses dekripsi akan menghasilkan *plain text* setelah melakukan operasi xor antara *cipher text* dan *keystream*.

2.2.1.2 AES

AES atau *Advanced Encryption Standard* merupakan algoritme *block cipher* dari proses seleksi yang diadakan NIST (*National Institute of Standard and Technology*) ketika algoritme DES sudah tidak dikatakan aman lagi. Dari beberapa teknik yang diajukan, algoritme yang diciptakan Vincent Rijmen dan Joan Daemen terpilih sebagai pemenangnya karena memiliki keseimbangan antara keamanan yang tinggi serta fleksibilitas dalam berbagai macam *platform* (Ariyus, 2008). Dibandingkan DES yang saat itu menggunakan kunci dengan panjang 56 bit, AES memiliki 3 tipe kunci yang lebih besar yaitu, 128 bit, 196 bit dan 256 bit. Algoritme AES juga memiliki 4 fungsi tranformasi yaitu *SubBytes, ShiftRows, MixColumn* dan *AddRoundKey* (Daemen & Rijmen, 2002). Adapun proses enkripsi dari algoritme AES dapat dilihat pada Gambar 2.4:



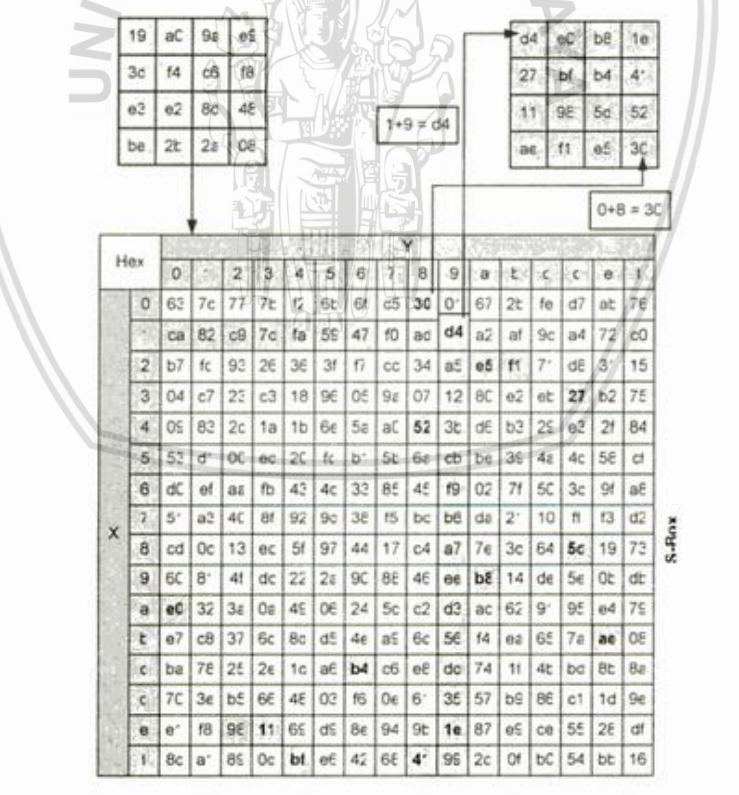
Gambar 2.4 Enkripsi AES

Sumber: (Kromodimoeljo, 2009)

Gambar 2.4 memperlihatkan proses enkripsi dari AES. Proses pertama yang dilakukan adalah transformasi *AddRoundKey* yang melakukan operasi xor antara *state* (matriks $4x4$ dari *byte plain text*) dengan kunci (matriks $4x4$ dari *byte* kunci).

Proses selanjutnya adalah transformasi *SubBytes, ShiftRows, MixColumn* dan *AddRoundKey* yang dilakukan sebanyak $Nr - 1$. Nr adalah jumlah putaran yang dilakukan pada proses enkripsi AES. Jumlah putaran yang dilakukan tergantung pada panjangnya kunci yang digunakan. 128 bit melakukan 10 kali putaran, 192 bit melakukan 12 putaran dan 256 bit melakukan 14 kali putaran (Daemen & Rijmen, 2002).

SubBytes merupakan proses yang mengganti nilai-nilai (subtitusi) pada setiap *byte* menggunakan kotak S-Box (Daemen & Rijmen, 2002). Kotak S-Box merupakan matriks $16x16$ yang berisikan 2 nilai heksadesimal pada setiap kotaknya. Cara mengganti nilai-nilai pada elemen *state* dengan nilai pada kotak S-Box adalah dengan merepresentasikan nilai sebelah kiri dan nilai sebelah kanan pada *state* sesuai dengan indeks x dan indeks y pada kotak S-Box. Adapun proses *SubBytes* dapat dilihat pada Gambar 2.5:

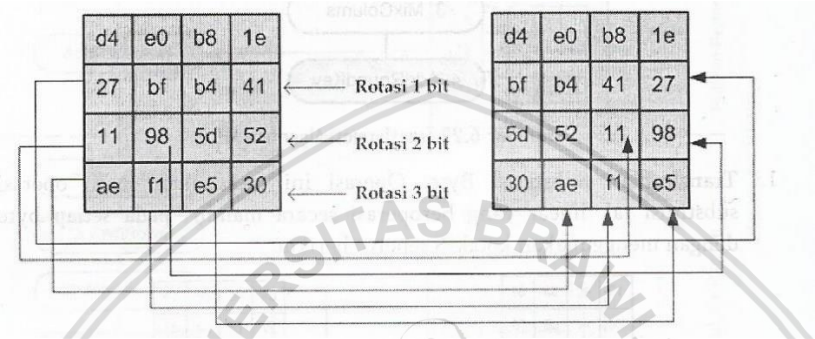


Gambar 2.5 Proses *SubBytes*

Sumber: (Ariyus, 2008)

Transformasi *ShiftRows* merupakan proses penggeseran baris terhadap *state* ke kiri. Adapun jumlah baris yang digeser adalah sebagai berikut (Daemen & Rijmen, 2002):

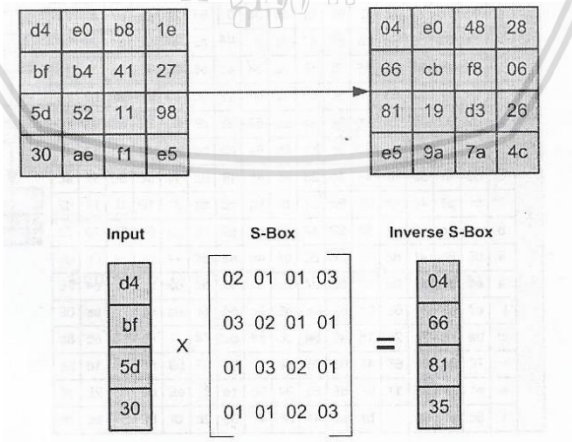
1. Baris 0 tidak digeser
2. Baris 1 menggeser sebanyak 1 kali
3. Baris 2 menggeser sebanyak 2 kali
4. Baris 3 menggeser sebanyak 3 kali



Gambar 2.6 Proses *ShiftRows*

Sumber: (Ariyus, 2008)

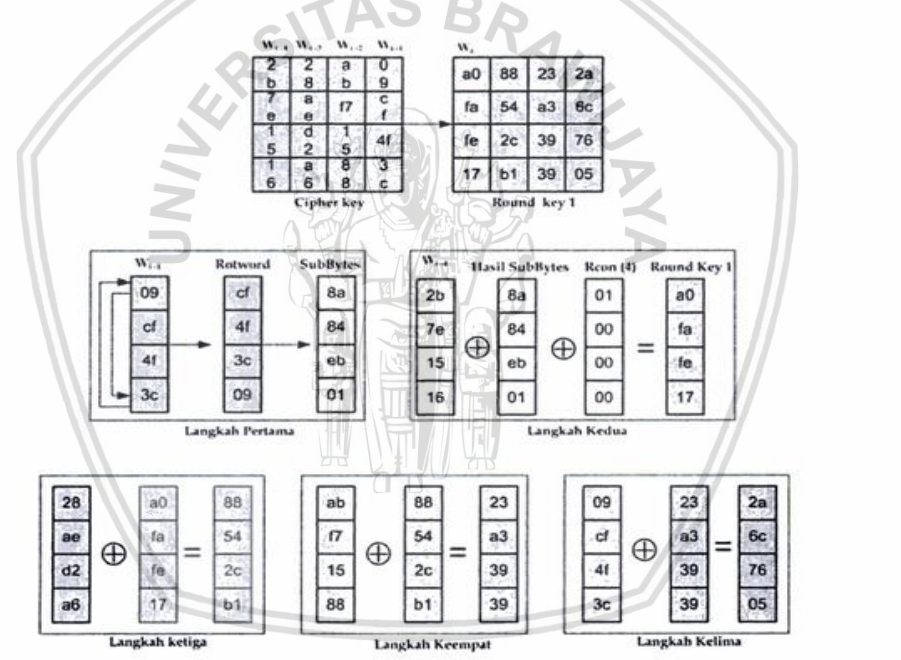
Sedangkan *MixColumn* memperlakukan kolom *state* sebagai *polynomial* dengan koefisien dalam $GF\,(2^8)$ (Daemen & Rijmen, 2002). Transformasi *MixClumn* mengkalikan setiap kolom pada *state* dengan *polynomial* $a(x)$. Adapun proses dari *MixColumn* dapat dilihat pada Gambar 2.7:



Gambar 2.7 Proses *MixColumn*

Sumber: (Ariyus, 2008)

AddRoundKey pada tiap putaran dalam proses ini menggunakan kunci yang berbeda-beda. Teknik ini disebut dengan ekspansi kunci (Ariyus, 2008). Ekspansi kunci mengambil nilai dari kolom terakhir *state* kunci untuk melakukan *rot word*, yaitu proses menggeser nilai kearah atas sebanyak satu kali. Setelah itu, hasil dari *rot word* akan di subtitusi dengan *S-Box* menggunakan transformasi *SubBytes* (lihat langkah pertama pada Gambar 2.8). Hasil dari trasnformasi ini akan melakukan operasi xor dengan kolom pertama dari matriks kunci pertama dan sebuah *round constant* (lihat langkah 2 pada Gambar 2.8). *Round constant* sendiri merupakan suatu komponen dari putaran tetap larik kata dalam perhitungan kunci ekspansi *routine* (Ariyus, 2008). Operasi xor ini akan menghasilkan kolom pertama dari matriks kunci yang baru. Untuk mendapatkan 3 kolom selanjutnya, maka kolom baru yang dihasilkan sebelumnya akan melakukan operasi xor dengan kolom yang sesuai urutannya. (lihat langkah ketiga, keempat dan kelima pada Gambar 2.8)



Gambar 2.8 Proses ekspansi kunci

Sumber: (Ariyus, 2008)

Setelah melakukan proses tersebut sebanyak Nr-1, proses selanjutnya mencakup transformasi *SubBytes, ShiftRows* dan *AddRoundKey* sebanyak satu kali yang akan menghasilkan *cipher text*. Adapun proses dekripsi memiliki tahap-tahap yang merupakan kebalikan dari proses enkripsi dan memiliki proses transformasi yang berbeda dengan enkripsi. Transformasi *SubBytes, ShiftRows* dan *MixColumn* pada

enkripsi diganti menjadi proses transformasi *invers*-nya yaitu *InverseSubBytes, InverseShiftRows* dan *InverseMixColumn* pada proses dekripsi.

2.2.1.3 Shamir's Secret Sharing

Shamir's Secret Sharing merupakan teknik kriptografi yang dapat memecah data (D)dalam beberapa bagian (n) menjadi $D_i \ \dots, D_n$. Untuk mendapatkan kembali nilai D_i maka dibutuhkan kumpulan nilai D_i sebanyak k, dimana $k > 1$ dan $n \geq k$. Dengan kata lain, data akan dengan lebih mudah kembali didapatkan jika memiliki D_i sebanyak k atau lebih dan pihak yang tidak diinginkan akan sulit mendapatkan data jika hanya memiliki D_i sebanyak $k - 1$ atau kurang (Shamir, 1979).

Algoritme ini menggunakan notasi (k, n) dalam memecah data. Skema ini biasa disebut dengan *threshold scheme*. Cara yang dilakukan untuk memecah data (*split*) adalah dengan memilih bilangan bulat acak sebanyak $k - 1$. Bilangan bulat ini masing-masing akan dilambangkan dengan $a_1, a_2 \dots, a_{k-1}$. Bilangan bulat serta data akan dinyatakan dalam bentuk fungsi polynomial $f(x)$ berderajat $k - 1$. Untuk mendapat nilai *share* yang ideal, maka algoritme ini menggunakan aritmatika modular dengan nilai yang dilambangkan p. Nilai p adalah bilangan priama acak yang dipilih, dimana $p > D$ (Shamir, 1979). Fungsi ini dapat dinotasikan menggunakan persamaan 2.7:

$$f(x) = D + \ a_1 x + \ a_2 x^2 \dots . + a_{k-1} x^{k-1} \ mod \ p \qquad (2.7)$$

Fungsi ini akan dijalankan untuk mendapatkan hasil *share* dari data yang ingin dipecah. Setiap nilai *share* didefinisikan dengan bentuk $(i, f(i))$ dimana nilai $i = 1, 2 \dots, n$ dan $f(i)$adalah nilai masing-masing dari *share* yang diciptakan dari fungsi $f(x)$.

Proses penggabungan (*reconstruct*) dari beberapa kumpulan *share* menggunakan *polynomial lagrange*. Proses *reconstruct* bekerja dengan mendefinisikan bentuk (x_i, y_i) yang didapat dari nilai $(i, f(i))$. Untuk mendapatkan nilai D pada persamaan 2.7, maka nilai x dalam $f(x)$ harus bernilai 0 menggunakan persamaan sebagai berikut (Beimel, 2007):

$$f(0) = D = \sum_{i=0}^{k} yi \ \prod_{1 \leq j \leq k, i \neq j} \frac{x_j}{x_j - x_i} \qquad (2.8)$$

Sebagai contoh, sebuah data bernilai 27 ($D = 27$) akan diamankan. Data akan dipecah menjadi 5 bagian ($n = 4$) dengan nilai *threshold* sama dengan 3 ($k = 3$). Selanjutnya adalah memilih bilangan bulat acak sebanyak $k - 1$ dan bilangan prima. Bilangan bulat yang akan digunakan adalah 2 dan 5 sedangkan bilangan prima yang dipilih adalah 31 ($a_1 = 2, \ a_2 = 5, \ p = 31$). Persamaan yang terbentuk dari nilai-nilai ini adalah:

$$f(x) = 27 + 2x + 5x^2 \ mod\ 31 \qquad (2.9)$$

Dari persamaan ini maka akan menghasilkan 4 nilai yaitu $D_1 = 3$, $D_2 = 10$, $D_3 = 16$, $D_4 = 22$. Berdasarkan nilai *threshold*, maka proses *reconstruct* akan menggunakan 3 nilai dari hasil *share* yang telah terbentuk. Hasil *share* yang dipilih dalam kasus ini akan dinotasikan dalam (x_i, y_i). Nilai tersebut adalah $(x_0, y_0) = (1,3)$, $(x_1, y_1) = (2,10)$ dan $(x_2, y_2) = (3,16)$. Langkah selanjutnya adalah menjalankan persamaan 2.8 dengan nilai-nilai yang sudah ditentukan. Persamaan tersebut dapat dilihat pada Persamaan 2.10:

$$D = \begin{pmatrix} 3 \times \frac{2\times3}{(2-1)(3-1)} + 20 \times \frac{1\times3}{(1-2)(3-2)} \\ + \\ 16 \times \frac{1\times2}{(1-3)(2-3)} \end{pmatrix} mod\ 31 = 27 \qquad (2.10)$$

Pada Persamaan 2.10, perhitungan yang dilakukan berdasarkan nilai yang dimasukkan akan menghasilkan nilai 27. Nilai 27 ini merupakan data awal yang sebelumnya dipecah menggunakan algoritme Shamir's Secret Sharing.

2.2.2 Cloud computing

Cloud computing adalah sebuah konsep dari pemakaian komputer yang dapat dijalankan serta dikembangkan melalui internet. Istilah *cloud* dalam teknologi ini dilambangkan sebagai awan yang merupakan representase dari internet itu sendiri (Waloeyo, 2012). *Cloud computing* sudah sering digunakan dalam berbagai macam kondisi, mulai dari pemakaian secara pribadi, untuk keperluan bisnis dan telah masuk dalam lingkup pemerintahan. Berdasarkan arsitekturnya, *cloud* dibagi menjadi 2 macam, yaitu *front-end* dan *back-end* (Waloeyo, 2012). *Front-end* merupakan bagian langsung yang berhubungan dengan pengguna (*client*) dan hal-hal yang menghubungkan *client* dengan *cloud* sedangkan *back-end* adalah bagian yang berisi *servers* hingga *data storage* yang membentuk layanan *cloud*.

Ada beberapa keuntungan yang dapat dihasilkan dalam penggunaan teknologi *cloud computing* (Waloeyo, 2012),diantaranya:

1. *Reduced cost*
   Biaya dalam penggunaan *cloud computing* terbilang lebih murah karena sumber daya yang digunakan sudah disediakan pada penyedia layanan.
2. *Increased storage*
   Salah satu sumber daya yang disediakan pada *cloud* adalah media penyimpanan. Memori yang disediakan dari *cloud* terdiri dari yang terbatas (*free*) maupun yang berbayar (*premium*).

3. *Highly automated*
   Layanan *cloud* yang digunakan *client* hanya dapat mengakses fitur-fitur umum. Sehingga *client* tidak perlu melakukan *update* terhadap sistem.
4. *Flexibility*
   Cloud computing dengan mudah dapat berorientasi pada profit dan perkembangan yang cepat berubah.
5. *More mobility*
   Client dapat dengan mudah menggunakan fitur *cloud* kapan pun dan dimana pun selama *client* masih terkoneksi jaringan internet

Bedasarkan jenis layanannya, *cloud computing* dapat dibagi menjadi 3 macam (Waloeyo, 2012) yaitu:

1. *Software as Service* (SaaS)
   SaaS merupakan jenis layanan yang paling dahulu populer digunakan. SaaS juga bisa didefinisikan dengan aplikasi (*software*) berbasis web yang disediakan pada lingkungan *cloud computing*.
2. *Platform as a Service* (PaaS)
   Jenis layanan ini akan menyediakan modul-modul yang dapat membantu *client* untuk membangun dan mengembangkan aplikasi berbasis *cloud* tanpa memikirkan perawatan *computing platform*. PaaS berada satu level di bawah SaaS dan masih tidak memiliki kendali dalam hal memori, media penyimpanan hingga *processing power*.
3. *Infrastructure as a Service* (IaaS)
   IaaS menyediakan layanan yang menyewakan infrastruktur dari *cloud* seperti memori, *processing power*, kapasitas jaringan, media penyimpanan hingga sistem operasi.

2.2.2.1 Cloud storage

Salah satu layanan yang disediakan oleh teknologi *cloud computing* adalah media penyimpanan. Media penyimpanan yang selanjutnya disebut *cloud storage* ini dapat diakses oleh *client* dengan mudah selama masih terkoneksi dengan jaringan internet. *Cloud storage* memiliki model penyimpanan *online* yang dijalankan secara *virtual* dan dikelola oleh pihak ketiga (Waloeyo, 2012). Pihak ketiga ini merupakan sebuah perusahan *hosting* yang menyediakan berbagai macam fitur yang dapat digunakan oleh *client* berdasarkan jenis layanan yang dipilih. Perusahaan yang juga disebut *cloud storage provider* ini akan mengoperasikan data sentral dan menyimpan data-data *client* di dalam *server* yang terus dirawat secara terus menerus secara otomatis maupun manual.

2.2.2.2 Dropbox

Dropbox adalah salah satu layanan penyedia jasa *cloud storage* paling sering digunakan. Dropbox menyediakan 2 GB memori penyimpanan *online* secara gratis dan ruang sebanyak yang dibutuhkan dalam penyimpanan untuk layanan yang berbayar. Dropbox menyimpan data dari *user* pada Amazon's Simple Storage Service (S3) serta mendapatkan pengamanan melalui Secure Socket Layer (SSL) dan Advanced Encryption Standard (Rouse, 2011).

2.2.2.3 Google Drive

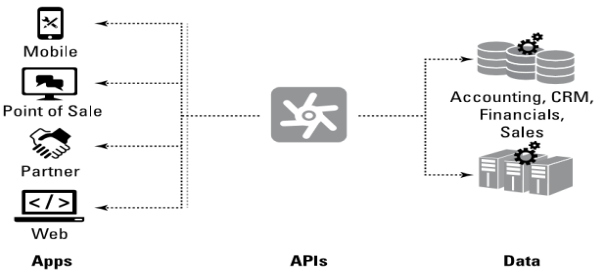
Google Drive adalah layanan penyimpanan data online yang dimiliki oleh Google. Google Drive terintegrasi langsung dengan berbagai macam layanan pada Google seperti, Gmail, Android, Google Analytics hingga Chrome (Rouse, 2011). Google Drive menyediakan 15 GB layanan penyimpanan secara gratis dan lebih dari 30 TB memori penyimpanan untuk layanan berbayar.

2.2.2.4 Box

Box adalah salah satu layanan penyedia *cloud computing* yang berdiri sejak tahun 2005 (Box,2017). Box juga terintegrasi langsung dengan Windows Explorer dan Mac Finder. Box menyediakan 10 GB memori penyimpanan untuk layanan yang tidak berbayar.

2.2.3 API

API merupakan singkatan dari *Application Programming Interface*. Menurut (Nijim & Pagano, 2014), API adalah sebuah *tool* yang dapat menghubungkan bisnis proses, layanan, konten dan data dari beberapa *developers* dengan mudah dan aman. Sebagai contoh, sebuah aplikasi berbasis *mobile* menggunakan akun Facebook untuk mengakses sistem. Hal ini berarti, aplikasi yang dibangun menggunakan API dari Facebook. Sedangkan secara teknis, API adalah cara bagaimana 2 aplikasi dapat bertukar data, informasi dan layanan melalui internet tanpa harus menyesuaikan bahasa pemrograman yang dimiliki masing-masing (Jacobson, Brail, & Woods, 2012). Ketika 2 aplikasi dengan struktur dan layanan yang berbeda ingin berkolaborasi, maka dengan menggunakan API, kedua aplikasi ini akan terintegrasi satu sama lain. Adapun arsitektur dari API dapat dilihat pada Gambar 2.9:



Gambar 2.9 Arsitektur API

Sumber: (Nijim & Pagano, 2014)

2.2.3.1 CloudRail

CloudRail merupakan sebuah layanan yang dapat mengitegrasikan banyak API dengan satu template saja. Hal ini dilakukan dengan menggabungkan beberapa API dalam satu SDK. Jenis API yang dapat dihubungkan CloudRail dengan berbagai aplikasi adalah *cloud storage, location services, messaging services, payment providers* dan *social networks*. Sebagai contoh, sebuah aplikasi yang menggunakan Dropbox dan Google Drive untuk meng-*upload* file. Dengan ini, fungsi *upload()* pada Dropbox maupun Google Drive memliki cara yang sama. CloudRail juga memiliki keuntungan dalam hal menangani API yang selalu berubah dengan hanya melakukan *update* pada *library* CloudRail ke dalam versi yang paling baru. Selain itu, data pada aplikasi yang terintegrasi dengan CloudRail tidak akan melewati server dari CloudRail karena semua transformasi data terjadi pada CloudRail SDK sehingga *developer* tidak perlu khawatir masalah kerahasiaan data (CloudRail, 2017).

2.2.4 Java

Menurut (Schildt, 2007), inovasi yang diciptakan dalam bahasa pemrograman mengacu pada 2 hal mendasar, yaitu dapat beradaptasi pada lingkungan dan pemakaian yang sering berubah serta mampu memberikan perbaikan dan kemajuan dalam dunia pemrograman. Java adalah Bahasa pemrograman yang sintaksnya diturunkan dari bahasa pemrograman C dan konsep *object oriented*-nya dipengaruhi dari bahasa pemrograman C++. Konsep *object oriented* pada Java memungkinkan program yang dibangun memiliki cara kerja seperti cara manusia menggunakan bahasa sehari-hari. Karenanya, Java termasuk bahasa pemrograman yang terbilang sederhana dan mudah dipahami.

Java dibuat oleh James Ghosling, Patrick Naughton dan Mike Sheridan dari Sun Microsystems pada tahun 1991. Bahasa pemrograman Java kini terus berkembang dan menjadi salah satu bahasa pemrograman paling populer dan sering digunakan.

Selain itu, Java juga mampu dikembangkan melalui berbagai macam *platform* seperti Windows, Linux maupun Mac.

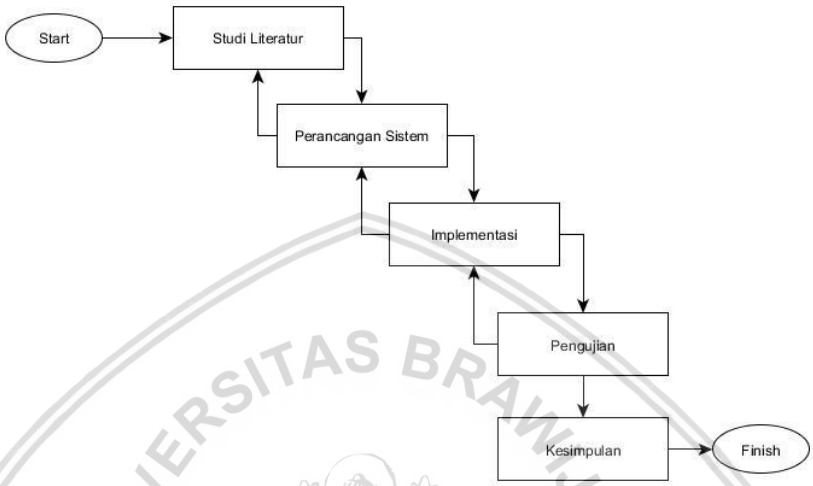
2.2.5 Maven

Maven adalah sebuah *tool* yang berfungsi untuk mengatur *project management* berdasarkan *project object model* (POM). Maven mampu mempermudah *developer* dalam mengatur struktur folder, *library* dan *framework* yang digunakan dalam membangun aplikasi. Dalam aplikasi yang dibangun menggunakan Maven, maka di dalam *project* akan terdapat file *pom.xml.* File ini berfungsi mendownload repository Maven yang berisi *library* maupun *framework* yang akan digunakan dalam *project*. Menurut (Sonatype, 2008), Maven memiliki 5 kelebihan di dalamnya, yaitu:

1. Mengatur Dependencies
2. Mengatur struktur folder pada IDE berbeda
3. Mempermudah pencarian *projects artifacts*
4. Penggunaan *build logic* ulang yang umum
5. *Repositories* yang di-*remote*

BAB 3 METODOLOGI

Bab ini akan menjelaskan langkah-langkah dalam penelitian yang dilakukan. Secara garis besar, tahapan penelitian dapat dilihat pada Gambar 3.10:



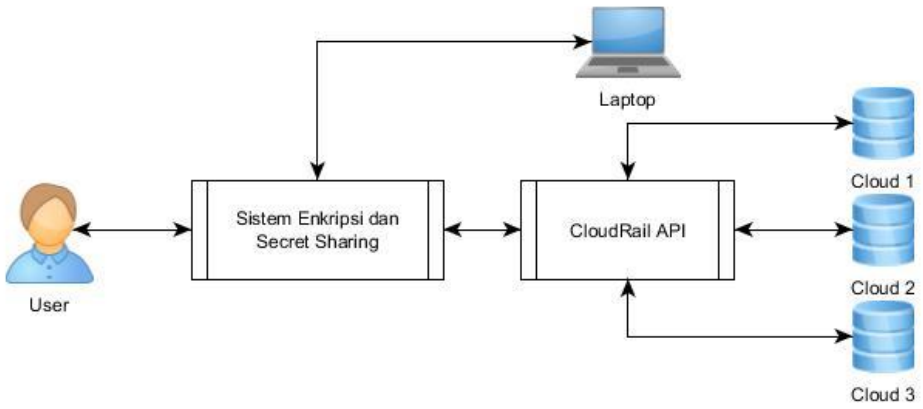
Gambar 3.10 Diagram alir penelitian

3.1 Studi literatur

Dalam pengerjaan penelitian, pemahaman mendalam akan konsep-konsep pada bidang ilmu terkait akan sangat dibutuhkan. Teori utama yang perlu dipelajari lebih dalam adalah kriptografi dan *cloud computing*. Selain itu, pengetahuan akan penelitian sebelumnya yang membahas kriptografi maupun *cloud computing* akan dikaji secara komprehensif. Hal ini bisa ditempuh dengan memperbanyak bacaan melalui jurnal, buku dan website resmi yang membahas topik seputar penelitian.

3.2 Perancangan sistem

Penelitian ini bertujuan untuk membangun sistem keamanan dengan menerapkan algoritme enkripsi dan *secret sharing* serta memanfaatkan fitur dari *cloud storage*. Tipe dari penelitian ini adalah implementatif. Tahap perancangan juga menjadi acuan yang digunakan ketika akan melakukan tahap implementasi. Tujuan dari tahap ini adalah untuk memberikan gambaran sistem yang akan dikembangkan. Pada tahap ini akan dijelaskan secara rinci kebutuhan-kebutuhan serta alur kerja pada setiap proses yang berjalan pada sistem dalam sub-bab lingkungan sistem dan perancangan aplikasi. Gambaran umum dari sistem dapat dilihat pada Gambar 3.11:



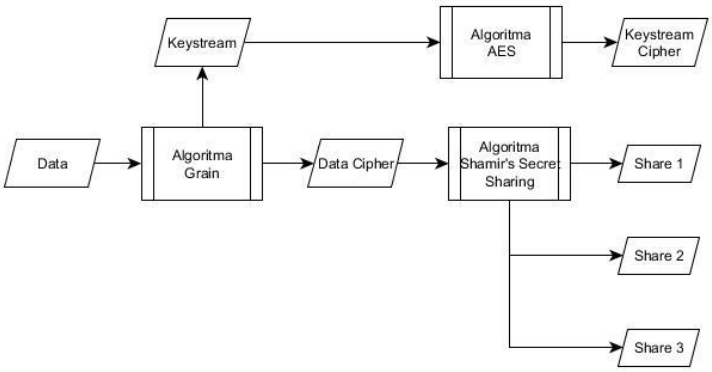
Gambar 3.11 Gambaran umum sistem

Secara umum, sistem memiliki 2 skenario utama. Masing-masing skenario ini terdiri dari 2 proses. Skenario yang pertama terdiri dari proses mengamankan dan menyimpan data sedangkan skenario yang kedua terdiri dari proses mengakses dan mengembalikan data. Proses mengamankan dan proses kebalikannya yaitu mengembalikan data berjalan pada sistem enkripsi dan *secret sharing* sedangkan proses menyimpan dan proses kebalikannya yaitu mengakses data berjalan pada sistem API CloudRail (Gambar 3.11).

Dalam melakukan pengamanan data, *user* memilih data yang akan diproses. Selanjutnya data akan melalui serangkaian algoritme enkripsi (Grain dan AES) dan *secret sharing* (Shamir's Secret Sharing) sehingga menghasilkan data *cipher*. Algoritme pertama adalah Grain. Grain menerima masukan *key* dan IV sehingga menghasilkan karakter acak yang disebut *keystream*. *Keystream* berperan sebagai kunci untuk melakukan enkripsi pada algoritme Grain. Selanjutnya, *keystream* melakukan proses xor dengan data yang ingin diamankan sehingga menghasilkan *data cipher*.

Algoritme kedua adalah AES. AES digunakan untuk melakukan enkripsi terhadap *keystream* menggunakan kunci yang dimasukkan *user*. Proses dari algoritme AES ini akan menghasilkan hasil enkripsi dari *keystream* yang disebut *keystream cipher*.

Algortime ketiga yang digunakan Shamir's Secret Sharing. Algoritme ini digunakan untuk memecah (*split*) *data cipher* menjadi 3 bagian (*share*) yang berbeda. Proses ini berjalan dengan menerima bilangan acak dan bilangan prima yang dimasukkan oleh *user*. Proses mengamankan data dapat dilihat pada Gambar 3.12:

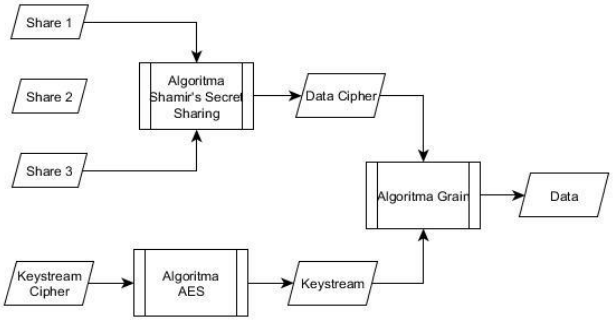


Gambar 3.12 Proses mengamankan data

Proses mengembalikan data adalah proses kebalikan dari mengamankan data. Pada proses mengamankan data, data yang diproses akan melalui serangkaian algoritme sehingga menghasilkan *data cipher*. Proses mengembalikan data bertujuan untuk merubah data yang sudah diamankan dalam bentuk *data cipher* menjadi data semula. Algoritme pertama yang berjalan pada proses ini adalah AES. AES menerima masukan kunci yang dimasukkan oleh *user*. Jika kunci sesuai, maka *keystream cipher* akan kembali menjadi *keystream*.

Algoritme kedua yang berjalan adalah Shamir's Secret Sharing. Pada tahap ini, terdapat 3 kombinasi dari penggabungan (*reconstruct*) nilai *share* untuk mendapatkan *data cipher*. Salah satu kombinasi yang dipilih *user* akan memberikan nilai *data cipher*.

Algoritme ketiga yang berjalan adalah Grain. *Data cipher* dan *keystream cipher* yang masing-masing didapatkan dari proses yang dijalankan Shamir's Secret Sharing dan AES akan melakukan proses xor. Hasil xor dari *data cipher* dan *keystream* ini akan mengembalikan data semula. Proses mengembalikan data dapat dilihat pada Gambar 3.13:



Gambar 3.13 Proses mengembalikan data

Adapun proses menyimpan dan mengakses data yang menggunakan API CloudRail hanya berlaku untuk data yang akan disimpan pada *cloud storage*. Sedangkan menyimpan dan mengakses data menggunakan laptop berjalan langsung pada sistem enkripsi dan *secret sharing*. Data yang disimpan pada *cloud storage* adalah 3 hasil *share* dari algoritme Shamir's Secret Sharing sedangkan data yang disimpan di laptop adalah *keystream cipher*.

Proses menyimpan akan berjalan setelah proses mengamankan data selesai. Proses menyimpan akan melakukan *upload* terhadap 3 hasil *share* ke 3 *cloud service provider*. Proses mengakses berjalan sebelum proses mengembalikan data dilakukan. Proses mengakses akan melakukan *download* terhadap 2 hasil *share* berdasarkan kombinasi yang dipilih oleh *user*.

3.3 Implementasi

Perancangan sistem yang sudah dibuat akan diimplementasikan pada tahap ini. Ada 3 macam implementasi yang dilakukan yaitu implementasi API, implementasi algoritme kriptografi dan implementasi layanan *cloud*.

1. Implementasi API
   API yang digunakan dalam sistem berfungsi untuk menghubungkan sistem dengan beberapa *cloud service provider*. CloudRail adalah sebuah *platform* yang memungkinkan beberapa *cloud service provider* terhubung dengan sistem menggunakan satu perintah yang sama pada setiap fungsi yang ingin dijalankan.
2. Implementasi algoritme kriptografi
   Pada tahap implementasi ini, setiap algoritme kriptografi akan ditulis ke dalam kode program. Algoritme yang digunakan antara lain Grain, AES dan Shamir's Secret Sharing.
3. Implementasi layanan *cloud*
   Pada tahap implementasi ini, akan dituliskan kode program dari fungsi-fungsi layanan *cloud* yang digunakan dalam sistem. Fungsi-fungsi tersebut terdiri dari fungsi *login* sebagai jembatan antara sistem dan *cloud service provider*, fungsi *upload* untuk menyimpan data dan fungsi *download* untuk mengakses data.

3.4 Pengujian

Tahap pengujian akan berisi skenario pengujian dan hasil pengujian yang didapat dari pengujian. Pengujian yang dilakukan ada 2 macam yaitu pengujian algoritme kriptografi dan pengujian layanan *cloud*.

Pengujian algoritme kriptografi melakukan pengujian terhadap masing-masing algoritme kriptografi yaitu Grain, AES dan Shamir's Secret Sharing. Pengujian terhadap algoritme Grain dilakukan dengan menguji *test vector*, proses enkripsi dan dekripsi. Pengujian terhadap algoritme AES dilakukan dengan menguji proses enkripsi dan dekripsi. Pengujian Shamir's Secret Sharing dilakukan dengan menguji proses *split* dan *reconstruct*.

Pengujian *test vector* bertujuan untuk menyesuaikan nilai *keystream* yang dihasilkan sistem dengan *keystream* pada *test vector* yang dirancang oleh pembuat algoritme Grain. Pengujian ini dilakukan sebagai validasi bahwa algoritme Grain yang diterapkan sudah sesuai dengan algoritme Grain yang dibuat oleh Martin Hell. Cara pengujian ini adalah dengan mencocokkan nilai *keystream* sistem dengan nilai *keystream test vector*.

Pengujian enkripsi dan dekripsi bertujuan untuk mengetahui apakah algoritme yang menggunakan teknik enkripsi dan dekripsi sudah berjalan dengan baik. Pengujian enkripsi bisa dikatakan berhasil jika data yang ingin diamankan dapat berubah menjadi data acak yang sulit untuk dibaca. Pengujian dekripsi bisa dikatakan berhasil jika data acak dapat kembali menjadi data semula. Cara pengujian ini dilakukan dengan melihat hasil enkripsi dan menyesuaikan data sebelum proses enkripsi dengan data setelah proses dekripsi.

Pengujian *split* dan *reconstruct* bertujuan untuk mengetahui apakah algoritme Shamir's Secret Sharing yang digunakan sudah berjalan dengan baik. Pengujian *split* bisa dikatakan berhasil jika data yang ingin dipecah dapat terbagi menjadi 3 hasil *share* yang berbeda. Pengujian *reconstruct* bisa dikatakan berhasil jika semua kombinasi yang tersedia untuk melakukan penggabungan dapat mengembalikan data semula. Cara pengujian ini dilakukan dengan melihat hasil *share* dan menyesuaikan data sebelum proses *split* dengan data setelah proses *reconstruct*.

Pengujian layanan *cloud* melihat fungsi-fungsi layanan *cloud* yang digunakan dalam sistem sudah berjalan dengan baik. Fungsi-fungsi tersebut antara lain fungsi *login*, fungsi *upload* dan fungsi *download*.

3.5 Kesimpulan

Kesimpulan merupakan rangkaian akhir dari keseluruhan penelitian. Kesimpulan dituliskan dalam bentuk rangkuman dari proses penelitian serta saran yang berguna untuk pengembangan dan perbaikan sistem yang ada saat ini.

BAB 4 PERANCANGAN

Bab ini menjelaskan perancangan dari sistem yang akan dibuat. Ada 2 sub-bab dalam bab ini. Sub-bab pertama adalah lingkungan sistem dan sub-bab kedua adalah perancangan aplikasi. Pada lingkungan sistem akan dijabarkan kebutuhan fungsional dari sistem yang dibangun. Tahap perancangan aplikasi akan menggambarkan dan menjelaskan secara rinci bagaimana alur dari sistem yang akan dibuat.

4.1 Lingkungan sistem

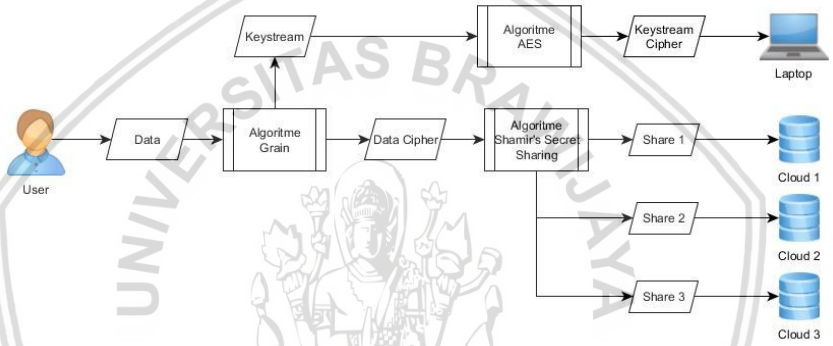
Pada tahap ini akan dituliskan beberapa kebutuhan fungsional yang melingkupi fitur-fitur pada sistem. Adapun fitur-fitur yang direncanakan adalah sebagai berikut:

a. Fitur *login*
   Fitur *login* bertujuan untuk menjembatani sistem dengan *cloud service provider* yang digunakan. Dengan ini, untuk mengakses *cloud* dibutuhkan sebuah fungsi autentifikasi dalam menjalankannya.
b. Fitur enkripsi
   Fitur enkripsi bertujuan untuk merubah data yang ingin diamankan menjadi data acak yang sulit untuk dibaca. Fitur enkripsi pada sistem ini terdapat pada algoritme Grain yag termasuk kelompok algoritme *stream cipher* dan AES yang termasuk kelompok algoritme *block cipher*.
c. Fitur dekripsi
   Fitur dekripsi bertujuan untuk merubah data acak yang sulit dibaca menjadi data asli yang memiliki nilai informasi. Fitur enkripsi pada sistem ini terdapat pada algoritme Grain yang termasuk kelompok algoritme *stream cipher* dan AES yang termasuk kelompok algoritme *block cipher*
d. Fitur *split*
   Fitur *split* bertujuan untuk memecah data menjadi beberapa bagian. Dalam sistem ini fitur *split* yang digunakan dapat memecah data menjadi 3 bagian (*share*) yang berbeda.
e. Fitur *reconstruct*
   Fitur *reconstruct* bertujuan untuk mengembalikan data hasil *share* dengan mengkombinasikan beberapa hasil *share*. Dalam sistem ini, fitur *reconstruct* memiliki 3 kombinasi untuk melakukan penggabungan yaitu kombinasi *share* 1 dan 2, kombinasi *share* 1 dan 3 serta kombinasi *share* 2 dan 3.
f. Fitur *upload*
   Fitur *upload* bertujuan untuk mengunggah data ke internet. Fitur *upload* pada sistem ini dilakukan untuk melakukan *upload* terhadap 3 hasil *share* ke 3 *cloud service provider*.

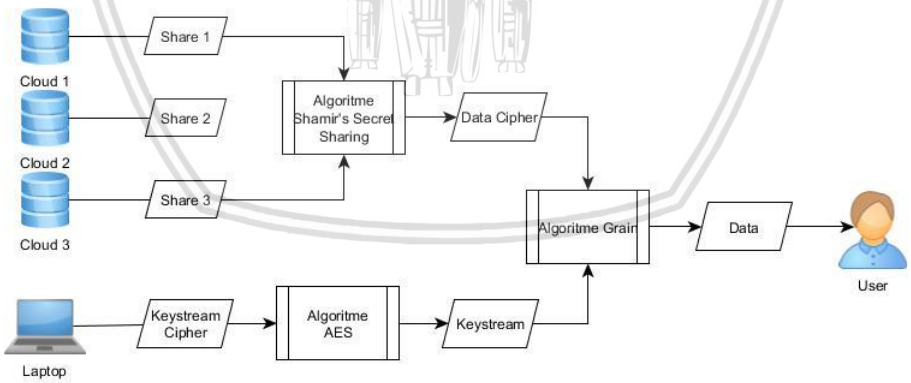
g. Fitur *download*

Fitur *download* bertujuan untuk mengunduh data dari internet. Fitur *download* pada sistem ini dilakukan untuk melakukan *download* terhadap 2 dari 3 hasil *share* berdasarkan kombinasi yang dipilih.

Sistem memiliki 2 skenario utama. Skenario pertama adalah mengamankan dan menyimpan data sedangkan skenario kedua adalah mengakses dan mengembalikan data. Skenario pertama terdiri dari proses enkripsi, *split* dan *upload* sedangkan skenario kedua terdiri dari proses dekripsi, *reconstruct* dan *download*. Berdasarkan penjelasan skenario dan kebutuhan fungsional yang telah dijabarkan sebelumnya, gambaran rinci setiap skenario dapat dilihat pada Gambar 4.14 untuk skenario pertama dan 4.15 untuk skenario kedua.



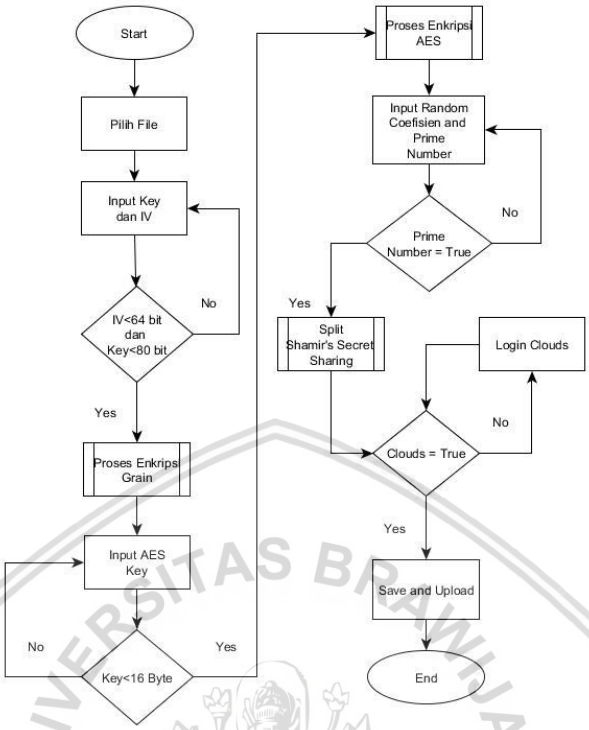
Gambar 4.14 Skenario pertama (proses enkripsi, *split* dan *upload*)



Gambar 4.15 Skenario kedua (proses dekripsi, *reconstruct* dan *download*)

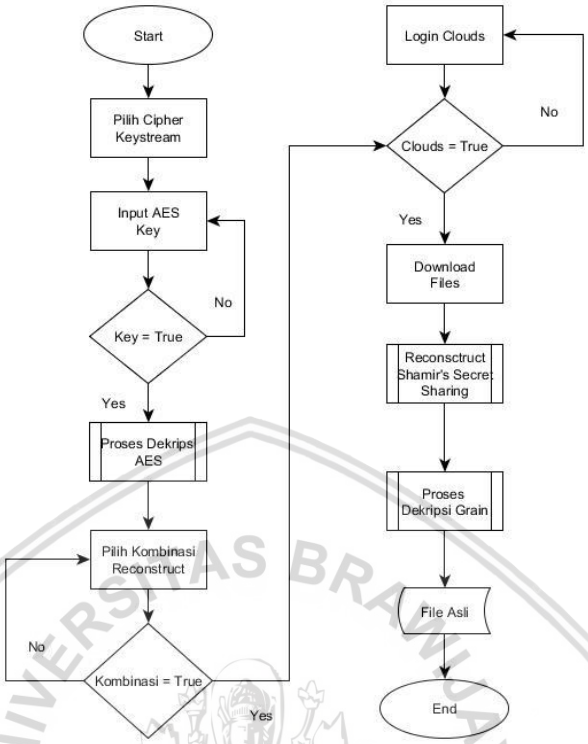
4.2 Perancangan Aplikasi

Tahap perancangan aplikasi akan menjelaskan bagaimana cara kerja sistem berdasarkan diagram alir. Fungsi-fungsi dari sistem terdiri dari *login*, enkripsi dan dekripsi, *split* dan *reconstruct* serta *upload* dan *download*.



Gambar 4.16 Diagram alir proses enkripsi, *split* dan *upload*

Diagram alir pada Gambar 4.16 menjelaskan bagaimana proses enkripsi, *split* dan *upload*. Langkah pertama *user* memilih file yang ingin dienkripsi dan dipecah. Sistem akan menjalankan algoritme Grain setelah *user* memasukkan nilai dari *key* dan *initial vector* (IV) yang menghasilkan *keystream*, sebuah karakter acak sepanjang 80 bit. Panjang maksimal *key* dan IV masing-masing adalah 80 dan 64 bit. Hasil dari *keystream* akan melakukan xor dengan file yang dipilih sehingga menghasilkan *data cipher*. Selanjutnya, sistem akan menjalankan algoritme AES dan melakukan enkripsi terhadap *keystream* setelah *user* memasukkan *key*. Panjang maksimal *key* adalah 16 byte. *Data cipher* yang sudah dihasilkan akan dipecah menjadi 3 *share* menggunakan algoritme Shamir's Secret Sharing setelah memasukkan koefisien acak dan bilangan prima. Bilangan prima adalah bilangan prima terbesar dan terdekat dari nilai desimal data yang ingin dipecah. Hasil enkripsi dari *keystream* akan disimpan pada memori internal laptop sedangkan 3 hasil *share* akan diunggah pada 3 penyedia layanan *cloud* berbeda.



Gambar 4.17 Diagram alir proses dekripsi, *reconstruct* dan *download*

Diagram alir dari Gambar 4.17 menjelaskan proses dekripsi, *reconstruct* dan *download*. Langka pertama *user* memilih file yang merupakan *keystream cipher* dari algoritme Grain. Selanjutnya, sistem akan menjalankan algoritme AES terhadap *keystream cipher* setelah *user* memasukkan nilai *key* yang sesuai. Tahap selanjutnya adalah memilih kombinasi untuk melakukan *reconstruct*. Hasil *share* yang dipilih untuk dikombinasikan akan diunduh dari *cloud storage* masing-masing. Kedua hasil *share* yang akan melalui proses *reconstruct* menggunakan algoritme Shamir's Secret Sharing dan menghasilkan *data cipher*. Hasil *keystream* dan *data cipher* yang sudah didapatkan akan melalui proses xor dan menghasilkan *data plain* atau file yang asli.

BAB 5 IMPLEMENTASI

5.1 Implementasi

Pada sub-bab ini akan dijabarkan tahap-tahap dari implementasi yang sebelumnya telah dirancang. Ada 3 tahap utama dalam implementasi yang akan dilakukan yaitu:

1. Implementasi API
2. Implementasi algoritme kriptografi
3. Implementasi layanan cloud

5.1.1 Implementasi API

Sistem yang dibangun memakai fitur *cloud* dalam penggunaannya. CloudRail merupakan *platform* yang memungkinkan beberapa *cloud service provider* terhubung dengan aplikasi menggunakan satu perintah yang sama pada setiap fungsi yang ingin dijalankan. Ada 2 tahap utama yang dilakukan dalam mengimplementasikan API CloudRail. Tahap pertama yaitu menambahkan *library* API pada sistem dan tahap kedua melakukan registrasi API pada masing-masing aplikasi *cloud* yang sudah dibuat sebelumnya.

5.1.1.1 Penambahan API library

Langkah pertama untuk mengintegrasikan API CloudRail dengan sistem adalah menambahkan *library* API CloudRail pada aplikasi yang dibangun. Ada 2 cara untuk melakukan tahap ini, cara pertama dengan mengunduh langsung SDK yang disediakan CloudRail. Cara kedua melalui *depencency* dari Maven, yaitu dengan menambahkan kode pada file pom.xml. Sistem ini menggunakan cara yang kedua. Adapun kode yang ditambahkan pada file pom.xml dapat dilihat pada Tabel 5.1:

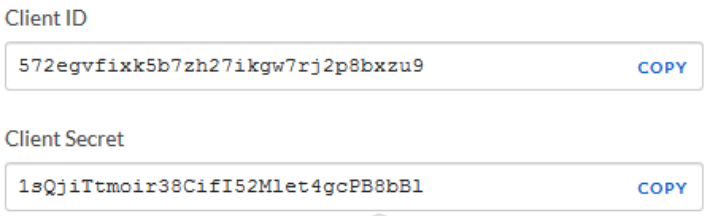
Tabel 5.1 CloudRail SDK

```
1  <dependency>
2          <groupId>com.CloudRail</groupId>
3          <artifactId>CloudRail-si-java</artifactId>
4          <version>2.16.3</version>
5      </dependency>
```

5.1.1.2 Registrasi API

Setelah melakukan penambahan *library* API CloudRail, langkah selanjutnya adalah dengan membuat aplikasi pada masing-masing *cloud service provier* yang digunakan. Melalui aplikasi ini maka akan didapatkan *client id* dan *client secret* yang berfungsi sebagai autentifikasi sistem agar dapat mengakses *cloud*. Selain itu, *redirect* URI juga

harus dituliskan. *Redirect* URI berfungsi untuk mengarahkan sistem ke sebuah halaman *end-point* ketika autentifikasi telah berhasil. Adapun *client id* dan *client secret* yang didapatkan dalam aplikasi *cloud* dapat dilihat pada Gambar 5.18 untuk Box, Gambar 5.19 untuk Google Drive dan Gambar 5.20 untuk Dropbox



Gambar 5.18 *Client id* dan *client secret* Box

Gambar 5.19 *Client id* dan *client secret* Google Drive

Gambar 5.20 *Client id* dan *client secret* Dropbox

Setelah melakukan konfigurasi terhadap masing-masing aplikasi pada *cloud service provider*, langkah selanjutnya adalah menciptakan variabel yang berfungsi untuk memanggil fungsi-fungsi yang tersedia pada layanan CloudRail. Untuk lebih jelas, kode yang dituliskan dapat dilihat pada Tabel 5.2

Tabel 5.2 Inisialisasi variabel

```
1   int port = 8082;
2
3      CloudStorage box = new Box(
4              new LocalReceiver(port),
5              "572egvfixk5b7zh27ikgw7rj2p8bxzu9",
6              "1sQjiTtmoir38CifI52Mlet4gcPB8bBl",
7              "http://localhost:" + port + "/",
8              "Box"
9      );
10     CloudStorage dropbox = new Dropbox(
11             new LocalReceiver(port),
12             "5awq0fmsf4wpopu",
13             "r1unsidi0272zaz",
14             "http://localhost:" + port + "/",
15             "Dropbox"
```

```
16      );
17
18      CloudStorage googledrive = new GoogleDrive(
19              new LocalReceiver(port),
20              "405723900763-
    6m96p7sffg4cejft3uhdgc060kdk3i73.apps.googleusercontent.com",
21              "TaU3ACGnsT1YIsUWu0hn9n8w",
22              "http://localhost:" + port + "/",
23              "GoogleDrive"
        );
```

Tabel 5.2 berisi kode dalam melakukan deklarasi terhadap variabel-variabel yang bertujuan untuk memanggil fungsi-fungsi pada layanan CloudRail. Baris 3-8 merupakan kode untuk menciptakan variabel yang berfungsi memanggil fungsi-fungsi pada layanan Box. Baris 10-16 merupakan kode untuk menciptakan variabel yang berfungsi memanggil fungsi-fungsi pada layanan Dropbox. Baris 18-23 merupakan kode untuk menciptakan variabel yang berfungsi memanggil fungsi-fungsi pada layanan Google Drive.

5.1.2 Implementasi algoritme kriptografi

Algortima kriptografi yang diterapkan pada sistem ini terdiri dari 3 macam yaitu algoritme Grain, algoritme AES dan algoritme Shamir's Secret Sharing.

5.1.2.1 Algoritme Grain

Algoritme Grain memiliki 2 proses utama dalam mendapatkan nilai *keystream* yaitu inisialisasi dan *generate*. Setiap proses memiliki beberapa fungsi yang dijalankan yaitu fungsi $f(x)$ yang dihasilkan dari *state* LFSR, fungsi $g(x)$ yang dihasilkan dari *state* NFSR, fungsi $h(x)$ yang dihasilkan dari 4 indeks LFSR dan 1 indeks NFSR. Setelah *keystream* didapatkan, maka proses yang dapat dijalankan adalah proses enkripsi dan proses dekripsi. Berikut kode beserta penjelasan dari setiap proses.

Tabel 5.3 Proses inisialisasi *state* LFSR dan NFSR

```
1   public void initialize() {
2   for (int i = 0; i < 160; i++) {
3               f_x = lfsr[62] ^ lfsr[51] ^ lfsr[38] ^ lfsr[23]
    ^ lfsr[13] ^ lfsr[0];

4               g_x = lfsr[0] ^ nfsr[62] ^ nfsr[60] ^ nfsr[52] ^
    nfsr[45] ^ nfsr[37] ^ nfsr[33] ^ nfsr[28] ^ nfsr[21]
5                   ^  nfsr[14]  ^  nfsr[9]  ^  nfsr[0]  ^
    (nfsr[63] & nfsr[60]) ^ (nfsr[37] & nfsr[33]) ^ (nfsr[15] &
6   nfsr[9])
                    ^ (nfsr[60] & nfsr[52] & nfsr[45]) ^
    (nfsr[33] & nfsr[28] & nfsr[21]) ^ (nfsr[63] & nfsr[45] &
7   nfsr[28] & nfsr[9])
```

```
                    ^ (nfsr[60] & nfsr[52] & nfsr[37] &
    nfsr[33]) ^ (nfsr[63] & nfsr[60] & nfsr[21] & nfsr[15])
 8                  ^ (nfsr[63] & nfsr[60] & nfsr[52] &
    nfsr[45] & nfsr[37]) ^ (nfsr[33] & nfsr[28] & nfsr[21] &
 9  nfsr[15] & nfsr[9])
                    ^ (nfsr[52] & nfsr[45] & nfsr[37] &
    nfsr[33] & nfsr[28] & nfsr[21]);
10          int x0 = lfsr[3];
            int x1 = lfsr[25];
11          int x2 = lfsr[46];
12          int x3 = lfsr[64];
13          int x4 = nfsr[63];
14
15          h_x = x1 ^ x4 ^ (x0 & x3) ^ (x2 & x3) ^ (x3 & x4)
16  ^ (x0 & x1 & x2)
17                  ^ (x0 & x2 & x3) ^ (x0 & x2 & x4) ^ (x1
    & x2 & x4) ^ (x2 & x3 & x4);
18          z = nfsr[1] ^ nfsr[2] ^ nfsr[4] ^ nfsr[10] ^
    nfsr[31] ^ nfsr[43] ^ nfsr[56];
19
20          filter = (h_x ^ z);

21          fxh = (f_x ^ filter); //indeks terakhir lfsr
    diisi dengan nilai ini
22          gxh = (g_x ^ filter); //indeks terakhir nfsr
    diisi dengan nilai ini

23          rotate.geser(lfsr);
24          rotate.geser(nfsr);
25
26          lfsr[79] = fxh;
27          nfsr[79] = gxh;
28  }
29
30
```

Tabel 5.3 merupakan kode untuk melakukan proses inisialisasi *state* LFSR dan NFSR pada algoritme Grain. Baris 3-17 adalah kode untuk mendapatkan nilai dari fungsi $f(x)$, $g(x)$ dan $h(x)$. Baris 18 adalah kode untuk mendapatkan fungsi z. Fungsi z merupakan perhitungan operasi xor terhadap beberapa indeks pada *state* NFSR. Baris 20-22 adalah kode untuk mendapatkan 2 nilai baru yang akan dimasukkan pada indeks terakhir LFSR dan NFSR. Nilai tersebut didapat dari operasi xor antara fungsi $f(x)$, $h(x)$ dan z untuk LFSR dan operasi xor antara fungsi $g(x)$, $h(x)$ dan z untuk NFSR. Baris 23-27 adalah kode untuk melakukan pergeseran terhadap *state* LFSR dan NFSR sebanyak 1 kali serta memasukkan 2 nilai baru pada indeks terakhir *state* LFSR dan NFSR. Proses ini dilakukan sebanyak 160 kali sehingga menghasilkan *state* LFSR dan NFSR yang baru.

Tabel 5.4 Proses *generate keystream*

```
1   public String generateFile(String mess) {
2           int[] keystream = new int[mess.length()];
3           String hasil = "";
4           for (int i = 0; i < mess.length(); i++) {
5               f_x = lfsr[62] ^ lfsr[51] ^ lfsr[38] ^ lfsr[23]
    ^ lfsr[13] ^ lfsr[0];
6
7               g_x = lfsr[0] ^ nfsr[62] ^ nfsr[60] ^ nfsr[52] ^
    nfsr[45] ^ nfsr[37] ^ nfsr[33] ^ nfsr[28] ^ nfsr[21]
8                       ^  nfsr[14]  ^  nfsr[9]  ^  nfsr[0]  ^
    (nfsr[63] & nfsr[60]) ^ (nfsr[37] & nfsr[33]) ^ (nfsr[15] &
    nfsr[9])
9                       ^ (nfsr[60] & nfsr[52] & nfsr[45]) ^
    (nfsr[33] & nfsr[28] & nfsr[21]) ^ (nfsr[63] & nfsr[45] &
    nfsr[28] & nfsr[9])
10                      ^ (nfsr[60] & nfsr[52] & nfsr[37] &
    nfsr[33]) ^ (nfsr[63] & nfsr[60] & nfsr[21] & nfsr[15])
11                      ^ (nfsr[63] & nfsr[60] & nfsr[52] &
    nfsr[45] & nfsr[37]) ^ (nfsr[33] & nfsr[28] & nfsr[21] &
    nfsr[15] & nfsr[9])
12                      ^ (nfsr[52] & nfsr[45] & nfsr[37] &
    nfsr[33] & nfsr[28] & nfsr[21]);
13
14              int x0 = lfsr[3];
15              int x1 = lfsr[25];
16              int x2 = lfsr[46];
17              int x3 = lfsr[64];
18              int x4 = nfsr[63];
19
20              h_x = x1 ^ x4 ^ (x0 & x3) ^ (x2 & x3) ^ (x3 & x4)
    ^ (x0 & x1 & x2)
21                      ^ (x0 & x2 & x3) ^ (x0 & x2 & x4) ^ (x1
    & x2 & x4) ^ (x2 & x3 & x4);
22
23              z = nfsr[1] ^ nfsr[2] ^ nfsr[4] ^ nfsr[10] ^
    nfsr[31] ^ nfsr[43] ^ nfsr[56];
24
25              keystream[i] = (h_x ^ z);
26              hasil += String.valueOf(keystream[i]);
27
28              rotate.geser(lfsr);
29              rotate.geser(nfsr);
30
31              lfsr[79] = f_x;
32              nfsr[79] = g_x;
33          }
34          //  String  hexKeystream  =  new  BigInteger(hasil,
    2).toString(16);
35          return hasil;
36      }
```

Tabel 5.4 adalah kode untuk melakukan proses *generate keystream*. Proses ini hampir sama dengan proses inisialisasi. Perbedaan 2 proses ini terdapat pada baris 25-35. Pada baris 25-35 fungsi $h(x)$ yang melakukan proses xor dengan fungsi z akan menjadi bit-bit *keystream*. Proses *generate keystream* dilakukan sebanyak panjang bit dari data yang ingin diamankan. Sedangkan indeks terakhir LFSR dan NFSR masing-masing akan diisi dengan nilai pada fungsi $f(x)$ dan $g(x)$ setelah melakukan pergeseran sebanyak 1 kali.

Tabel 5.5 Proses enkripsi dan dekripsi Grain

```
1   public String encryptGrainFile(String mess, String keystream)
2   {
3           int[] message = new int[mess.length()];
4           for (int i = 0; i < mess.length(); i++) {
5               message[i]                                          =
    Byte.parseByte(String.valueOf(mess.charAt(i)));
6           }
7           int panjang_mess = mess.length();
8           String test = Integer.toString(panjang_mess);
9
10          int[] tmpKeystream = new int[keystream.length()];
11          for (int i = 0; i < keystream.length(); i++) {
12              tmpKeystream[i]                                     =
    Byte.parseByte(String.valueOf(keystream.charAt(i)));
13          }
14
15          int[] tmpCipher = new int[mess.length()];
16          String cipher = "";
17
18          for (int i = 0; i < mess.length(); i++) {
19              tmpCipher[i] = tmpKeystream[i] ^ message[i];
20              cipher += String.valueOf(tmpCipher[i]);
21          }
22
23          return cipher;
24      }
25
26  public   String   decryptGrain(String   cipherGrain,   String
    keystream) {
27          int[] bin_keystream = new int[keystream.length()];
28          int[] bin_cipher = new int[cipherGrain.length()];
29
30          for (int i = 0; i < kstream.length(); i++) {
31              bin_keystream[i]                                    =
    Byte.parseByte(String.valueOf(kstream.charAt(i)));
32          }
33          for (int i = 0; i < cipher.length(); i++) {
34              bin_cipher[i]                                       =
    Byte.parseByte(String.valueOf(cipher.charAt(i)));
35          }
36          String binary_plain = "";
37          int[] tmpPlain = new int[cipher.length()];
```

```
38          for (int i = 0; i < cipher.length(); i++) {
39              tmpPlain[i] = bin_cipher[i] ^ bin_keystream[i];
40              binary_plain += String.valueOf(tmpPlain[i]);
41          }
42          return binary_plain;
43      }
```

Tabel 5.5 adalah kode untuk melakukan proses enkripsi dan dekripsi. Proses enkripsi dan dekripsi dilakukan dengan melakukan operasi xor antara bit-bit data dengan bit-bit *keystream*. Baris 1-24 adalah kode untuk melakukan enkripsi dan baris 26-43 adalah kode untuk melakukan proses dekripsi.

5.1.2.2 Algoritme AES

Algoritme AES adalah algoritme kriptografi kedua yang diterapkan pada sistem ini. Proses dalam algoritme ini adalah enkripsi untuk mengacak data dan dekripsi untuk mengembalikan data acak menjadi data asli. Dalam melakukan proses enkripsi dan dekripsi, dibutuhkan sebuah kunci untuk merubah sebuah data. Sistem ini menggunakan *library* dari Java untuk menjalankan algoritme AES. Tabel 5.6 merupakan proses untuk melakukan enkripsi dan dekripsi. Baris 1-12 adalah kode untuk melakukan enkripsi dan baris 14-25 adalah proses untuk melakukan dekripsi

Tabel 5.6 Proses enkripsi dan dekripsi AES

```
1   public static String encryptAES(String Data, String
2   kunciEnkripsi)
3               throws Exception {
4
5           Cipher c = Cipher.getInstance("AES");
            Key key = generateKey(kunciEnkripsi);
6           c.init(Cipher.ENCRYPT_MODE, key);
7           byte[] encVal = c.doFinal(Data.getBytes());
8
9           String                encryptedValue                =
    DatatypeConverter.printBase64Binary(encVal);
10          return encryptedValue;
11      }
12
13  public static String decryptAES(String encryptedData, String
    kunciEnkripsi)
14              throws Exception {

15          Cipher c = Cipher.getInstance("AES");
16          Key key = generateKey(kunciEnkripsi);
17          c.init(Cipher.DECRYPT_MODE, key);
18
19          byte[] decordedValue = DatatypeConverter
                        .parseBase64Binary(encryptedData);
20          byte[] decValue = c.doFinal(decordedValue);
21          String decryptedValue = new String(decValue);
```

```
22          return decryptedValue; }
23
24
```

5.1.2.3 Algoritme Shamir's Secret Sharing

Shamir's Secret Sharing merupakan algoritme kriptografi terakhir yang diimplementasikan dalam sistem. Ada 2 proses utama dalam algoritme Shamir's Secret Sharing. Proses pertama adalah *split* dan proses kedua adalah *reconstruct*. Sistem yang dibangun memungkinkan data akan dipecah menjadi 3 bagian $(n = 3)$. Dalam mengembalikan data, maka hanya dibutuhkan 2 dari 3 bagian $(k = 2)$. Dengan ini persamaan yang dibentuk adalah:

$$f(x) = D + a_1 x \bmod p \qquad (2.11)$$

Tabel 5.7 Proses *split*

```
1  public String[] share(String cipherGrain) {
2  BigInteger dec_hasil = new BigInteger(cipherGrain, 16);
3          BigInteger[] share = new BigInteger[INDEX];
4          BigInteger[] shareFinal = new BigInteger[INDEX];
5          String[] bagian = new String[INDEX];
6  BigInteger coeff = new
   BigInteger("98755321212436578564657687864532457 89");
7          BigInteger mod = new
   BigInteger("774297941774711167597341130752418072262985264423341744615435326881063930 91");
8          for (int i = 1; i <= 3; i++) {
9              test[i] =
   coeff.multiply(BigInteger.valueOf(i));
10             share[i] = dec_hasil.add(test[i]);
11             shareFinal[i] = share[i].mod(mod);
12
13             bagian[i] = shareFinal[i].toString(16);
14         }
15         return bagian;
16     }
17
18
```

Tabel 5.7 adalah kode untuk melakukan *split* data. Algoritme ini melakukan perhitungan terhadap bilangan desimal. Baris 2 adalah kode untuk merubah data (*hexadecimal*) ke dalam bentuk desimal $(D = 774297941774711167597341130752418072262985264423341744615435326881063922 23)$. Baris 3-4 adalah kode untuk mendeklarasikan variabel yang akan menampun nilai dari hasil *split*. Baris 6 dan 7 masing-masing merupakan kode yang menampung nilai *random number* $(a_1 x =$

9875532121243657856465768786453245789) dan prima ($p =$ 774297941774711167597341130752418072262985264423341744615435326881063930 91). Bilangan prima yang dipilih adalah bilangan prima terbesar dan terdekat dari bilangan desimal data.

***Tabel 5.8 Proses* reconstruct**

```
1  public String reconstruct(String pil) {
2          String cipherShamir1;
3          String cipherShamir2;
4          BigInteger decShamir1;
5          BigInteger decShamir2;
6          BigInteger bil_1;
7          BigInteger bil_2;
8          BigInteger dec_dekripsi = null;
9          BigInteger              mod              =              new
   BigInteger("774297941774711167597341130752418072262985264423341744615435326881063930
   91");
10
11         if (pil.equals("1")) {
12             cipherShamir1                                          =
   myFile.readTextFile("box.txt").toUpperCase();
13             cipherShamir2                                          =
14 myFile.readTextFile("gd.txt").toUpperCase();
15
16             decShamir1                                             =
   otherFunctions.hex2decimal(cipherShamir1);
17             decShamir2                                             =
   otherFunctions.hex2decimal(cipherShamir2);
18
19             bil_1                                                  =
   BigInteger.valueOf(2).multiply(decShamir1).divide(BigInteger
   .valueOf(1));
20             bil_2                                                  =
   BigInteger.valueOf(1).multiply(decShamir2).divide(BigInteger
   .valueOf(-1));
21             dec_dekripsi = bil_1.add(bil_2);
22
23         } else if (pil.equals("2")) {
24             cipherShamir1                                          =
   myFile.readTextFile("box.txt").toUpperCase();
25             cipherShamir2                                          =
   myFile.readTextFile("db.txt").toUpperCase();
26
27             decShamir1                                             =
   otherFunctions.hex2decimal(cipherShamir1);
28             decShamir2                                             =
   otherFunctions.hex2decimal(cipherShamir2);
29
30             bil_1                                                  =
   BigInteger.valueOf(3).multiply(decShamir1).divide(BigInteger
   .valueOf(2));
31
```

```
            bil_2                                                   =
BigInteger.valueOf(1).multiply(decShamir2).divide(BigInteger
.valueOf(-2));
            dec_dekripsi = bil_1.add(bil_2);
        } else if (pil.equals("3")) {
            cipherShamir1                                           =
myFile.readTextFile("gd.txt").toUpperCase();
            cipherShamir2                                           =
myFile.readTextFile("db.txt").toUpperCase();

            decShamir1                                              =
otherFunctions.hex2decimal(cipherShamir1);
            decShamir2                                              =
otherFunctions.hex2decimal(cipherShamir2);

            bil_1                                                   =
BigInteger.valueOf(3).multiply(decShamir1).divide(BigInteger
.valueOf(1));
            bil_2                                                   =
BigInteger.valueOf(2).multiply(decShamir2).divide(BigInteger
.valueOf(-1));
            dec_dekripsi = bil_1.add(bil_2);
        }
        BigInteger dec_dekripsiFinal = dec_dekripsi.mod(mod);
        System.out.println(dec_dekripsiFinal);
        String cipherGrain = dec_dekripsiFinal.toString(16);

        return cipherGrain;
    }
```

Tabel 5.8 adalah kode untuk proses *reconstruct*. Pada sistem ini, *reconstruct* memiliki 3 kombinasi yang dapat dilakukan. Kombinasi pertama adalah *share* 1 dan *share* 2 (baris 11-21). Kombinasi kedua adalah *share* 1 dan *share* 3 (baris 23-32). Kombinasi ketiga adalah *share* 2 dan *share* 3 (baris 33-43). Perhitungan kombinasi pertama dapat dilihat pada persamaan 2.12, kombinasi kedua pada persamaan 2.13 dan kombinasi ketiga pada persamaan 2.14.

$$D = \left(share\ 1\ \times \frac{2}{2-1} + share\ 2\ \times \frac{1}{1-2}\right) mod\ p \quad (2.12)$$

$$D = \left(share\ 1\ \times \frac{3}{3-1} + share\ 3\ \times \frac{1}{1-3}\right) mod\ p \quad (2.13)$$

$$D = \left(share\ 2\ \times \frac{3}{3-2} + share\ 3\ \times \frac{2}{2-3}\right) mod\ p \quad (2.14)$$

5.1.3 Implementasi layanan cloud

CloudRail memungkinkan penggunaan *multi-cloud* dapat berjalan dengan perintah yang sama. Sistem ini menggunakan 3 *cloud* sebagai alternatif media penyimpanan. *Cloud service provider* yang digunakan adalah Box, Google Drive dan

Dropbox. Ada 3 fungsi yang diterapkan pada sistem ini yaitu fungsi *login*, fungsi *upload* dan fungsi *download*.

Tabel 5.9 Fungsi *login*

```
1   public void login() {
2           CloudRail.setAppKey("5936127b9270873eded62247");
3           box.login();
4           googledrive.login();
5           dropbox.login();
6       }
```

Tabel 5.9 adalah kode untuk melakukan fungsi *login* pada masing-masing *cloud*. Baris 2 adalah kode untuk mendeklarasikan *app key* dari akun CloudRail yang telah dibuat. Kode tersebut dituliskan setiap kali melakukan fungsi-fungsi yang ingin dijalankan pada sistem.

Tabel 5.10 Fungsi *upload*

```
1   public void upload() throws FileNotFoundException {
2           CloudRail.setAppKey("5936127b9270873eded62247");

3           InputStream upload = new
    FileInputStream("share1.txt");
4           box.upload("/bara/share1.txt", upload, 1024L, true);
5           System.out.println("Berhasil Upload ke Box");
6
7           InputStream upload2 = new
    FileInputStream("share2.txt");
8           googledrive.upload("/bara/share2.txt", upload2,
    1024L, true);
9           System.out.println("Berhasil Upload ke Google
10  Drive");

11          InputStream upload3 = new
12  FileInputStream("share3.txt");
            dropbox.upload("/bara/share3.txt", upload3, 1024L,
13  true);
            System.out.println("Berhasil Upload ke Dropbox");
14
15      }
16
```

Tabel 5.10 adalah kode untuk melakukan fungsi *upload*. Baris 3-5 adalah kode untuk proses *upload share* 1 ke Box. Baris 7-10 adalah kode untuk proses *upload share 2* ke Google Drive. Baris 12-14 adalah kode untuk proses *upload share* 3 ke Dropbox.

Tabel 5.11 Fungsi *download*

```
1  public void download1() throws FileNotFoundException {
2          CloudRail.setAppKey("5936127b9270873eded62247");
3
4          InputStream                download                =
   box.download("/bara/share1.txt");
5          OutputStream   simpan   =   new   FileOutputStream(new
   File("box.txt"));
6
7          try {
8              int read = 0;
9              byte[] bytes = new byte[1024];
10
11             while ((read = download.read(bytes)) != -1) {
12                 simpan.write(bytes, 0, read);
13             }
14
15         } catch (IOException e) {
16             e.printStackTrace();
17         } finally {
18             if (download != null) {
19                 try {
20                     download.close();
21                 } catch (IOException e) {
22                     e.printStackTrace();
23                 }
24             }
25             if (simpan != null) {
26                 try {
27                     // outputStream.flush();
28                     simpan.close();
29                 } catch (IOException e) {
30                     e.printStackTrace();
31                 }
32             }
33         }
34         System.out.println("Berhasil Download dari Box");
35
36         InputStream                download2               =
   googledrive.download("/bara/share2.txt");
37         OutputStream   simpan2   =   new   FileOutputStream(new
   File("gd.txt"));
38
39         try {
40             int read = 0;
41             byte[] bytes = new byte[1024];
42
43             while ((read = download2.read(bytes)) != -1) {
44                 simpan2.write(bytes, 0, read);
45             }
46
47         } catch (IOException e) {
48             e.printStackTrace();
```

```
49          } finally {
50              if (download2 != null) {
51                  try {
52                      download2.close();
53                  } catch (IOException e) {
54                      e.printStackTrace();
55                  }
56              }
57              if (simpan2 != null) {
58                  try {
59                      // outputStream.flush();
60                      simpan2.close();
61                  } catch (IOException e) {
62                      e.printStackTrace();
63                  }
64              }
65          }
66          System.out.println("Berhasil  Download  dari  Google
    Drive");
67
68      }
69
70      public void download2() throws FileNotFoundException {
71          CloudRail.setAppKey("5936127b9270873eded62247");
72
73          InputStream                download                =
    box.download("/bara/share1.txt");
74          OutputStream  simpan  =  new  FileOutputStream(new
    File("box.txt"));
75
76          try {
77              int read = 0;
78              byte[] bytes = new byte[1024];
79
80              while ((read = download.read(bytes)) != -1) {
81                  simpan.write(bytes, 0, read);
82              }
83
84          } catch (IOException e) {
85              e.printStackTrace();
86          } finally {
87              if (download != null) {
88                  try {
89                      download.close();
90                  } catch (IOException e) {
91                      e.printStackTrace();
92                  }
93              }
94              if (simpan != null) {
95                  try {
96                      // outputStream.flush();
97                      simpan.close();
98                  } catch (IOException e) {
99                      e.printStackTrace();
```

```
100                 }
101             }
102         }
103         System.out.println("Berhasil Download dari Box");
104 
105         InputStream download3 = dropbox.download("/bara/share3.txt");
106         OutputStream simpan3 = new FileOutputStream(new File("db.txt"));
107 
108         try {
109             int read = 0;
110             byte[] bytes = new byte[1024];
111 
112             while ((read = download3.read(bytes)) != -1) {
113                 simpan3.write(bytes, 0, read);
114             }
115 
116         } catch (IOException e) {
117             e.printStackTrace();
118         } finally {
119             if (download3 != null) {
120                 try {
121                     download3.close();
122                 } catch (IOException e) {
123                     e.printStackTrace();
124                 }
125             }
126             if (simpan3 != null) {
127                 try {
128                     // outputStream.flush();
129                     simpan3.close();
130                 } catch (IOException e) {
131                     e.printStackTrace();
132                 }
133             }
134         }
135         System.out.println("Berhasil Download dari Dropbox");
136 
137     }
138 
139     public void download3() throws FileNotFoundException {
140 
141         CloudRail.setAppKey("5936127b9270873eded62247");
142 
143         InputStream download2 = googledrive.download("/bara/share2.txt");
144         OutputStream simpan2 = new FileOutputStream(new File("gd.txt"));
145 
146         try {
147             int read = 0;
148             byte[] bytes = new byte[1024];
```

```
149
150             while ((read = download2.read(bytes)) != -1) {
151                 simpan2.write(bytes, 0, read);
152             }
153
154         } catch (IOException e) {
155             e.printStackTrace();
156         } finally {
157             if (download2 != null) {
158                 try {
159                     download2.close();
160                 } catch (IOException e) {
161                     e.printStackTrace();
162                 }
163             }
164             if (simpan2 != null) {
165                 try {
166                     // outputStream.flush();
167                     simpan2.close();
168                 } catch (IOException e) {
169                     e.printStackTrace();
170                 }
171             }
172         }
173         System.out.println("Berhasil  Download  dari  Google
    Drive");
174
175         InputStream                 download3                =
    dropbox.download("/bara/share3.txt");
176         OutputStream  simpan3  =  new  FileOutputStream(new
    File("db.txt"));
177
178         try {
179             int read = 0;
180             byte[] bytes = new byte[1024];
181
182             while ((read = download3.read(bytes)) != -1) {
183                 simpan3.write(bytes, 0, read);
184             }
185
186         } catch (IOException e) {
187             e.printStackTrace();
188         } finally {
189             if (download3 != null) {
190                 try {
191                     download3.close();
192                 } catch (IOException e) {
193                     e.printStackTrace();
194                 }
195             }
196             if (simpan3 != null) {
197                 try {
198                     // outputStream.flush();
199                     simpan3.close();
```

```
200                 } catch (IOException e) {
201                     e.printStackTrace();
202                 }
203             }
204         }
205         System.out.println("Berhasil      Download      dari
    Dropbox");
206
207     }
```

Tabel 5.11 adalah kode untuk proses *download*. Ada 3 *method* yang dapat dipanggil dalam proses *download*. Ketiga *method* ini dijalankan masing-masing berdasarkan pilihan kombinasi *reconstruct* yang diinginkan. Baris 1-68 adalah kode untuk melakukan proses *download* pada Box dan Google Drive. Baris 70-137 adalah kode untuk melakukan proses *download* pada Box dan Dropbox. Baris 139-207 adalah kode untuk melakukan proses *download* pada Google Drive da Dropbox.

BAB 6 PENGUJIAN

Skenario pengujian dan hasil pengujian akan dijabarkan pada bab ini. Pengujian dibagi menjadi pengujian algoritme kriptografi serta pengujian layanan *cloud*.

6.1 Pengujian algoritme kriptografi

Pada sub-bab ini akan dijabarkan skenario pengujian dan hasil pengujian terhadap algoritme-algoritme kriptografi yang diterapkan pada sistem. Pengujian ini dibagi menjadi pengujian algoritme Grain, pengujian algoritme AES dan pengujian algoritme Shamir's Secret Sharing. Pengujian ini bertujuan untuk melihat fungsionalitas algoritme-algoritme yang digunakan. Pengujian dapat dikatakan berhasil jika data yang akan diamankan berubah menjadi data acak yang tidak bernilai dan sulit untuk dibaca. Jika tujuan dari pengujian ini berhasil, maka aspek keamanan dalam hal kerahasiaan informasi dapat terpenuhi. File yang akan diamankan dalam pengujian ini adalah file txt yang dapat dilihat pada Gambar 6.21:

Gambar 6.21 File yang akan diamankan

6.1.1 Pengujian algoritme Grain

Pengujian pada algoritme Grain dilakukan dengan menguji validitas *test vector* serta menguji proses enkripsi dan dekripsi. Pengujian validitas *test vector* bertujuan untuk menyesuaikan hasil keluaran dari *keystream* pada sistem dengan *keystream* keluaran *test vector* yang dirancang oleh pembuat algoritme Grain. Adapun masukan dan keluaran pada *test vector* adalah sebagai berikut:

Key:	00000000000000000000
IV:	000000000000000
Keystream:	7b978cf36846e5f4ee0b
Key:	0123456789abcdef1234
IV:	0123456789abcdef
Keystream:	42b567cccc65317680225

Algoritme Grain dikatakan *valid* jika hasil *keystream* pada sistem bernilai sama dengan hasil *keystream* pada *test vector*. Proses enkripsi dikatakan *valid* jika data yang diamankan berhasil dirubah menjadi data acak yang sulit dibaca. Dekripsi dikatakan *valid* jika data sebelum melakukan enkripsi bernilai sama dengan data setelah melakukan proses dekripsi. Prosedur pengujian akan melakukan proses enkripsi dan dekripsi pada data yang dapat dilihat pada Gambar 6.21.

File yang akan diamankan menggunakan algoritme Grain adalah file txt. Gambar file dapat dilihat pada Gambar 6.21. Hasil pengujian *test vector* dapat dilihat pada Tabel 6.12 sedangkan hasil pengujian proses enkripsi dan dekripsi dapat dilihat pada Tabel 6.13

Tabel 6.12 Hasil pengujian *test vector*

No	*Key*	IV	*Keystream*	Keterangan
1	00000 00000 00000 00000	000 000 000 000 000 0	7B978CF36846E5F4EE0BD4D686 11156F45435A8720CCB762D118 982E784CD6	*Valid*
2	01234 56789 abcde f1234	012 345 678 9ab cde f	42B567CCC65317680225CD83B2 1DB3E41781EA81F82274C44F8E 2EA43C188D	*Valid*

Tabel 6.12 adalah hasil pengujian *test vector* algoritme Grain. Nilai masukan ini disesuaikan dengan nilai *key* dan IV pada *test vector*. Berdasarkan kedua pengujian yang dilakukan, 20 karakter pertama dari masing-masing nilai *keystream* adalah **7B978CF36846E5F4EE0B** dan **42B567CCC65317680225**. Nilai *keystream* yang dihasilkan ini memiliki nilai yang cocok dengan nilai *keystream* pada *test vector*. Dengan ini, pengujian *test vector* Grain dapat dikatakan *valid*.

Tabel 6.13 Hasil pengujian enkripsi dan dekripsi Grain

No	*Keystream*	*Cipher*	*Plain*	Keterangan
1	7B978CF36846E 5F4EE0BD4D686 11156F45435A8 720CCB762D118 982E784CD6	2BD2DEB02704A4B5 A02B959AC15E47261 10E1BA76B9EFE3285 57DF7C390A9F	7B978CF368 46E5F4EE0B D4D686111 56F45435A8 720CCB762 D118982E78 4CD6	*Valid*

2	42B567CCC65317680225CD83B21DB3E41781EA81F82274C44F8E2EA43C188D	12F0358F891156294C058CCFF552E1AD43CCABA1B3703D941BC169F67D5EC4	42B567CCC65317680225CD83B21DB3E41781EA81F82274C44F8E2EA43C188D	*Valid*
3	368E0AB60960181456737FC11E0F86F111CA5B436C4BD441D12C8420558997	66CB58F54622595518533E8D5940D4B845871A6327199D118563C37214CFDE	368E0AB6096018145673 7FC11E0F86F111CA5B436C4BD441D12C8420558997	*Valid*
4	FFFF7F00383F87D07F3969AE8293A740B2A979E6AE9D0FE49B01427915F502	AFBA2D43777DC691311928E2C5DCF509E6E438C6E5CF46B4CF4E052B54B34B	FFFF7F00383F87D07F3969AE8293A740B2A979E6AE9D0FE49B01427915F502	*Valid*
5	14963A7B255DB0FCC49F7B8DB333B425C08C96CB61B53C1D925FE2ABB36B66	44D368386A1FF1BD8ABF3AC1F47CE66C94C1D7EB2AE7754DC610A5F9F22D2F	14963A7B255DB0FCC49F7B8DB333B425C08C96CB61B53C1D925FE2ABB36B66	*Valid*
6	6FA2A8CFC73128DF5012203EA17B9174E4693E8CC745EB80F3ADA52F1CD474	3FE7FA8C8873699E1E326172E634C33DB0247FAC8C17A2D0A7E2E27D5D923D	6FA2A8CFC73128DF5012203EA17B9174E4693E8CC745EB80F3ADA52F1CD474	*Valid*
7	E7D307E37E7462CD87D6A8ADE7EC569F10455364E711A438C909C99A9247F0	B79655A03136238CC9F6E9E1A0A304D644081244AC43ED689D468EC8D301B9	E7D307E37E7462CD87D6A8ADE7EC569F10455364E711A438C909C99A9247F0	*Valid*
8	293143C0DF7ABC0280766CEB25994AA9DD013D6E066F26AEDA1D6CB0219FA2	797411839038FD43CE562DA762D618E0894C7C4E4D3D6FFE8E522BE260D9EB	293143C0DF7ABC0280766CEB25994AA9DD013D6E066F26AEDA1D6CB0219FA2	*Valid*

9	45058AD2A75B7FF966334744C2A9725CF3655DC61CC6F814A4244C9CD10890	1540D891E8193EB82813060885E62015A7281CE65794B144F06B0BCE904ED9	45058AD2A75B7FF966334744C2A9725CF3655DC61CC6F814A4244C9CD10890	*Valid*

Tabel 6.13 adalah hasil pengujian untuk proses enkripsi dan dekripsi. Kolom *Keystream* berisi nilai *keystream* yang telah didapatkan setelah memasukkan nilai *key* dan IV pada tahap sebelumnya. Bit-bit dari nilai *keystream* akan melakukan proses xor dengan bit-bit data yang akan diamankan. Hasil dari proses xor adalah nilai yang ditulis pada kolom *Cipher*. Ada 9 percobaan yang dilakukan. 2 percobaan pertama menggunakan nilai *key* dan IV berdasarkan *test vector*. 7 percobaan selanjutnya menggunakan nilai *key* dan IV sebagai berikut:

1. 80 bit *key* bernilai 1 dan 80 bit IV bernilai 0
2. 80 bit *key* bernilai 0 dan 80 bit IV bernilai 1
3. 80 bit *key* dan IV bernilai 1
4. 40 bit pertama *key* dan IV bernilai 1 serta 40 bit terakhir *key* dan IV bernilai 0
5. 40 bit pertama *key* dan IV bernilai 0 serta 40 bit terakhir *key* dan IV bernilai 1
6. 40 bit pertama *key* dan 40 bit terakhir IV bernilai 0 serta 40 bit pertama IV dan 40 bit terakhir *key* bernilai 1
7. 40 bit pertama *key* dan 40 bit terakhir IV bernilai 1 serta 40 bit pertama IV dan 40 bit terakhir *key* bernilai 0

Berdasarkan pengujian yang dilakukan, hasil enkripsi dari algoritme Grain menghasilkan sekumpulan karakter acak yang tidak bermakna. Dengan ini, proses enkripsi dari algoritme Grain dapat dikatakan *valid*.

Proses dekripsi melakukan proses xor antara bit-bit nilai *cipher* dan bit-bit nilai *keystream*. Berdasarkan pengujian yang dilakukan, hasil dari proses xor antara *cipher* dan *keystream* menghasilkan data yang sebelumnya diamankan. Dengan ini, proses dekripsi dari algoritme Grain dapat dikatakan *valid*.

6.1.2 Pengujian algoritme AES

Pengujian algoritme AES dilakukan dengan menguji proses enkripsi dan dekripsi. Proses enkripsi dikatakan *valid* jika data asli dapat dirubah menjadi sekumpulan karakter acak yang tidak bermakna. Proses dekripsi dikatakan *valid* jika data sebelum enkripsi bernilai sama dengan data setelah melakukan proses dekripsi. Prosedur pengujian akan melakukan proses enkripsi dan dekripsi terhadap nilai *keystream* yang

didapat saat menjalankan algoritme Grain. Selain itu, pengujian ini menerima masukan kunci dengan panjang maksimal 16 byte oleh *user*.

Data yang akan dienkripsi menggunakan algoritme AES pada pengujian ini adalah nilai *keystream* yang didapat dari proses algoritme Grain pada tahap sebelumnya. Hasil pengujian proses enkripsi dan dekripsi dari algortima AES dapat dilihat pada Tabel 6.14:

Tabel 6.14 Hasil pengujian proses enkripsi dan dekripsi AES

No	Data	Kunci	*Cipher*	*Plain*	Keterangan
1	7B978CF36846E5F4EE0BD4D68611156F45435A8720CCB762D118982E784CD6	KuNc1435…	0KTq952X9igtcx/gF14VUqMmU6GWSU6q+w7kZrcl1ZVVQb+3pOTZKxeHTdYpkS/1bpOgFq9YvwHOG5J6vU7X4A==	7B978CF36846E5F4EE0BD4D68611156F45435A8720CCB762D118982E784CD6	*Valid*
2	43A844CFDC613E640D73D1A7C5C257A19F1AB6077BD6E27B0A8EDDB199982C	KuNc1435…	iaCv+NBvbUNJqcApL5vlX0NMcUX9/XTYLC7WTJjyFGDCoWPrqoAoK9NV2L0BEooKK6ADEhPrPbYkmuag2XD0vA==	43A844CFDC613E640D73D1A7C5C257A19F1AB6077BD6E27B0A8EDDB199982C	*Valid*
3	368E0AB609601814567377FC11E0F86F111CA5B436C4BD441D12C8420558997	KuNc1435…	cYkw/Kaf3Mxog6EU+KuqwdU62kWA7vKeFvsm3W8iWHHzrslfbxGTK5WEB1rFLPdFVMUqdZm6aNUzH2pt9UIdcw==	7B978CF36846E5F4EE0BD4D68611156F45435A8720CCB762D118982E784CD6	*Valid*
4	FFFF7F00383F87D07F3969AE8293A740B2A979E6AE9D0FE49B01427915F502	KuNc1435…	YvVofWbfQ1nxlFAPHCEeAt1vNVfcMfitFEr40wJ3Ibgl5UGj+n1baC1m+OEw7iZtavt7VdFWxK/60JHm+5/ETg==	FFFF7F00383F87D07F3969AE8293A740B2A979E6AE9D0FE49B01427915F502	*Valid*
5	14963A7B255DB0FCC49F7B8DB333B425C08C96CB	KuNc1435…	zC6qqlaVYAo6SO2+ERq+w3yBAbWLWBdTuTNW9VdwutGQhVydWNLO0TIFUx	14963A7B255DB0FCC49F7B8DB333B425C08C96CB61B53C1D	*Valid*

	61B53C1D925FE2ABB36B66		GWgyLeL5Wr35oHxl0hzKg9KQbHJw==	925FE2ABB36B66	
6	6FA2A8CFC73128DF5012203EA17B9174E4693E8CC745EB80F3ADA52F1CD474	KuNc1435…	TSeACS3A76IN5PXg/kxrVb1DeO/iWyyPPIlUjg5G4zHT41yA7Pz2OLFssk1gJnqugC4+Sbx5nqQLGDESNxLENw==	6FA2A8CFC73128DF5012203EA17B9174E4693E8CC745EB80F3ADA52F1CD474	*Valid*
7	E7D307E37E7462CD87D6A8ADE7EC569F10455364E711A438C909C99A9247F0	KuNc1435…	4miPc99GQ4JehL/iS/MszT8Iz+AQwaMwCxosKdpbnaf5uC9mvuWvlO3T0PkpgXLWobdidq1eOeb7VfdRuhR2NQ==	E7D307E37E7462CD87D6A8ADE7EC569F10455364E711A438C909C99A9247F0	*Valid*
8	293143C0DF7ABC0280766CEB25994AA9DD013D6E066F26AEDA1D6CB0219FA2	KuNc1435…	U5TdlaA6tvOUt45+EXVNPWqIccFB2EUfaue+aKo8bykR0gEeqQ/jAiKjf6KzgmP8wQPmnfK+mfHKAEnHwxXcmw==	293143C0DF7ABC0280766CEB25994AA9DD013D6E066F26AEDA1D6CB0219FA2	*Valid*
9	45058AD2A75B7FF966334744C2A9725CF3655DC61CC6F814A4244C9CD10890	KuNc1435…	PrLtwfWTVlbKRqKi7SrnYaCUzv3uGM/gBfDdinEAuugZoMpIJ0YT7bBAgFqZ1Teurs8vGT+LLvjPEfzxWDi5UA==	45058AD2A75B7FF966334744C2A9725CF3655DC61CC6F814A4244C9CD10890	*Valid*

Kolom Data pada Tabel 6.14 adalah nilai *keystream* yang didapat dari proses Grain pada tahap sebelumya sedangkan kolom Kunci berisi nilai masukan kunci oleh *user*. Selanjutnya, data dan kunci ini akan melakukan serangkaian proses dalam algoritme AES. Hasil kombinasi data dan kunci akan menghasilkan nilai *cipher* yang dapat dilihat pada kolom *Cipher*. Percobaan dilakukan sebanyak 9 kali menggunakan nilai *keystream* yang didapat pada pengujian Grain sebelumnya sebagai data awal. Sedangkan kunci yang digunakan berisi nilai yang sama pada setiap percobaan yang dilakukan. Berdasarkan pengujian yang dilakukan, nilai dari *cipher* merupakan nilai yang berbeda dengan nilai pada kolom Data. Nilai pada *Cipher* juga terdiri dari

sekumpulan karakter acak yang tidak memiliki arti. Dengan ini, proses enkripsi dari algoritme AES dapat dikatakan *valid*.

Proses dekripsi pada algoritme AES menerima masukan kunci yang sama dengan kunci saat melakukan proses enkripsi. Berdasarkan pengujian yang dilakukan, kombinasi kunci dan *cipher* menghasilkan data yang sama sebelum melakukan proses enkripsi. Dengan ini, proses dekripsi dari algoritme AES dapat dikatakan *valid*.

6.1.3 Pengujian algoritme Shamir's Secret Sharing

Pengujian algoritme Shamir's Secret Sharing dilakukan dengan menguji proses *split* dan *reconstruct*. Proses *split* dapat dikatakan *valid* jika data berhasil terbagi menjadi 3 data dengan nilai yang berbeda. Proses *reconstruct* dapat dikatakan *valid* jika kombinasi antara 2 *share* dapat mengembalikan data asli seperti semula. Prosedur pengujian akan melakukan proses *split* dan *reconstruct* terhadap nilai *cipher* yang sebelumnya didapat pada proses algoritme Grain. Selain itu, pengujian ini menerima masukan bilangan bulat acak dan bilangan prima acak oleh *user*.

Data yang akan melakukan proses *split* dan *reconstruct* pada pengujian ini adalah data *cipher* yang didapat saat menjalankan algoritme Grain. Hasil pengujian *split* dapat dilihat pada Tabel 6.15 sedangkan hasil pengujian *reconstruct* dapat dilihat pada Tabel 6.16

Tabel 6.15 Hasil pengujian *split*

No	Data	*Random Number*	Mod	*Share* 1	*Share* 2	*Share* 3	Keterangan
1	2BD2DEB02704A4B5A02B959AC15E4726110E1BA76B9EFE328557DF7C390A9F	9875532121243657856465768786453245789	7742979417747111675973411307524180722629852644233417446154353268810639309 1	76DF55356D96DFF1CD2EFAAE202ABF9	EDBEAA6ADB2DBFE39A5DF55C4055B56	1649DFFA048C49FD5678CF00A6080AB3	*Valid*
2	12F0358F891156294C058CCFF552E1AD43CCABA1B3703D941BC169	9875532121243657856465768786453245789	3346111028245811070742196975412725311271382261622525158505721413087771840 1	76DF55356D96DFF1CD2EFAAE202AEA0	EDBEAA6ADB2DBFE39A5DF55C4055DFD	1649DFFA048C49FD5678CF00A6080D5A	*Valid*

	F67D5EC4						
3	66CB58F54622595518533E8D5940D4B845871A6327199D118563C37214CFDE	9875532121243657856465768786453245789	1816218534214633805143624573204900485751654058323851130157419829979400644423	76DF55356D96DFF1CD2EFAAE202AE94	EDBEAA6ADB2DBFE39A5DF55C4055DF1	1649DFFA048C49FD5678CF00A6080D4E	*Valid*
4	AFBA2D43777DC691311928E2C5DCF509E6E438C6E5CF46B4CF4E052B54B34B	9875532121243657856465768786453245789	3104831814593997403757363918658781117526203149627259921578083908656376513 81	76DF55356D96DFF1CD2EFAAE202AEF3	EDBEAA6ADB2DBFE39A5DF55C4055E50	1649DFFA048C49FD5678CF00A6080DAD	*Valid*
5	44D368386A1FF1BD8ABF3AC1F47CE66C94C1D7EB2AE7754DC610A5F9F22D2F	9875532121243657856465768786453245789	1216046786597018637540121180730076573923650563728350985942422344679692814 21	76DF55356D96DFF1CD2EFAAE202AEFF	EDBEAA6ADB2DBFE39A5DF55C4055E5C	1649DFFA048C49FD5678CF00A6080DB9	*Valid*
6	3FE7FA8C8873699E1E326172E634C33DB0247FAC8C17A2D0A7E2E27D5D923D	9875532121243657856465768786453245789	1129124232736670700533339866203879183434017139890698648844256317666291595 71	76DF55356D96DFF1CD2EFAAE202AF07	EDBEAA6ADB2DBFE39A5DF55C4055E64	1649DFFA048C49FD5678CF00A6080DC1	*Valid*

7	B79655A03136238CC9F6E9E1A0A304D644081244AC43ED689D468EC8D301B9	9875532121243657856465768786453245789	3243705832721407202876609405203849051261714086993634754270977577065866204497	76DF55356D96DFF1CD2EFAAE202AEC5	EDBEAA6ADB2DBFE39A5DF55C4055E22	1649DFFA048C49FD5678CF00A6080D7F	*Valid*
8	797411839038FD43CE562DA762D618E0894C7C4E4D3D6FFE8E522BE260D9EB	9875532121243657856465768786453245789	2145895695887455017841687128797857866783713546121502703734632815210120381133	76DF55356D96DFF1CD2EFAAE202AF53	EDBEAA6ADB2DBFE39A5DF55C4055EB0	1649DFFA048C49FD5678CF00A6080E0D	*Valid*
9	1540D891E8193EB828130608 85E62015A7281CE65794B144F06B0BCE904ED9	9875532121243657856465768786453245789	3755133884078289495418203906321724296500953981585950903609012942507450751111	76DF55356D96DFF1CD2EFAAE202AF3F	EDBEAA6ADB2DBFE39A5DF55C4055E9C	1649DFFA048C49FD5678CF00A6080DF9	*Valid*

Kolom Data pada tabel 6.15 adalah nilai *cipher* yang akan dipecah. Kolom *Random Number* adalah nilai bilangan bulat acak sedangkan kolom *Mod* adalah nilai bilangan prima. Percobaan dilakukan sebanyak 9 menggunakan nilai *cipher* pada pengujian sebelumnya. Nilai *random number* yang digunakan tetap sama dalam semua percobaan sedangkan bilangan prima yang digunakan pada setiap percobaan didapat dari nilai terbesar dan terdekat dari nilai desimal data *cipher*. Berdasarkan pengujian yang dilakukan, algoritme Shamir's Secret Sharing yang diterapkan dapat memecah nilai *cipher* pada kolom Data menjadi 3 data *share* yang berbeda. Ketiga data berbeda ini dapat dilihat masing-masing pada kolom *Share* 1, *Share* 2 dan *Share* 3. Dengan ini, proses *split* pada algoritme Shamir's Secret Sharing dapat dikatakan *valid*.

Tabel 6.16 Hasil pengujian *reconstruct*

No	Nilai *share* pertama	Nilai *share* kedua	Hasil	Keterangan
1	76DF55356D96DFF1CD2EFAAE202ABF9	EDBEAA6ADB2DBFE39A5DF55C4055B56	2BD2DEB02704A4B5A02B959AC15E4726110E1BA76B9EFE328557DF7C390A9F	*Valid*
2	76DF55356D96DFF1CD2EFAAE202ABF9	1649DFFA048C49FD5678CF00A6080AB3	2BD2DEB02704A4B5A02B959AC15E4726110E1BA76B9EFE328557DF7C390A9F	*Valid*
3	EDBEAA6ADB2DBFE39A5DF55C4055B56	1649DFFA048C49FD5678CF00A6080AB3	2BD2DEB02704A4B5A02B959AC15E4726110E1BA76B9EFE328557DF7C390A9F	*Valid*

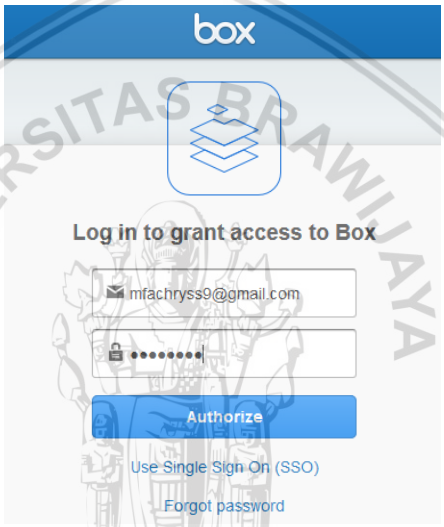
Proses *reconstruct* pada algoritme ini mengambil 2 data dari 3 data *share* yang sebelumnya didapat saat menjalankan proses *split*. Pengujian yang dilakukan akan mengembalikan nilai *cipher* pada pengujian algoritme Grain yang memiliki nilai *key* dan IV sesuai dengan *test vector*. Ada 3 percobaan yang dilakukan. Ketiga percobaan ini merupakan ketiga kombinasi yang dapat dilakukan ketika mengembalikan data. Kolom nilai *share* pertama dan kedua masing-masing merupakan 2 nilai yang digunakan untuk mengembalikan data. Percobaan nomor 1 menggunakan nilai *share* 1 dan nilai *share* 2, percobaan nomor 2 menggunakan nilai *share* 1 dan *share* 3 dan percobaan 3 menggunakan nilai *share* 2 dan *share* 3. Berdasarkan pengujian yang dilakukan, 3 jenis kombinasi yang mungkin dilakukan pada sistem berhasil mengembalikan data yang sama sebelum melakukan proses *split*. Dengan ini, proses *reconstruct* pada algoritme Shamir's Secret Sharing dapat dikatakan *valid*.

6.2 Pengujian layanan *cloud*

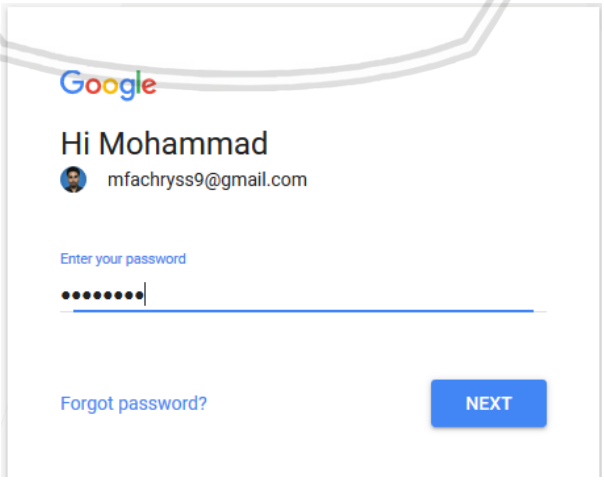
Pengujian pada layanan *cloud* dilakukan dengan menguji fungsi-fungsi layanan *cloud* yang terdiri dari fungsi *login*, fungsi *upload* dan fungsi *download*. Fungsi *login* dilakukan setiap saat ingin menjalankan fungsi-fungsi yang terdapat pada layanan *cloud* yang digunakan sistem. Fungsi *upload* dijankan saat melakukan proses *upload* terhadap 3 data *share* pada 3 *cloud* yang berbeda. Fungsi *download* dijalankan saat melakukan proses *download* terhadap 2 data *share* pada masing-masing *cloud*.

Prosedur pengujian akan menjalankan masing-masing fungsi dan melihat hasil yang terjadi setelah setiap fungsi dijalankan.

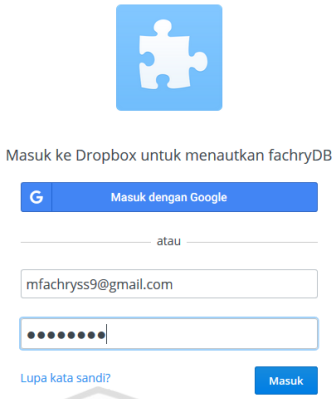
Ketika fungsi *login* pada masing-masing *cloud* dijalankan, maka akan muncul tampilan jendela untuk memasukkan *username* dan *password*. Kedua masukkan ini berfungsi untuk menjembatani sistem dengan masing-masing *cloud service provider*. Berdasarkan pengujian yang dilakukan, fungsi *login* pada setiap *cloud service provider* dapat berjalan dengan baik. Tampilan jendela autentifikasi setiap *cloud* dapat dilihat pada Gambar 6.22 untuk Box, Gambar 6.23 untuk Google Drive dan Gambar 6.24 untuk Dropbox.



Gambar 6.22 Jendela autentifikasi Box

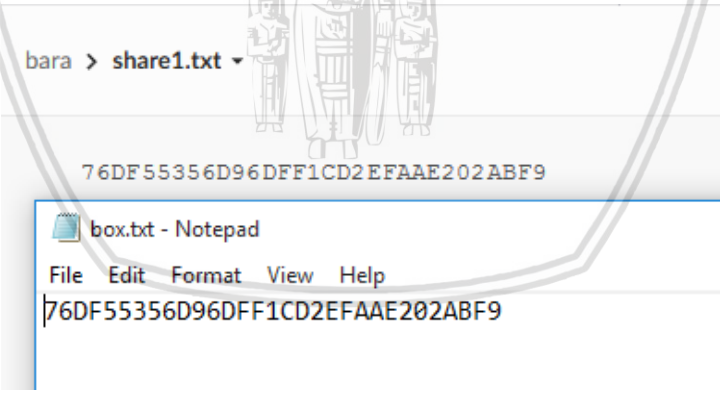


Gambar 6.23 Jendela autentifikasi Google Drive

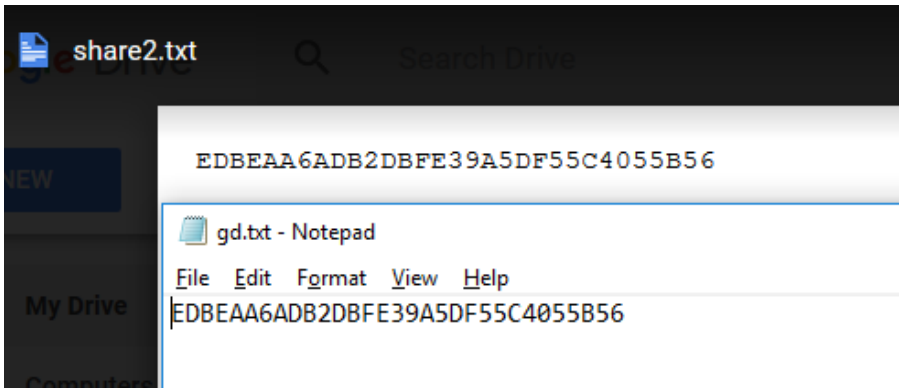


Gambar 6.24 Jendela autentifikasi Dropbox

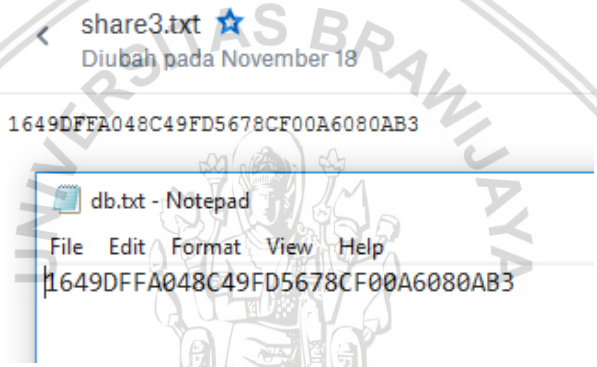
Ketika fungsi *upload* dijalankan, maka sistem akan melakukan proses *upload* terhadap 3 data *share* dan menyimpannya pada 3 *cloud* yang berbeda. Box akan menyimpan data *share* 1, Google Drive akan menyimpan data *share* 2 dan Dropbox akan menyimpan data *share* 3. Proses *download* akan mengunduh data *share* yang telah tersimpan pada masing-masing *cloud* ketika dijalankan. Berdasarkan pengujian yang dilakukan, data *share* berhasil tersimpan dan dapat diakses pada masing-masing *cloud*. Hasil dari *file upload* dan *download* dapat dilihat pada Gambar 2.26 untuk Box, Gambar 2.27 untuk Google Drive dan Gambar 2.28 untuk Dropbox.



Gambar 6.25 *File share* 1 (Box)



Gambar 6.26 *File share* 2 (Google Drive)



Gambar 6.27 *File share* 3 (Dropbox)

BAB 7 PENUTUP

Bab ini akan berisi kesimpulan dari penelitian dan saran untuk pengembangan lebih lanjut.

7.1 Kesimpulan

Adapun kesimpulan yang dapat ditarik dari penelitian skripsi ini antara lain:

1. Penerapan teknik enkripsi dan *secret sharing* yang dilakukan pada penelitian ini bertujuan untuk menjaga informasi dari pihak yang tidak diinginkan. Berdasarkan pengujian yang dilakukan, penerapan teknik enkripsi dan *secret sharing* berhasil merubah informasi atau data yang memiliki nilai menjadi sebuah data atau informasi yang sulit untuk dibaca.

7.2 Saran

Adapun saran yang dapat dituliskan dari penelitian ini antara lain:

1. Dalam meningkatkan keamanan pada sebuah data, diperlukan penambahan aspek-aspek keamanan untuk penerapannya pada sistem. Aspek-aspek yang bisa ditambahkan antara lain *availability*, *integrity*, *non-repudation*, *authority* dan *access control*.
2. Perlu dilakukannya penelitian lebih lanjut pada sistem dengan menggunakan kombinasi-kombinasi dari algoritme kriptografi yang lain.

DAFTAR PUSTAKA

AlJadaani, S., AlMaliki, M., AlGhamdi, W., 2016. Security issues in cloud computing. *International journal of applied engineering research*, 11 (12), pp. 7669-7671.

Ariyus, D., 2008. *Pengantar ilmu kriptografi: teori, analisis dan implementasi.* Yogyakarta: ANDI.

Beimel, A., 2007. Secret-sharing schemes: a survey. *International conference on coding and cryptology*, pp. 11-46.

Box, 2017. *About us | Box*. [online] Tersedia di: <https://www.box.com/about-us> [Diakses 17 Juli 2017]

CloudRail, 2017. *CloudRail API integration solution*. [online] Tersedia di: <https://CloudRail.com/features> [Diakses 17 Juli 2017]

Daemen, J., Rijmen, V., 2002. *The design of rijndael, AES – the advanced encryption standard*. Springer-Verlag.

Hell, M., Johannson, T., & Meier, W., 2007. Grain – a stream cipher for constrained environments. *International journal of wireless and mobile computing.*

Jacobson, D., Brail, G., & Woods, D., 2012. *APIs a strategy guide.* [e-book]. Sebastopol: O'Reilly Media Inc.

Kromodimoeljo, S., 2009. *Teori dan aplikasi kriptografi.* [e-book]. SPK IT Consulting.

Kurniawan, Yusuf., 2004. *Kriptografi: Keamanan internet dan jaringan telekomunikasi*. Bandung: Informatika Bandung.

Nijim, S., & Pagano, B., 2014. *APIs for dummies*. [e-book]. Hoboken: John Wiley & Sons, Inc.

Nithya, B., & Sripiya, P., 2016. A review of cryptographic algorithm in network security. *International journal of engineering and technology*, pp. 324-331.

Rouse, M., 2011. *What is Dropbox? | Techtarget.* [online] Tersedia di: <http://searchmobilecomputing.techtarget.com/definition/Dropbox> [Diakses 17 Juli 2017]

Rouse, M., 2011. *What is Google Drive? | Techtarget.* [online] Tersedia di: <http://searchmobilecomputing.techtarget.com/definition/Google-Drive> [Diakses 17 Juli 2017]

Schildt, H. 2007. *Java: the complete reference*. [e-book]. New York: McGraw-Hill Education.

Shamir, A., 1979. How to share a secret. *Communications of the ACM*, 22 (11), pp. 612-613.

Sonatype., 2008. *Maven the definitive guide.* [e-book]. Sebastopol: O'Reilly Media, Inc.

Waloeyo, Y. J., 2012. *Cloud computing – aplikasi berbasis web yang mengubah cara kerja dan kolaborasi anda secara online*. Yogyakarta: ANDI.

Wibowo, S., 2016. S*istem pengamanan data pada media penyimpanan cloud dengan metode split data distribution*. S1. Universitas Brawijaya.