PENGGUNAAN MULTIPLE KRIPTOGRAFI DAN STEGANOGRAFI BERBASIS ANDROID UNTUK PENYEMBUNYIAN PESAN TEKS PADA CITRA DIGITAL

Disusun Oleh:

N a m a : M Agus Khamsinindo

NIM : 13523253

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2020

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGGUNAAN MULTIPLE KRIPTOGRAFI DAN STEGANOGRAFI BERBASIS ANDROID UNTUK PENYEMBUNYIAN PESAN TEKS PADA CITRA DIGITAL

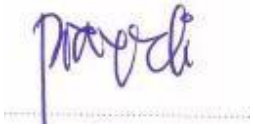
TUGAS AKHIR

Disusun Oleh:

N a m a	: M Agus Khamsinindo
NIM	: 13523253

Yogyakarta, 14 Mei 2020

Pembimbing,



(Yudi Prayudi, Dr., S.Si., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGGUNAAN MULTIPLE KRIPTOGRAFI DAN STEGANOGRAFI BERBASIS ANDROID UNTUK PENYEMBUNYIAN PESAN TEKS PADA CITRA DIGITAL

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 14 Mei 2020

Tim Penguji

Yudi Prayudi, Dr., S.Si., M.Kom. _____

Anggota 1
Ahmad M. Raf'ie Pratama, S.T., M.I.T., Ph.D. _____

Anggota 2

Septia Rani, S.T., M.Cs. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : M Agus Khamsinindo

NIM : 13523253

Tugas akhir dengan judul:

PENGGUNAAN MULTIPLE KRIPTOGRAFI DAN STEGANOGRAFI BERBASIS ANDROID UNTUK PENYEMBUNYIAN PESAN TEKS PADA CITRA DIGITAL

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 14 Mei 2020

(M Agus Khamsinindo)

HALAMAN PERSEMBAHAN

Seluruh proses pembuatan dan hasil pencapaian tugas akhir ini, penelitian ini persembahkan kepada ALLAH SWT yang selalu menjadi patokan iman dalam kebodohan pada saat kekhawatiran tidak baik, kedua orang tua yang saling support dan selalu mendukung untuk menyelesaikan jenjang kuliah dan segera memasuki jenjang kehidupan selanjut, untuk abang – abang dan teman – teman saya yang selalu mengingatkan dan membantu saya. Tanpa dukungan orang terdekat dan Tuhan pencipta alam semesta ini dalam memberikan energy positif, penelitian ini tidak akan sampai pada titik sekarang ini

HALAMAN MOTO

Mulai aja dulu, pasti ada jalan

(Go Jek)

Tidak masalah sebarapa lambat kau berjalan, asalkan kau tidak berhenti

(Confucius)

Katakanlah pada hatimu. Kalau takut gagal, justru lebih buruk dari kegagalan itu sendiri

(Paulo Coelho)

Tidak usah takut gagal. Bekerjalah semaksimal mungkin dan percayalah, bahwa semua jerih payah kita akan diperhitungkan oleh tuhan

(Merry Riana)

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Assalamu'alaikum Warahmatullahi Wabarakatuh

Alhamdulillah, penulis panjatkan kehadirat Allah SWT yang telah memberikan rahmat, hidayah, dan karunia-Nya, sehingga laporan tugas akhir ini dapat penulis selesaikan. Tak lupa shalawat dan salam penulis haturkan kepada junjungan kita Nabi Muhammad SAW.

Tugas Akhir ini dibuat sebagai salah satu syarat yang harus dipenuhi untuk memperoleh gelar sarjana di Jurusan Informatika Universitas Islam Indonesia. Adapun Tugas Akhir yang dilakukan penulis merupakan Aplikasi Penggunaan Multiple Kriptografi dan Steganografi Berbasis Android Untuk Penyembunyian Teks pada Citra Digital.

Pelaksanaan Tugas Akhir ini merupakan salah satu mata kuliah wajib jurusan Informatika Fakultas Teknologi Industri Universitas Islam Indonesia. Selain itu merupakan sarana bagi penulis untuk menambah wawasan serta pengalaman dalam menerapkan keilmuan, sesuai dengan yang diambil di bangku perkuliahan.

Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih kepada:

1. Kedua orang tua atas segala doa dan dukungan selama penulis menyelesaikan Tugas Akhir.
2. Fathul Wahid, selaku Rektor Universitas Islam Indonesia
3. Prof. Dr. Ir. Hari Purnomo, M.T., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Dr. Raden Teduh Dirgahayu, S.T., M.Sc., selaku Ketua Jurusan Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Yudi Prayudi, Dr., S.Si., M.Kom., selaku Dosen Pembimbing Tugas Akhir di Jurusan Informatika
6. Cahyo, Argo, Anak Kontrakan Terutama Sulistyo Anggara, dan Abang – abang saya, yang selalu support mendukung saya dalam keterpurukan pada saat mengerjakan Tugas Akhir.
7. Semuah pihak yang tidak bisa saya sebut satu persatu dalam membantu dan menjadikan salah satu sumber terpenting dalam penulisan dan pembuatan Tugas Akhir

Penulis menyadari bahwa laporan ini masih belum sempurna karena keterbatasan kemampuan dan pengalaman penulis di lapangan. Oleh karena itu, penulis mengharapkan kritik

dan saran yang membangun demi kesempurnaan Laporan Tugas Akhir ini. Akhir kata, penulis berharap agar laporan ini dapat bermanfaat bagi semua pihak.

Wassalamu'alaikum Warahmatullahi Wabarakatuh

Yogyakarta, 14 Mei 2020

(M Agus Khamsinindo)

SARI

Perkembangan Teknologi yang semakin canggih, mengakibatkan kenyamanan bagi pengguna dalam memberikan segala informasi tanpa mengetahui ancaman – ancaman yang ada dibelakangnya, sehingga menjadikan sebuah kerugian dari perkembangan teknologi tersebut. Maka dari itu, dibutuhkan sebuah aplikasi yang memiliki aspek dalam segi keamanan teknologi dalam penyampain dan penerimaan informasi antara kepentingan diri sendiri maupun kepentingan kedua belah pihak.

Dengan sebab itu, dibutuhkannya sebuah penarapan dari ilmu keamanan informasi yang dinamakan kriptografi dan Steganografi yang diimplementasikan ke dalam sebuah perangkat *hardware* dan software. Banyak metode kriptografi yang dapat digunakan yaitu kriptografi klasik ataupun kriptografi modern. Pada penelitian ini mengkombinasikan kriptografi klasik dan modern yang dimana metode tersebut yaitu *affine cipher, hill cipher, caesar cipher,* dan *advanced encrypt standard* (AES). Sedangkan untuk steganografi, penelitian menggunakan metode *least significant bit* (LSB).

Dari hasil penelitian yang diperoleh, menunjukan dengan menambah proses perhitungan pada kriptografi klasik dan mengkombinasikan dengan kriptografi modern memperkuat keamanan informasi. Ke empat metode tersebut dapat dikombinasikan menjadi satu dalam proses kriptografi untuk menutupi kelemahan – kelemahan dari metode tersebut. Serta dengan melakukan proses kombinasi antara kriptografi dan steganografi, memberikan sebuah hasil dari segi keamanan pada pengelihatan, sehingga data informasi yang berlalu lalang bebas tidak memberikan kecurigaan sama sekali.

Sistem yang dibangun pada penelitian ini, merupakan sebuah aplikasi berbasis android yang mempermudah dalam memberikan informasi.

Kata kunci : *keamanan informasi, software, aplikasi kriptografi dan steganografi, algoritma affine cipher, algoritma hill cipher, algoritma caesar cipher, algoritma advanced encrypt standard, least significant bit (LSB), android*

GLOSARIUM

Source Code	Sebuah rangkaian deklarasi yang ditulis dalam bahasa pemrograman komputer yang dapat dibaca manusia
Hexadecimal/hex	Sistem bilangan yang menggunakan simbol dengan Urutan 1 – 15, yang dimana angka 10 – 15 digantikan dengan karakter A,B,C,D,E,F
Embedding	Suatu proses yang memasukan sesuatu data ke dalam sebuah objek yang dapat digunakan
Extracting	Suatu proses yang mengeluarkan data dari dalam sebuah objek untuk dapat digunakan lagi
Plaintext	Sebuah masukan pesan/kalimat yang berupa kumpulan dari karakter – karakter huruf dan simbol untuk digunakan sebagai bahan objek penyandian pada ilmu kriptografi
ciphertext	Sebuah kumpulan karakter – karakter acak yang berupa huruf dan simbol hasil dari proses penyandian kriptografi

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi menyebabkan perubahan sebuah zaman, yang dimana segala sesuatu dipermudahkan oleh sebuah sarana teknologi, sehingga perubahaan pola pikir dalam tingkah laku suatu kegiatan ikut berubah. Sebelum terjadi perkembangan yang pesat ini dalam mendapatkan suatu informasi, manusia berinteraksi hanya sebatas orang terdekat maupun keluarga terdekat saja, akan tetapi dengan seiring perkembangan teknologi yaitu sebuah internet. Sekarang dalam berinteraksi dan mencari informasi tentang keluarga maupun orang yang jauh dapat dengan mudah.

Dengan adanya teknologi internet, perkembangan alat penggunapun ikut berubah. Yang dimana pada awalnya, alat yang digunakan tidak bisa dibawa kemana – mana, sekarang alat tersebut dapat digunakan dimana saja, yaitu sebuah *handphone*. Dengan adanya perubahan alat tersebut. Memberikan sebuah kemudahan bagi penggunanya dalam memberikan informasi seperti data rahasia, bisnis, berita ataupun yang lainnya dengan mengirimkan informasi tersebut melalui aplikasi bawaan maupun aplikasi download, yaitu SMS, *Bluetooth*, *Gmail* dan media sosial.

Dampak baik dari sebuah perkembangan teknologi, pasti memiliki dampak negatif yang diberikan juga, yaitu sebuah informasi yang mudah berlalu lalang bebas dapat dijadikan sebuah target berbahaya bagi seseorang yang ingin informasinya aman sampai tujuan. Kejahatan ini merupakan perubahan prilaku manusia untuk mencari celah suatu perkembangan teknologi, yang digunakan untuk kepentingan pribadi maupun kepentingan kelompok dalam mencari keuntungan semata. Salah satu contoh berita tentang kejahatan dunia maya (Cyber crime), belakangan ini yaitu peretasan informasi media sosial dan Whatsapp sejumlah tokoh politik (Simanjuntak 2019). Dengan hal ini merupakan suatu indikasi keamanan informasi merupakan suatu hal yang sangat penting untuk diperhatikan, sehingga mencegahnya terjadi kejahatan – kejahatan dalam hal penyadapan, peretasan, dan penyerangan suatu informasi.

Untuk mencegah terjadinya kejahatan dalam peretasan dan panyadapan informasi yang dikirim maupun tersimpan didalam sebuah teknologi dari pihak yang tidak bertanggung jawab dalam mencari keuntungan semata. Maka teori tentang keamanan data dapat digunakan untuk mencegahnya yaitu dengan ilmu kriptografi, fungsinya untuk mengacak sebuah informasi dan

untuk memperkuat ilmu tersebut terhindar dari orang lain yaitu menyembunyikan informasi ke dalam suatu objek, sehingga orang lain tidak mengetahui isi informasi tersebut. Sehingga ilmu steganografi merupakan kombinasi yang baik dengan ilmu kriptografi.

Kriptografi merupakan sebuah ilmu keamanan yang berfungsi untuk mengubah informasi menjadi kode – kode yang tidak bisa dipahami oleh orang lain, sedangkan ilmu kriptografi terdapat 2 algoritma yaitu *simetris* dan *asimetris*. Dalam penelitian ini proses enkripsi menggunakan algoritma *simetris*. Algoritma *simetris* ini sipenerima pesan harus diberikan kunci dari pesan tersebut agar bisa mendeskripsikan pesan yang dikirim (Ariyus 2006), Sehingga kemanan pesan dari algoritma ini tergantung pada kunci. Untuk mencegah kekurangan tersebut, maka penelitian ini membuat beberapa konsep yaitu menggabungkan 4 metode kriptografi simetris, yang dimana 1 metode digunakan untuk mengenkripsi kunci yang digunakan dari salah 1 metode tersebut. Sehingga ruang kunci yang digunakan besar.

Metode algoritma *caesar cipher* yang digunakan untuk mengenkripsi kunci dari salah 1 metode tersebut. Algoritma ini merupakan proses pengubahan tulisan dengan menggantikan posisi huruf dari tabel *alphabet*. Kebanyakan penelitian – penelitian sebelumnya menggunakan 25 karakter *alphabet*. Untuk itu penelitian ini, ingin menggunakan salah satu dari penelitian terdahulu yang sudah dimodifikasi, yaitu dari penelitian (Latifah, Ambo, and Kurnia 2017) didalam penelitiannya memodifikasi *caesar cipher* menggunakan 95 karakter *alphabet*, sehingga enkripsi yang digunakan dapat digunakan untuk menampilkan karakter lebih banyak. Dengan itu maka dapat dipergunakan untuk memperkuat dijadikan kunci enkripsi.

Dalam proses enkripsi informasi, ada 3 metode dalam penelitian ini yang akan digunakan yaitu *Affine cipher*, *Hill cipher*, *Advanced Encryption Standard* (AES). Menurut (Zuli and Irawan 2014) dalam penelitiannya menerapakan kombinasi sandi Caesar dan Vigenere, meyakini bahwa dalam menggabungkan konsep kriptografi memperkuat keamanan, sehingga terhindar dari kriptanalisis. (Juliadi, Prihandono, and Kusumastuti 2013) dalam penelitiannya menggunakan metode modifikasi *affine cipher* yang diperkuat vigenere, menghasilkan suatu percobaan bahwa dengan semakin banyaknya tabel *alphabet* kunci yang dingunakan, menyulitkan kriptanalisis untuk mengetahui kuncinya, karena harus mengetahui nilai kunci pertama dan kunci kedua (Invers Modular) lalu harus mengetahui kunci ke tiga yaitu kunci dari metode *vigenere*. (Barmawi and Hamdani 2016) menggunakan Implementasi *hill cipher* pada Sistem pengamanan dokumen, dalam penilitiannya melakukan enkripsinya menggunakan *hill cipher* dengan matriks 2 x 2, akan tetapi menyarankan menggunakan Matriks Ordo selain 2 x 2 dalam enkripsi untuk memperkuat hasil *ciphertext*. Penelitian (Abidin, Hardianti, and Setiano

2016) yaitu mengimplementasi dan menganalisis proses kriptografi *Advanced Encryption Standard* (AES), dalam kesimpulannya bahwa meskipun algoritma metode tersebar luas tidak akan dapat membongkar data tanpa kunci yang tepat. Sehingga dalam penelitian yang dia lakukan, menjadikan penilitian ini untuk menggabungkan antara *Caesar Cipher* dan *Advanced Encryption Standard*.

Steganography merupakan ilmu mempelajari menyembunyikan suatu informasi ke dalam sebuah objek digital. Menurut (Ariyus 2006) perbedaan Steganografi dengan Kriptografi yaitu bagaimana proses penyembunyian data dan hasil akhir dari proses tersebut sedangkan kriptografi merupakan proses pengacakan suatu data. Keuntungan dalam penyembunyian menggunakan teknik steganografi yaitu objek gambar, yang dimana sering kali digunakan untuk memberikan informasi kepada orang lain ataupun urusan pribadi. Sehingga penelitian ini, menjadikan sebuah topik untuk menggunakan steganografi dalam memperkuat keamanan informasi untuk terhindar dari kecurigaan orang lain.

Metode yang digunakan untuk menyembunyikan pesan tersebut yaitu Steganografi Least Significant Bit (LSB). Metode ini merupakan metode yang banyak diterapkan sehingga sederhana dalam pembuatan dan metode ini juga dijadikan sebuah konsep untuk mengelabuhi orang karena meskipun teknik sederhana akan tetapi jika digunakan secara benar maka bisa menipu mata normal kita. Steganografi LSB ini menyembunyikan pesan ke dalam sebuah gambar yang dimana gambar tersebut dijadikan wadah dalam penyembunyiannya. Menurut (Anwar 2017) dalam peniltiannya mengimplementasikan pengamanan data dan informasi dengan metode steganografi LSB dan algoritma Kriptografi AES hasil pengujian bahwa metode LSB memenuhi aspek imperceptibility, dimana keberadaan pesan rahasia pada citra sulit untuk dipersepsi oleh inderawi.

Berdasarkan Latar belakang tersebut, maka akan dilakukan penelitian yang menggabungkan algoritma kriptografi Simetris yaitu Caesar Cipher, Affine Cipher, Hill Cipher, Advanced Encryption Standard (AES) dan Steganografi LSB. Untuk pembuatan aplikasi ini menggunakan Android sehingga penamaan penilitian ini "Penggunaan Multiple Kriptografi dan Steganografi Berbasis Android Untuk Penyembunyian Pesan Teks Pada Citra Digital".

1.2 Rumusan Masalah

Berdasarkan penjelasan dari latar belakang diatas, Maka penelitian ini merumuskan beberapa masalah sebagi berikut:

a. Merancang dan mengimplementasikan beberapa metode kriptografi yang dapat dikombinasikan untuk memperkuat hasil yang didapat
b. Membuat sebuah aplikasi yang digunakan untuk mengamankan pesan dengan menggunakan metode kriptografi dan steganografi

1.3 Batasan Masalah

Dalam proses pembuatan aplikasi tersebut, penelitian ini membuat batasan dalam aplikasi tersebut antara lain:

a. Dalam pengenkripsian menggunakan metode AES 128 menggunakan kunci 16 karakter
b. Dalam proses pembuatan agar tidak menyimpang kriptografi yang digunakan hanya sandi hill, sandi affine, Caesar Cipher, dan sandi AES, Tidak menggunakan sandi lain
c. Hill Cipher menggunakan matriks 4 x 4 dengan kunci tetap
d. Kunci AES didapat dari Caesar Cipher.
e. Pesan Yang disembunyikan adalah pesan teks
f. Wadah/stego objek pesan adalah gambar PNG dan JPEG
g. Metode Steganografi menggunakan metode LSB sekuensial
h. Implementasi kriptografi dan steganography berbasis android dengan menggunakan Bahasa pemrograman java

1.4 Tujuan

Dengan melakukan penelitian ini, diharapkan dapat memberikan beberapa manfaat dengan rincian sebagai berikut :

a. Memberikan konsep kombinasi enkripsi simetris bagi penulis maupun pembaca sehingga bermanfaat bagi siapapun
b. Mengamankan pesan teks yang bersifat rahasia dalam file citra dengan mengkombinasikan beberapa metode kriptografi dan steganografi untuk keamanan pesan
c. Membuat aplikasi penyembunyian pesan teks berbasis android yang mudah untuk digunakan

1.5 Manfaat Penelitian

Dengan melakukan penelitian ini, diharapkan dapat memberikan beberapa manfaat yang dapat digunakan, yaitu:

a. Dapat menyembunyikan pesan ke dalam file objek citra sehingga dapat mengelabuhi orang lain
b. Dapat digunakan sebagai referensi bacaan untuk semua orang yang ingin menggunakan atau mempelajari kriptografi maupun steganography.
c. Dapat meningkatkan keamanan suatu pesan rahasia.
d. Mendapatkan ilmu tentang penerapan dan teori dari kriptografi dan steganografi dalam melakukan pengamanan dan penyembunyian

1.6 Metode Penelitian

Penelitian ini dilakukan dengan langkah – langkah sebagai berikut :

a. Studi literatur

   Dengan Mencari sumber – sumber referensi yang berkaitan dengan penelitian yang akan dilakukan guna membantu dalam penelitian. berupa buku, jurnal, ebook, makalah, dan serta berbagai referensi lain.

b. Analisis Data

   Dalam tahap ini mengkaji dan menganalisis hasil data yang didapat dan kemudian dilakukan analisis hasil studi literature yang diperoleh

c. Perancangan Perangkat Lunak

   Dalam tahap ini penelitian membuat sebuah perancangan perangkat lunak aplikasi yang akan dibuat untuk berfungsi sebagai penataan fungsi aplikasi

d. Implementasi Sistem

   Dalam tahap ini akan dilakukan pengimplementasian dari hasil perancangan desain dan system untuk mengetahui kelayakan hasil apakah sudah layak digunakan

e. Penyusunan Laporan

   Pada tahap ini dilakukan penyusunan laporan dari analisis dan perancangan system yang sudah dibuat

1.7 Sistematika Penulisan

Sistematika penulisan penelitian ini disusun sebagai gambaran umum untuk memudahkan dalam memahami penelitian di lakukan sehingga membantu dalam pembuatan laporan. Secara garis besar, sistematika penulisan pada penelitian ini sebagai berikut :

a. BAB I : PENDAHULUAN

   Pada bab ini dijelaskan mengenai latar belakang, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan

b. BAB II : LANDASAN TEORI

   Pada bab ini menjelaskan teori – teori yang digunakan sebagai suatu landasan penelitian yang dibuat untuk digunakan untuk menyelesaikan permasalahan yang dibuat pada penelitian ini, Bahasan dalam penelitian tentang algoritma Kriptografi dan steganography dengan metode – metodenya yang akan digunakan

c. BAB III : METODOLOGI DAN PERANCANGAN

   Pada bab ini berisi penjelasan metode yang akan digunakan dalam penelitian serta dalam pada bab ini berisi juga penjelasan mengenai rancangan dan kebutuhan dalam penelitian ini

d. BAB IV : IMPLEMENTASI DAN PENGUJIAN

   Pada bab ini hasil dari implementasi penelitian dan pembuatan system yang sudah dilakukan berdasarkan pada perancangan yang sudah di buat pada bab sebelumnya dan bab ini membahas hasil pengujian untuk mengetahui hasil yang dilakukan system berhasil apa tidak

e. BAB V : KESIMPULAN DAN SARAN

   Pada bab ini akan menjelaskan sebuah kesimpulan dari penelitian yang sudah dilakukan dan akan diberikan juga saran untuk mengetahui kekurangan dalam penelitian sehingga dapat dikembangkan lagi

BAB II
LANDASAN TEORI

2.1 Kriptografi

Kriptografi sebuah ilmu yang memberikan pengetahuan tentang penyandian suatu pesan yang dipergunakan pada peperangan romawi dulu untuk menghindarkan orang lain tidak mengetahui isi dari informasi tersebut. Menurut (Ariyus 2006) mengatakan bahwa kriptografi berasal dari Bahasa yunani, menurut Bahasa dibagi menjadi dua yaitu kripto dan graphia, kripto berarti rahasia (secret) dan graphi berarti tulisan (Writing). Menurut terminologinya kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan ketika pesan dikirim suatu tempat ke tempat lain. Sehingga kriptografi yang dipergunakan pada perang modern ini berfungsi untuk menjaga kerahasian dibidang militer.

Seiring berjalannya waktu kriptografi menjadikan ilmu untuk mempertahankan diri untuk menjaga kerahasiaan seperti yang dikatakan (Kurniawan 2004) mengatakan kriptografi bukan lagi monopolo militer, setiap individu berhak mengamankan komunikasinya tanpa kuatir dimata – matai pihak lain. Setiap individu berhak melindungi komunikasi berisi rahasia keluarga, bisnisnya, perkerjaannya, dan pendapat – pendapatnya.

Kriptografi memiliki sistematis dalam algoritma pengenkripsian dan pendekripsian pesan yang dimana menurut pandangan (Sadikin 2012) sistem kriptografi terdiri dari 5 bagian yaitu :

a. Plaintext

pesan atau data dalam bentuk aslinya yang dapat terbaca. *Plaintext* adalah masukan bagi algoritma enkripsi untuk selanjutnya digunakan istilah teks asli sebagai padanan kata plaintext

b. Secret Key

secret key yang juga merupakan masukan bagi algoritma enkripsi merupakan nilai yang bebas terhadap teks asli dan menentukan hasil keluaran algoritma enkripsi. Untuk selanjutnya digunakan istila kunci rahasia sebagai padanan kata *secret key*

c. Ciphertext

ciphertext adalah keluaran algoritma enkripsi. *Ciphertext* dapat digunakan sebagai pesan dalam bentuk tersembunyi. Algoritma enkripsi yang baik akan menghasilkan ciphertext

yang terlihat acak untuk selanjutnya digunakan istilah teks sandi sebagai padanan kata ciphertext

d. Algoritma Enkripsi

algoritma enkripsi memiliki 2 masukan teks asli dan kunci rahasia. Algoritma enkripsi melakukan transformasi terhadap teks asli sehingga menghasilkan teks sandi

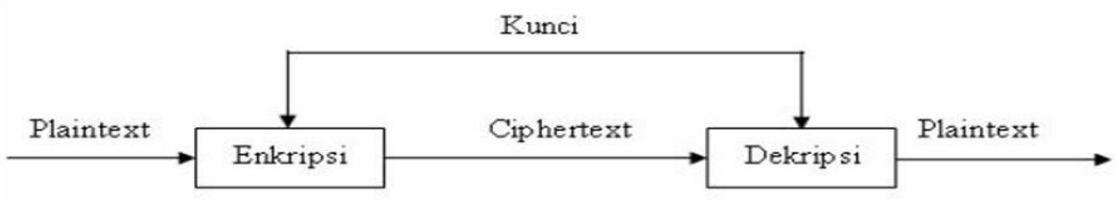
e. Algoritma Dekripsi

algoritma dekripsi memiliki 2 masukan yaitu teks sandi dan kunci rahasia. Algoritma dekripsi memulihkan kembali teks sandi menjadi teks asli bila kunci rahasia dipakai algoritma dekripsi sama dengan kunci rahasia yang dipakai algoritma enkripsi.

Kriptografi yang sudah dimodernisasi mengandalkan pada ilmu matematika untuk melakukakan penyandian sebuah pesan menjadi kata sandi yang dimana ada sebuah algoritma – algoritma yang dipergunakan dalam ilmu Kriptografi Klasik dan Kriptografi Modern. Menurut (Kurniawan 2004), Ada 2 jenis algoritma dalam kriptografi yang dipergunakan yaitu

a. Algoritma Simetris

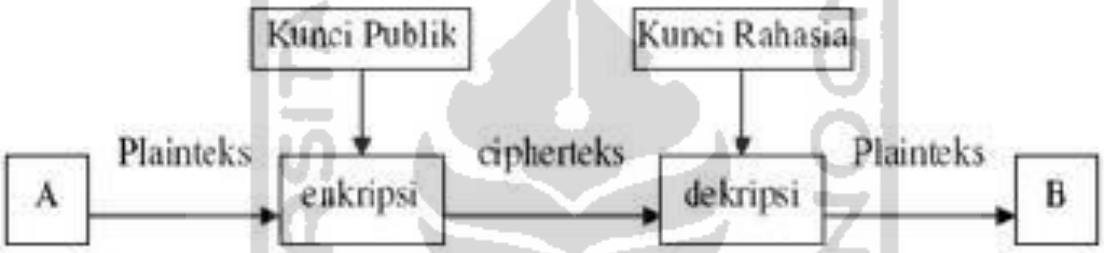
Algoritma Simetris biasanya disebut sebagai algoritma yang sering digunakan (Konvensional). Algoritma simetris merupakan penyandian yang menggunakan satu kunci private untuk mengenkripsi dan mengdekripsi. Kunci Private dalam algoritma simetris bisa disebut sebagai kunci rahasia ataupun kunci tunggal yang memiliki fungsi sebagai penyandian, yang dimana kunci private tersebut dapat melakukan pengenkripsian dan pendekripsian yang dapat menghasilkan pesan asli menjadi pesan sandi maupun sebaliknya pesan sandi menjadi pesan asli. Kunci Private dalam algoritma simetris, pengirim dan penerima harus mengetahui kunci yang digunakan bersama sebelum mereka melakukan penyandian dalam berkomunikasi



Gambar 2.1 Ilustrasi alur proses algoritma simetris

b. Algoritma Asimetris

Algoritma Asimetris bisa disebut algoritma dengan kunci public, yang dimana proses penyandiannya berbeda dengan algoritma simetris. Algoritma asimetris dalam pengenkripsian dan pengdekripsian memiliki 2 kunci yang berbeda dan saling terhubung satu sama lain, yaitu kunci private dan kunci public. Kunci public biasa disebut kunci yang orang lain dapat mengetahuinya sedangkan kunci private merupakan kunci yang orang lain tidak boleh mengetahuinya dan kunci yang hanya diketahui oleh satu orang saja. Kunci public pada algoritma asimetris berfungsi sebagai penyandian/pengenkripsian yang melakukan proses pengubahan pesan asli menjadi pesan sandi sedangkan kunci private digunakan untuk melakukan pengdekripsian yang dimana pesan sandi menjadi pesan asli.



Gambar 2.2 Ilustrasi alur proses algoritma Asimetris

2.2 Kriptografi Klasik

Kriptografi klasik merupakan ilmu yang digunakan pada zaman dulu dalam melakukan pengamanan pesan rahasia, pada umumnya kriptografi klasik menggunakan teknik penyandian yang menggunakan algoritma simetrik dalam melakukan Enkripsi dan Dekripsinya. Kriptografi klasik biasanya menggunakan metode matematika dalam melakukan proses penyandian, yang dimana metode – metode yang digunakan antara lain yaitu bilangan integer, aritmetika modular dan matriks

2.2.1 Aritmetika Integer

Aritmetika Interger merupakan sebuah operasi aritmetika yang terdiri dari " Penjumlahan, Perkalian, Pengurangan, dan Pembagian ". himpunan bilangan suatu integer biasanya di simbolkan $\mathbb{Z}$ yang berarti merupakan himpunan bilangan bulat (tanpa bilangan pecahan) yang memiliki nilai $-\infty$ sampai ∞ yang dijabarkan dengan persamaan seperti ini.

$$\{-\infty,\ldots,-5,-4,-3,-2,-1,0,1,2,3,4,5,\ldots,\infty\} \quad (2.1)$$

a. Operasi pada himpunan bilangan integer

Bilangan integer terdapat 3 operasi yaitu penjumlahan (a + b), pengurangan (a – b), dan perkalian (a x b) memerlukan 2 bilangan masukan yang nanti hasilnya menjadi sebuah bilangan integer lain. Operasi pembagian ( ) integer di interpresentasikan memiliki 2 keluaran yaitu hasi bagi q (quotient) dan sisa bagi r (remainder).

Sebagai contoh a = 70 (nilai yang dibagi) dan n = 17 (nilai pembag) maka ditemukan hasil bagi q = 4 (hasil Bagi) lalu sisa bagi didapatkan r = 2 (sisa bagi) menghasilkan persamaan 70 = 17 x 4 + 2 yang dijabarkan dalam rumus persamaan seperti ini.

$$a = n \times q + r \quad (2.2)$$

b. Faktor persekutuan terbesar (*Greated commong divisor*)

Faktor Persekutuan terbesar atau bisa dibilang *Greates Common Divisor* (GCD) adalah elemen terbesar pada himpunan divisor dua bilangan integer. Divisor merupakan bilangan integer yang membagi habis sebuah bilangan integer. Seorang matematikawan yang bernama aleksander pada zaman Ptolemy I membuat algoritma menemukan gcd dua buah bilangan integer α dan b dengan cara rekursif.

$$(2.3)$$

Pada persamaan rumus tersebut menyatakan bahwa mencari nilai gcd(a,b) dapat direduksi menjadi gcd(b, a mod b). contoh mencari nilai divisor 24 memiliki divisor {1,2,3,4,6,8,12,24} dan 32 memiliki divisor {1,2,4,8,16}, maka himpunan divisor {1,2,4,8} dan yang terbesar adalah 8 yang dijabarkan sebagai gcd (24,32) = 8. (Sadikin 2012).

2.2.2 Aritmetika Modular

a. Aritmetika modular m

Aritmetika modular m merupakan aritmetika yang digunakan untuk menghasilkan nilai integer pada ruang lingkup sama. Yang berarti nilai hasil tidak akan melebih jumlah dari

modulus, yang dimana alphabet “A” sampai “Z” di ubah menjadi urutan “0” sampai “25” maka nilai modulus yang harus diberikan berjumlah nilai 26 sehingga ruang lingkup tidak melebihi urutan (Sadikin 2012). Maka bisa dijabarkan seperti ini.

Contoh (14 + 13) mod 26, jika tanpa modulus maka hasil dari (14 + 13) = 27 yang berarti melebih urutan yang sudah ditentukan, sedangkan dengan menggunakan modulus maka (14 + 13) mod 26 = (27 mod 26) yang menghasilkan nilai 1 yang artinya nilai yang didapat tidak melebih jumlah urutan atau bisa dibilang modulus merupakan pembatas dalam ilmu kriptografi.

b. Invers Modular

Invers modular merupakan sebuah pencarian identitias 2 buah bilangan yang dimana kedua bilangan tersebut memiliki hubungan terhadap nilai modulus (Sadikin 2012). Yang artinya apabila 2 bilangan a dan b pada $\mathbb{Z}_{26} = 1$ maka nilai b didapat sebagai sebuah invers terhadap nilai a, dengan rumus a x b = 1 (mod m). Sebagai contoh nilai a = 5 dan nilai b = 21, maka dengan menggunakan rumus tersebut 5 x 21 = 105. 105 lalu di modulus dengan nilai 26 maka menghasilkan nilai 1, yang artinya bahwa nilai a dan b memiliki hubungan

2.2.3 Matriks berelement $\mathbb{Z}m$

Kriptografi Klasik memiliki beberapa metode yang menggunakan matriks dalam algoritmanya yang dimana tiap elemen memiliki beberapa indeks kolom dan indeks baris yang digunakan sebagai kunci dan *plaintext*. Matriks merupakan sebuah kumpulan nilai dalam sebuah table dengan ukuran m x n yang artinya jumlah baris dan kolom memiliki jumlah yang di tentukan (Sadikin, 2012).

$$A = \begin{bmatrix} \square\square\square & \cdots & \square\square\square \\ \vdots & \ddots & \vdots \\ \square\square\square & \cdots & \square\square\square \end{bmatrix} \text{atau } [\alpha]_{mn} \qquad (2.4)$$

Matriks A yang terbentuk dari m x n merupakan matriks bujur sangkar, perkalian matriks memiliki sifat asosiatif yang artinya matriks dengan identitas m x n jika m = n = 1 pada diagonal pertama dan elemen lainnya bernilai 0 maka matriks tersebut dinamai matriks identitas (Ariyus 2006).

$$\square = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (2.5)$$

2.3 Caesar Cipher

Caesar Cipher merupakan kriptografik klasik yang menerapkan aritmetika modular. Teknik *Caesar Cipher* ini menggunakan subtitusi yang menggeser sebuah karakter lain menjadi karakter lain yang dimana karakter tersebut diurutkan terlebih dahulu. *Caesar Cipher* merupakan subtitusi cipher yang pertama dalam dunia penyandian pada waktu pemerintahaan yulius Caesar yang dikenal dengan *Caesar Cipher*, dengan mengganti posisi huruf awal alphabet (Ariyus 2006). Secara umum, metode *Caesar Cipher* mengunakan operasi shift, Operasi Shift adalah mensubtitusikan suatu huruf menjadi huruf pada daftar karakter berada di-k sebelah kanan dan sebelah kiri huruf itu (Sadikin 2012). Yang dapat dicontohkan seperti tabel dibawah ini :

Table 2.1 Subtitusi awal

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Setelah menggunakan metode Caesar Cipher maka menjadi seperti ini :

Table 2.2 Perubahaan subtitusi

D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0	1	2

Dalam proses penyandian *Caesar Cipher* ini, harus mengetahui jumlah pergeseran/operasi shift nilai dari kunci yang ditetapkan sebelum melakukan komunikasi antara kedua bela pihak si penerima dan si pengirim. Misalkan dalam Kunci (K) yang digunakan K = 3, maka jumlah pergeseran setiap alphabet akan bergeser 3 langkah yang dimana alphabet ‘A’ bergeser menjadi ‘D’, alphabet ‘B’ bergeser menjadi ‘E’ dan seterus sehingga semua alphabet yang bertotal 26 huruf ini berpindah tempat. Untuk huruf akhir yang perpindahannya melebihi jumlah total dalam kolom alphabet maka bergeser ke kolom awal. Seperti alphabet ‘X’ bergeser menjadi ‘A’, alphabet ‘Y’ bergeser menjadi ‘B’, dan alphabet ‘Z’ bergeser menjadi ‘C’ untuk melihat lebih jelas maka ada di Table 2.1 dan Table 2.2 setelah perpindahan menggunakan Kunci(K) = 3. Rumus yang digunakan dalam metode penyandian Caesar Cipher yaitu :

$$C = (P + K) \bmod 26 \qquad (2.6)$$

$$P = (C – K) \bmod 26 \qquad (2.7)$$

Keterangan :

C = Ciphertext

P = Plaintext

K = Kunci/shift

Secara umum maka konsep dalam Caesar cipher dirangkum dengan menggunakan langkah – langkah sebagai berikut:

a. Menentukan jumlah pergeseran atau kunci K yang digunakan untuk mengubah sebuah *plaintext* ke *ciphertext* maupun dari *ciphertext* ke *plaintext* yang dimana kunci memiliki rentang { 1,…,26 } tergantung jumlah tabel karakter. Pada penelitian ini menggunakan 95 karakter yang memiliki rentang { 1, …, 95 }.
b. Lalu menukar sebuah karakter *plaintext* atau chiperteks dengan menggunakan jumlah pergeseran kunci yang sudah ditetapkan.

Keamanan metode *Caesar Cipher* tidak terlalu kuat, hal ini dapat didemonstrasi dengan cara tipe penyerangan *Ciphertext-only attack* yaitu dengan hanya melihat teks sandi saja nilai kunci K pada Sandi Caesar dapat dipecahkan yang disebabkan oleh besar ruang nilai yang mungkin bagi kunci K terlalu kecil yaitu 26 (Sadikin 2012).

Meskipun Caesar cipher lemah akan tetapi para kriptanalisi harus mengetahui *ciphertext* terlebih dahulu agar bisa memecahkan enkripsi karena harus mengetahui jumlah pergeseran setiap karakter, yang artinya menurut (Kurniawan 2004) mengatakan Teknik *Ciphertext-only Attack* yaitu Cryptanalyst hanya memiliki beberapa pesan *chipertext*, semuanya dienkripsi dengan algoritma yang sama.

2.4 Affine Cipher

Affine Cipher merupakan metode kriptografi klasik yang sudah di perluas dari metode *caesar cipher*. *Affine Cipher* merupakan sandi monoalfabetik yang menggunakan teknik subtitusi dan fungsi liner a.p + b untuk enkripsi teks asli dan a^{-1}.(c – b) untuk dekripsi teks sandi pada $\mathbb{Z}_{26}$ (Sadikin 2012).

Untuk memperjelas teori *Affine Cipher*, maka rumus *Affine Cipher* dalam Enkripsi dan Dekripsi di asumsikan menggunakan 26 karakter. Contoh 26 karakter seperti pada Table 2.1. Maka rumus yang digunakan seperti ini

$$E_k(x) = C = a.P + b \bmod 26 \quad (2.8)$$

$$E_d(x) = P = a^{-1}(C - b) \bmod 26 \quad (2.9)$$

Perbedaan *Caesar cipher* dan *affine cipher* yaitu pada metode ini ada penambahan nilai kunci dalam penyandiannya. Penyandian dalam metode *affine cipher* menggunakan 2 integer kunci dalam pengenkripsian dan pengdekripsian yaitu nilai a dan nilai b. nilai kunci "a" pada integer yang dipakai dalam penyandian $\mathbb{Z}_{26}$ harus memiliki invers untuk memenuhi kebutuhan dalam melakukan enkripsi dan dekripsi yang memenuhi Invers Modular, yang berarti nilai yang terkandung dalam kunci a dan a^{-1} harus memiliki nilai 1 pada nilai integer modulus 26. Jika nilai a tidak Invers Modular dengan mod 26 maka tidak dapat digunakan dalam penyandian. Sedangkan untuk nilai kunci "b" digunakan sebagai kunci pergseran.

Secara umum maka konsep penyandian Affine Cipher yang dirangkum dengan langkah – langkah sebagai berikut.

a. Menentukan nilai kunci "a" enkripsi dan dekripsi yang memiliki nilai Invers Modular dan menentukan jumlah kunci pergeseran yang di simbolkan dengan " b ".
b. Lalu menukar setiap karakter *plaintext* ke *ciphertext* dengan menggunakan kunci nilai a dan nilai b yang sudah ditentukan
c. Untuk melakukan dekripsi, maka mencari nilai invers dari kunci a yang memiliki invers modular. Maksud dari invers modular yaitu kunci "a" dengan kunci "a^{-1}" menghasilkan nilai 1 terhadap mod 26. Jika nilai kunci "a" tidak invers modular terhadap kunci invers "a^{-1}" pada mod 26, maka kunci tidak dapat digunakan sebagai kunci dekripsi
d. Jika nilai enkripsi "a" invers modular terhadap kunci invers invers "a^{-1}", maka kunci invers "a^{-1}" dijadikan sebagai kunci dekripsi dan nilai kunci "b" yang digunakan tetap sama sebagai kunci pergeseran, lalu proses dapat mengubah *ciphertext* menjadi *plaintext*

Sandi Affine memiliki kelemahan yang sama dengan sandi Caesar yaitu ukuran ruang kunci yang kecil. Ukuran ruang kunci sandi *affine cipher* adalah 12 x 26 = 312 kali percobaan kriptanalisis untuk dapat dipecahkan. 12 adalah jumlah bilangan integer yang memiliki invers di $\mathbb{Z}_{26}$ yang dapat dipakai sebagai nilai " a " sedangkanya untuk " b " terdapat 26 nilai kemungkinan. Dengan ruang kunci yang kecil menyebabkan sistem sandi *affine cipher* dapat dipecahkan dengan pencarian *brute force* (Sadikin 2012).

2.5 Hill Cipher

Hill Cipher merupakan kriptografi klasik yang menerapkan ilmu matematika yaitu aritmetika modular dan matriks dalam penyandiannya. *Hill Cipher* ditemukan/diciptakan oleh Lester S. Hill pada tahun 1929. Ide hill cipher adalah mengambil m kombinasi linier dari m karakter alphabet dalam satu element *plaintext*, sehingga menghasilkan m alphabet karakter dalam satu element (Ariyus 2006).

Secara penjelasan maka Hill cipher menggunakan algoritma K dengan matriks *m x m* yang dimana digunakan untuk mensubtitusi *n* alphabet. Dengan matriks K *m x m* mengindentifikasi suatu blok matriks yang dimaksud bahwa Matriks K harus memiliki invers dan determinan Matriks K harus memiliki invers juga terhadap $\mathbb{Z}_{26}$ yang berarti Matriks K merupakan Matriks *invertible* yaitu *multiplicative inverse* seperti ini (Sadikin 2012).

$$K^{-1} == K.K^{-1} = I \qquad (2.10)$$

Kreteria dalam matriks K yang mempunyai invers atau tidak mempunya invers yaitu jika suatu determinan dari Matriks K membentuk suatu bujur sangkar yang akan memenuhi invers dan determinan (det(K)) $\neq$ 0 tidak bernilai 0 maka matriks K mempunyai Invers yang dilambang K^{-1}, sedangkan jika determinan (det(K)) = 0 maka matriks K tidak mempunyai invers. Maka dari itu diuraikan dengan rumus seperti ini dari persamaan Matriks K^{-1} dengan ordo 2 x 2

$$K^{-1} = \frac{1}{\square\square - \square\square} \begin{bmatrix} \square & -\square \\ -\square & \square \end{bmatrix} \neq 0 \qquad (2.11)$$

Rumus yang digunakan dalam Kriptografi Klasik pada metode Hill Cipher yaitu:

$$(\square 1 \quad \dots \quad \square\square) = (\square 1 \quad \dots \quad \square\square) \begin{bmatrix} \square 11 & \cdots & \square 1\square \\ \vdots & \ddots & \vdots \\ \square\square 1 & \cdots & \square\square\square \end{bmatrix} \qquad (2.12)$$

Proses Enkripsi pada Hill Cipher akan di jabarkan pada langkah – langkah sebagai berikut.

a. Menentukan jumlah Matriks kunci K dengan ukuran *m x m*. Matriks yang digunakan harus memiliki invers

b. Membagi setiap *plaintext* ke dalam sebuah blok – blok matriks yang dimana setiap blok memiliki *n* karakter dan nilai *n* sama dengan ukuran matriks Kunci. Jika menggunakan Kunci Matriks 2 x 2 maka setiap karakter akan di ubah menjadi 2 blok perkarakter.
c. Jika total karakter melebihi atau tidak memenuhi jumlah *n* pada karakter blok, maka ada penambahan karakter " X " pada sejumlah karakter yang kekurangan pada blok terakhir.
d. Mengalikan Karakter Kunci dengan *plaintext* setiap blok masing – masing.

Untuk melakukan Dekripsi langkah – langkahnya sama seperti enkripsi tetapi Matriks Kunci yang digunakan yaitu Matriks Invers dari kunci awal.

Hill Cipher merupakan kriptografi klasik yang termasuk sulit depecahkan oleh kriptanalisis untuk mengetahui sebuah pesan yang terkandung didalamnya, akan tetapi tidak memungkinkan *Hill Cipher* tidak memiliki suatu celah. Teknik yang digunakan untuk memecahkan *Hill Cipher* yaitu dengan analisis sandi yang dinamakan *Known Plaintext attack*. Apabila penyerang mampu mengumpulan pasangan teks asli dan teks sandi dengan menggunakan kunci yang sama, penyerang dapat menemukan kunci sandi hill cipher (Sadikin 2012).

Maka Oleh sebab itu, untuk mengurangi kelemahan Algoritma *Hill Cipher* pada penelitian ini. Algoritma *Hill Cipher* diproses setelah Algoritma *Affine Cipher*, sehingga menutupi kelemahan dari Teknik *Known Plaintext Attack*.

2.6 Kriptografi Modern

Kriptografi Modern berbeda dengan enkripsi Konversional, dikarenakan pada Enkripsi modern berbeda dengan enkripsi kriptografi klasik. Pada Kriptografi Klasik, menggunakan sistem subtitusi dan permutasi karakter dari *plaintext* dan menggunakan rumus matematika, sedangkan Kriptografi Modern Karakter yang ada di konversi ke dalam suatu urutan digit biner (bits) yaitu 0 dan 1, yang umum digunakan untuk Schema Encoding ASCII (*American Standard Code for Information Interchange*) (Ariyus 2006). Dengan perbedaan itulah Kriptografi Modern hanya bisa dilakukan pada komputer dalam pengoprasian enkripsi dan dekripsinya .

Sejak kriptografi modern menggunakan urutan digit biner (bits), maka penggunaannya dibiasakan menggunakan metode kombinasi yang disebut *Exculisve OR* atau bisa ditulis XOR (⊕). Fungsi dalam Exculisve OR atau XOR untuk bertujuan menghasilkan logika nilai TRUE (1) dan logika nilai FALSE (0), yang dimana jika sebuah logika TRUE di XOR-kan dengan

FALSE maka hasilnya TRUE selain itu maka FALSE. seperti yang digambarkan sebagai berikut.

Table 2.3 Fungsi XOR digit biner

A	B	A⊕B
1	1	0
1	0	1
0	1	1
0	0	0

Sedangkan dalam Kriptografi Modern menggunakan cara umum dalam menulis suatu bit-string dengan menggunakan *Hexadecimal*. Untuk *Hexadecimal* dibagi beberapa kelompok yang berukuran 4 bit. Seperti yang digambarkan sebagai berikut.

Table 2.4 Kelompok *hexadecimal*

0000 = 0	0001 = 1	0010 = 2	0011 = 3
0100 = 4	0101 = 5	0110 = 6	0111 = 7
1000 = 8	1001 = 9	1010 = A	1011 = B
1100 = C	1101 = D	1110 = E	1111 = F

2.7 AES (Advanced Encrypt Standard)

Advanced Encryption Standard (AES) merupakan salah satu kriptografi modern yang menggunakan bit – bit *schema encoding ASCII* dalam proses penyandiannya. Kriptografi ini memiliki cerita dalam pembuatannya sebagai kriptografi yang dianggap kuat dan efesien yang dimana dengan langkah - angkah yang sudah di tentukan. Untuk mengetahui proses cerita AES ini berdiri dan proses langkah – langkahnya, maka akan diperjelas sebagai berikut

2.7.1 Deskripsi AES

Advanced Encryption Standard (AES) dipublikasikan Oleh NIST (National Institute of Standard) pada Tahun 2001. AES merupakan algoritmat simetris block cipher yang menggantikan DES (Data Encryption Standard). Karena algoritma DES (Data Encryption Standar) sudah berhasil dipecahkan pada tahun 1998 dalam 96 Hari, dengan alasan tersebut NIST mengadakan kompetisi untuk menggantikan algoritma DES dengan kriteria yang ditetapkan yaitu dari segi tingkat keamanan, harga, algoritma dan karakteristik implementasi.

Pada tahun agustus 1995 dipilih 5 kandidat untuk seleksi akhir yaitu *Rijendel* termasuk dalam 5 kandidat tersebut dan akhirnya pada 2 oktober 2000 pemenang dalam kompetisi yang terpilih yaitu algoritma Rijndael sebagai pemenang, yang diciptakan oleh Dr. Vincent Rijem dan Dr.Joan Daemen. Karena algoritma *rijendel* algoritma paling aman dari 5 kandidat dalam segi keseimbangan antara keamanan dan fleksibilitas (Ariyus 2006).

Kunci *advanced Encryot Standard* AES memiliki panjang bit yaitu 128, 192, dan 256 bit, jumlah ronde AES tergantung panjang kunci bit yang digunakan . Setiap kunci yang digunakan untuk memasukan Ronde berikutnya yang dimana berdasarkan kunci yang diberikan. Maka digambar sebagai berikut hubungan panjang kunci AES dengan Jumlah Ronde.

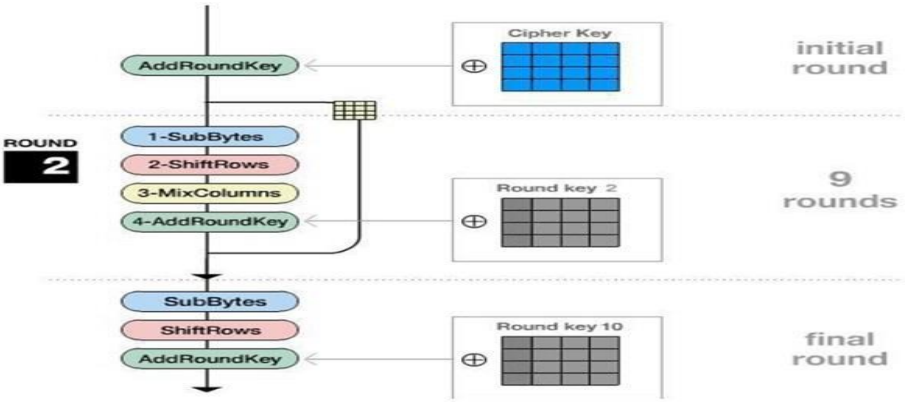
Table 2.5 Hubungan panjang kunci dan jumlah ronde

Panjang Kunci AES dalam Bit	**Jumlah Ronde/putaran (Nr)**
128	10
192	12
256	14

Dalam penjelasan dari pandangan (Sadikin 2012) mengatakan AES memiliki 5 unit ukuran data : bit, Byte, word, blok, dan state. Bit merupakan satuan data terkecil, yaitu nilai digit sistem biner. Sedangkan 1 Byte berukuran 8 Bit, 1 word berukuran 4 Byte (32 bit). 1 Block berukuran 16 byte (128 Bit) lalu *state* merupakan block sebuah matriks yang berukuran 4x4. Sedangkan untuk ukuran kunci dari Panjang Kunci AES berbeda – beda, dimana untuk panjang kunci AES 128 bit memiliki 4 word (16 Byte), panjang kunci AES 192 memiliki 6 word (24 Byte), dan Panjang Kunci AES 256 memiliki 8 word (32 Byte).

2.7.2 Algoritma AES (Rijndael)

Dalam proses penggunaan algoritma AES harus mengetahui beberapa langkahnya seperti yang digambarkan dibawah ini.



Gambar 2.3 Algoritma AES

Seperti pada Gambar 2.3, algoritma AES terdiri dari 3 langkah yaitu, *initial round* (putaran awal), *round* (putaran 1 – 9), dan *final round* (putaran akhir). Dekripsi pada AES sama saja seperti Enkripsi cuman posisi kebalikan dari Enkripsi AES. Setiap Ronde AES memiliki 4 jenis transformasi yaitu SubByte, ShiftRows, MixColumns dan AddRoundKey. Sebelum melakukan 4 jenis transformasi sebuah *plaintext* dan kunci di kelompokan terlebih dahulu ke dalam sebuah *state*.

Seperti penjelasan pada deskripsi AES, *state* merupakan suatu blok matriks 4 x 4 yang artinya setiap *plaintext* maupun kunci enkripsi dan dekripsi AES dikelompokan dalam matriks. Seperti ini

$$\text{State } 4x4 = \begin{bmatrix} \square 0 & \square 4 & \square 8 & \square 12 \\ \square 1 & \square 5 & \square 9 & \square 13 \\ \square 2 & \square 6 & \square 10 & \square 14 \\ \square 3 & \square 7 & \square 11 & \square 15 \end{bmatrix} \quad (2.13)$$

Seperti pada persamaan 2.13, satu baris b_n merupakan sebuah Byte yang dimana memiliki nilai 8 bit sehingga jika dikalikan dengan total 16 byte maka menghasilkan 128 bit (16 byte x 8 bit). Setiap state menggunakan bilangan heksadesimal dan akan diproses menjadi keluaran yang dimana keluarannya juga menghasilkan 128 bit. Setelah mengubah sebuah *plaintext* menjadi state maka proses selanjutnya menggunakan 4 jenis transformasi.

a. SubByte

SubByte merupakan subtitusi nonlinier yang dimana dalam transformasi SubByte menggunakan 2 tabel subtitusi yaitu tabel SubBytes (S-Box) dan InvSubByte (Inv – Box), fungsi ini digunakan untuk menunjukan indeks kolom dari tabel subtitusi. Contoh tabel SubByte seperti ini.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Gambar 2.4 Tabel transformasi SubByte (S-Box)

Jika dilihat pada Gambar 2.4 terdapat sebuah baris dan kolom berbentuk tabel yang menunjukan sebuah kelompok bilangan *hexadecimal*, untuk melihat bentuk kelompok *hexadecimal* pada Table 2.4. Setiap sebuah *state* yang bernilai Bytes disubtitusikan menggunakan tabel tersebut, sehingga mendapatkan sebuah bilangan *hexadecimal* lagi. Tabel transformasi SubBytes digunakan untuk melakukan enkripsi, contoh dalam penggunaan tabel SubByte (S-Box), jika dalam state memiliki nilai " D3 " dengan tabel subtitusi maka menghasilkan " 66 ". Untuk melakukan Dekripsi maka menggunakan Tabel InvSubByte (Inv-Box). Seperti yang digambarkan dibawah ini

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	D0	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

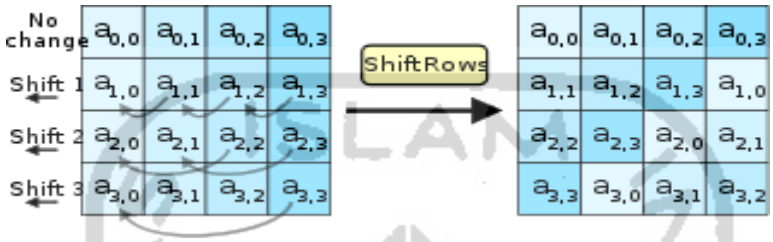
Gambar 2.5 Tabel transformasi InvSubByte (Inv-Box)

Dengan menggunakan Gambar 2.5 ini sebuah *state* yang sudah di enkrip dapat didekripsi. Sebagai contoh menggunakan sebuah state dengan nilai " 66 ". nilai ini diambil dari hasil

menggunakan Tabel SubByte maka jika menggunakan Tabel InvSubByte menghasilkan nilai " D3 ".

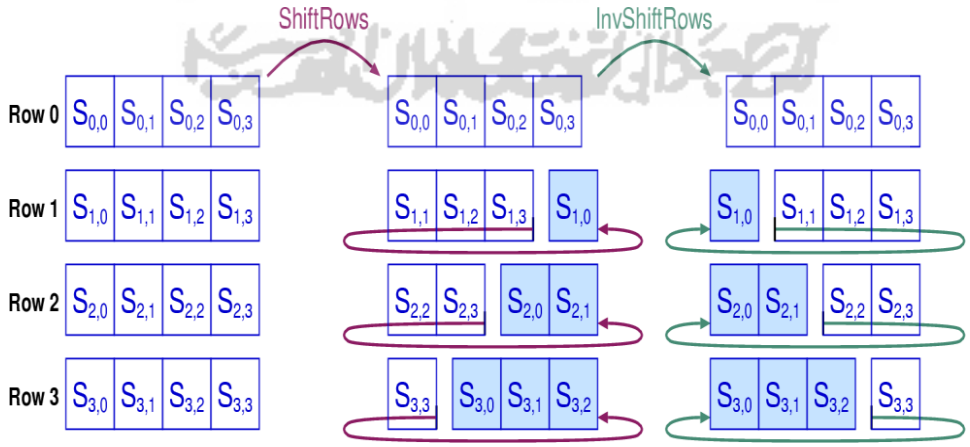
b. ShiftRows

SiftRows merupakan jenis transformasi permutasi yang berguna untuk mengganti sebuah element pada sebuah *state*. Permutasi ini menjalankan sebuah operasi *circular shift left* sebanyak baris pada sebuah state. Maka dijelaskan pada gambar seperti ini.



Gambar 2.6 Permutasi *ShiftRows*

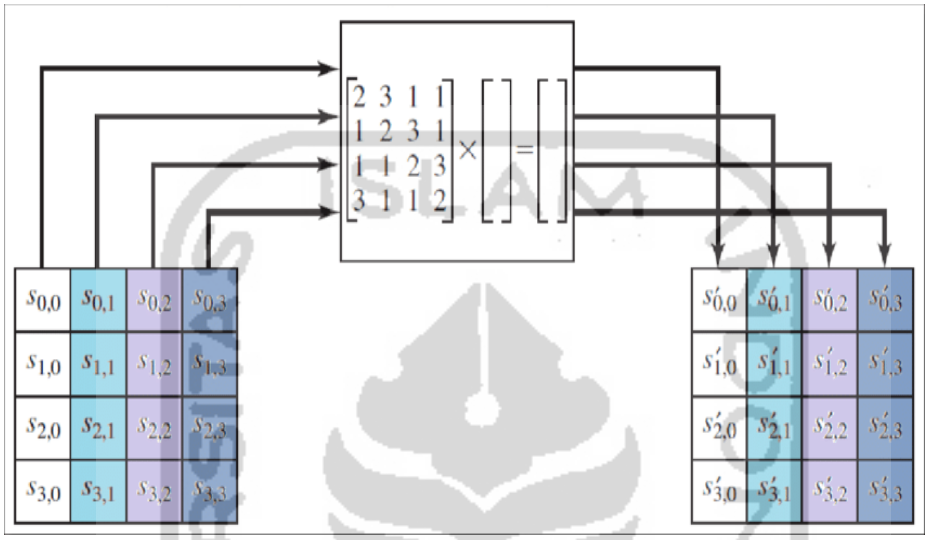
Dalam Gambar 2.6 menjelaskan sebuah *state* dalam proses ShiftRow yang dimana baris pertama tidak mengalami perubahan, sedangkan setelah baris pertama mengalami perubahan yaitu baris ke 2 mengalami pergeseran 1 kali kekiri, baris ke 3 mengalami pergeseran 2 kali kekiri dan untuk baris ke 4 mengalami perubahan sebanyak 3 kali kiri. Proses ini dipergunakan untuk melakukan enkripsi sedangkan untuk melakukan dekripsi yaitu dengan cara melakukan pergeseran yang sama tetapi pergeseran dilakukan ke sebelah kanan. Seperti digambar dibawah ini



Gambar 2.7 Permutasi *InvShiftRows*

c. MixColumns

Transformasi MixColumns menggunakan perkalian matriks dengan mencapur sebuah kolom kolom state terhadap sebuah baris matriks konstan. Transformasi MixColumn yang menjadikan biner element state sebagai polynomial. Lalu sebuah baris ke-i pada konstan dan kolom ke-j pada state. Seperti ilustrasi di bawah ini.



Gambar 2.8 Transformasi MixColumns

Pada Gambar 2.8 memperlihatkan sebuah kolom *state* melakukan perkalian pada matriks konstan fungsi ini untuk mengenkripsi. Untuk melakukan proses perkaliannya maka akan di jabarkan dibawah ini.

$$S'_{0,j} = (S'_{0,j}*2) \text{ XOR } (\oplus) (S'_{0,j}*3) \text{ XOR } (\oplus) (S'_{0,j}*1) \text{ XOR } (\oplus) (S'_{0,j}*1)$$

$$S'_{0,j} = (S'_{0,j}*1) \text{ XOR } (\oplus) (S'_{0,j}*2) \text{ XOR } (\oplus) (S'_{0,j}*3) \text{ XOR } (\oplus) (S'_{0,j}*1)$$

$$S'_{0,j} = (S'_{0,j}*1) \text{ XOR } (\oplus) (S'_{0,j}*1) \text{ XOR } (\oplus) (S'_{0,j}*2) \text{ XOR } (\oplus) (S'_{0,j}*3)$$

$$S'_{0,j} = (S'_{0,j}*3) \text{ XOR } (\oplus) (S'_{0,j}*1) \text{ XOR } (\oplus) (S'_{0,j}*1) \text{ XOR } (\oplus) (S'_{0,j}*2)$$

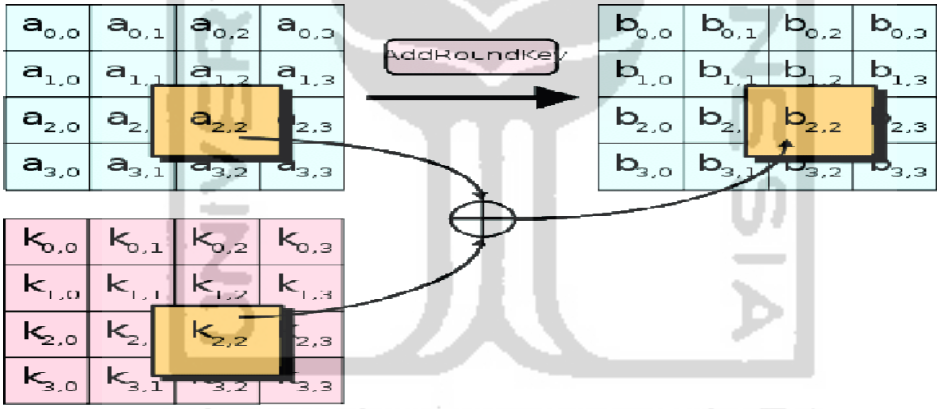
Sedangkan untuk melakukan dekripsi, proses yang dilakukan sama seperti enkripsi yaitu mengkalikan sebuah state dengan matriks konstan. Tetapi matriks yang digunakan pada dekripsi berbeda dengan enkripsi, Seperti ini pada Gambar 2.9.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Gambar 2.9 Transformasi InvMixColums

d. AddRoundKey

Transformasi AddRoundKey merupakan transformasi yang memilik sifat *self invers* yang artinya transformasi invers sama dengan transformasi aslinya dan Transformasi AddRoundKey digunakan untuk mencapurkan sebuah state dengan kunci menggunakan *Exclusive OR* atau biasa disebut XOR (⊕) dengan posisi byte yang sama. Maka akan di jelaskan seperrti gambar berikut.



Gambar 2.10 Transformasi AddRoundKey

Posisi $a_{2,2}$ sama dengan posisi $k_{2,2}$, dengan posisi yang sama maka transformasi AddRoundKey beroperasi menggunakan XOR (⊕). Jika kita misalkan $a_{2,2}$ = 76 (bentuk dalam Heksadesimal) di XOR-kan (⊕) dengan $k_{2,2}$= 12 (bentuk dalam Heksadesimal) maka menghasilkan sebuah $b_{2,2}$ = 64.

2.7.3 Ekspansi Kunci

Pada Algoritma *Advanced Encrypt Standar* (AES) menggunakan fungsi ekspansi kunci untuk memperluas kunci AES, karena kunci – kunci tersebut digunakan untuk setiap *round*(putaran) pada algoritma AES yang dimana memiliki sebuah prosedur untuk melakukan

ekspansi yaitu dengan memasukan 4 *word* kunci eksternal. Ketika word eksternal memasuki rounds(putaran) maka kunci tersebut di ekpansi untuk melanjutkan rounds berikutnya. Setiap putaran dari 4 *word* kunci akan di proses sampai akhir, sehingga menghasilkan 44 *word*. 44 *word* ini didapat dari kunci eksternal (kunci luar) dan Ekspansi kunci (10 rounds) sehingga mendapatkan 176 byte (4 *word* * 44 *word*).

Langkah – langkah ekpansi kunci AES ada 3 langkah berikut penjelasannya yaitu.

a. Rot Word (4 Bytes)

Proses ini seperti proses shift rows pada sebuah state yang dimana menggeser pada baris – barisnya seperti Gambar 2.6. Contoh 4 byte pada word bergeser, yang dimana 4 byte (k_0, k_1, k_2, k_3) setelah melakukan rows word menjadi seperti ini (k_1, k_2, k_3, k_0).

b. Sub Word

Pada Langkah ini sama seperti transformasi SubByte menggunakan sebuah tabel S-box seperti pada Gambar 2.4.

c. RCon

Pada langkah ini menggunakan sebuah tabel dalam bentuk heksadesimal seperti ini.

Table 2.6 Rcon *hexadecimal*

I	Rc[i]
1	01000000
2	02000000
3	04000000
4	08000000
5	10000000
6	20000000
7	40000000
8	80000000
9	1B000000
10	36000000

Fungsi Rcon ini akan bergerak tergantung dari penggunaan kunci yang dimana jika menggunakan AES-128 Rcon ini akan bergerak pada Round (Putaran) ke-4 dengan perhitungan seperti ini (4 /(16 / 4)) – 1 = 0 jika menggunakan metode AES lain seperti AES-192 dan AES-255 maka fungsi ini bergerak pada Round (Putaran) yang berbeda.

2.7.4 Keamanan Advanced Encrypt Standard (AES)

Sandi AES sampai saat ini masih dianggap aman untuk digunakan, keamanan kunci AES disebabkan oleh jumlah bit yang digunakan dibandingkan DES, karena kunci AES memakai 128, 192, dan 255 bit sedangkan untuk DES (Data Encrypt Standard) menggunakan 56 bit.

Dalam pandangan (Kurniawan 2004) mencantumkan waktu rata – rata untuk memecahkan kunci dengan teknik *brute force-attack*. Artinya hanya setengah dari seluruh kemungkinan kunci yang dicoba dan dianggap akan mendapatkan satu buah kunci. Yang di hitungkan sepert ini

Table 2.7 *Brute force-attack* memecahkan kunci

Ukuran Kunci	Jumlah Kemungkinan Kunci	1 Kunci /ms	10^6 Kunci/ms
40 bit	$2^{40} = 1{,}1 \times 10^{12}$	2^{39}=152,7 jam	55 ms
56 bitx`	$2^{56} = 7.2 \times 10^{20}$	2^{55} ms = 1142 tahun	10.01 jam
128 bit	$2^{128} = 3.4 \times 10^{38}$	2^{127}ms = 5,4 x 10^{24} tahun	5.4x 10^{18} tahun

2.8 Fungsi Hash Satu Arah

Fungsi *hash* adalah salah satu algoritma kriptografi modern yang banyak dimanfaatkan untuk aplikasi pengamanan data, seperti integritas pesan dan otentikasi (Maryanto 2008). Metode ini biasa digunakan untuk melakukan pengujian sebuah file yang dikirim ataupun yang didownload melalui media untuk melihat nilai ciri (*signature*)/identik dari sebuah file, agar mengetahui terjadinya sebuah perubahan dalam file.

Metode ini merupakan metode yang sangat dibutuhkan untuk menjaga data dari sebuah ancaman kerusakan, pengubahan, dan penambahan. Dikarenakan fungsi hash adalah suatu fungsi yang secara efisien mengubah string input dengan panjang berhingga menjadi string output yang panjangnya tetap yang disebut hash (Sulianto 2014). Yang artinya sedikit saja perubahan pada data file, maka hash dari suatu data file akan berubah, sehingga dapat mengetahui keaslian data file tersebut.

Fungsi hash memiliki perhitungan (rumus) dalam mendapatkan nilai identiknya. Berikut persamaan rumus hash dalam mendapatkan nilai identiknya.

$$h = H(\mathrm{M}) \tag{2.14}$$

Berikut penjelasan fungsi – fungsi dari persamaan 2.14. *h* merupakan hasil dari proses *H* yang memiliki masukan M dengan nilai panjang tetap, *H* sendiri merupakan sebuah blok – blok

yang berukuran berapa saja, sedangkan untuk M merupakan masukan yang memasuki blok – blok dengan panjang/atau ukuran berapa saja. Dengan menggunakan fungsi *hash* ini, pesan yang sudah dirubah menjadi Message Digest tidak dapat dikembalikan lagi menjadi masukan pesan semula. Dikarenakan konsep *hash* satu arah ini berbeda dengan kriptografi lain yang tidak memiliki kunci private ataupun kunci public untuk melakukan dekripsinya.

2.9 Messagest Digest 5 (MD 5)

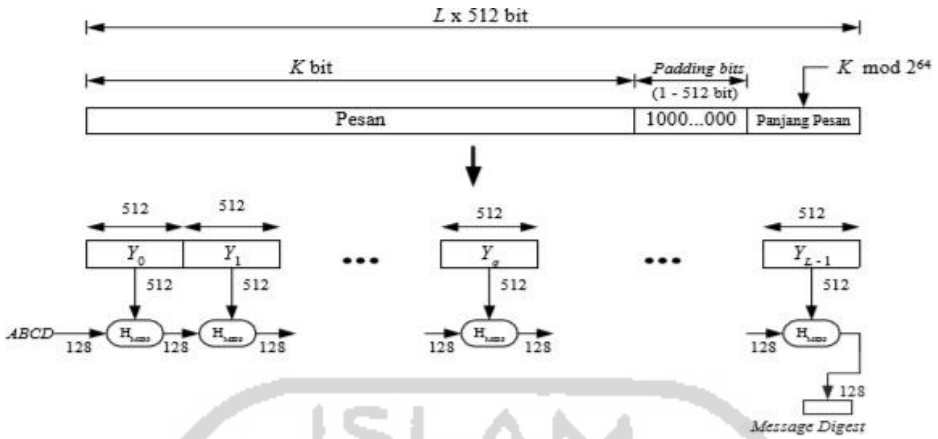
Messagest Digest 5 atau bisa disebut MD 5 ini merupakan salah satu algoritma yang terdapat di Fungsi Hash satu Arah dalam melakukan pengamanan data, yang beroprasi pada kondisi 128 bit dengan rangkaian 32 karakter digit *hexadecimal*. MD 5 merupakan algoritma yang paling banyak digunakan dalam keamanan jaringan komputer dan internet, yang dirancang oleh Ron Rivest yang merupakan salah satu pengembang algoritma RSA (Sukrisno and Utami 2007).

MD 5 merupakan pengembangan dari algoritma MD 4 yang memiliki perbedaan dalam jumlah putaran. Algoritma MD 4 memiliki 3 jumlah putaran, sedangkan untuk MD 5 memiliki 4 putaran, serta pada algoritma MD 5 memiliki penambahan nilai konstanta baru dan unik untuk diproses dengan *string* yang sudah ada. Dengan konsep pengembangan dari MD 4, algoritma MD 5 dapat bertahan dalam serang *cryptanalysis* dalam mengamankan data.

Dalam konsepnya, proses Message Digest memiliki 4 langkah secara garis besar untuk mendapatkan hash 32 karakter digit *hexadecimal* (Maryanto 2008), yang akan diuraikan sebagai berikut

a. Menambah bit – bit pengganjal

   Pesan ditambah dengan jumlah bit pengganjal sedemikian sehingga panjang pesan kongruen dengan 448 bit modul 512, yang artinya panjang pesan setelah ditambah bit – bit pengganjal 64 kurang dari kelipatan 512. Angka 512 ini digunakan, karena dalam proses MD5 pada setiap blok – blok pesan yang digunakan berukuran 512 bit. Meskipun pesan dengan panjang 448 bit, tetap ditambah dengan bit – bit pengganjal. Jika panjang pesan 448 bit, maka pesan ditambah dengan 512 bit menjadi 960 bit seperto pada Gambar 2.11. jadi konsepnya panjang bit – bit pengganjal antara 1 sampai 512 yang terdiri atas sebuah bit 1 yang diikuti dengan sisa bit 0.



Gambar 2.11 Proses blok – blok MD 5

b. Penambahan panjang pesan

Mengisi sisa ruang 64 bit tersebut dengan informasi panjang pesan semula. Jika panjang pesan 2^{64} maka yang diambil adalah panjang dalam modulo 2^{64}. Dengan kata lain, jika panjang pesan semula adalah *K* bit, maka 64 bit yang ditambahkan menyatakan *K* modulo 2^{64} Seperti pada Gambar 2.11. Setalah ditambah dengan 64 bit. panjang pesan yang sekarang menjadi kelipatan 512 bit.

c. Inisialisasi penyangga MD 5

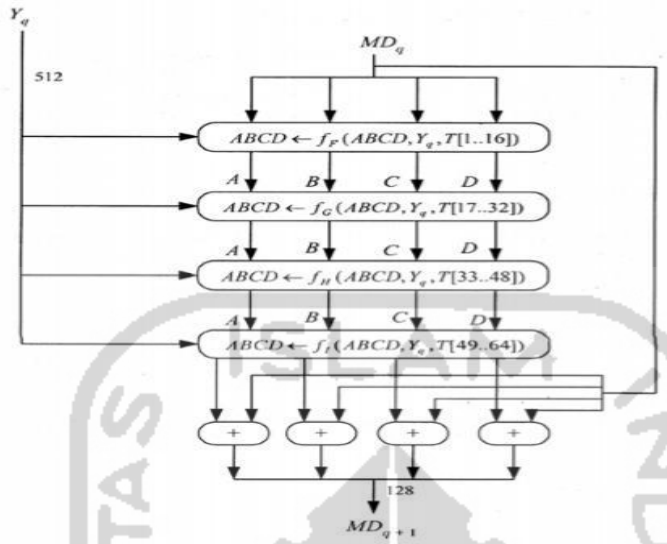
Dalam proses memalukan *hashing* pada algoritma MD5, membutuhkan sebuah penyangga yang dimana memiliki 4 penyangga yang masing – masing panjangnya 32 bit. total panjang penyangga yang digunakan pada algoritma MD5 ini 4 x 32 = 128 bit. Penyangga – penyangga ini, biasanya diberikan inisialisasi dengan nama A, B, C, dan D. setiap penyangga memiliki nilai – nilai yang berbeda dengan bentuk notasi *hexadecimal*. Berikut nilai inisialisasi dalam penyangga.

A = 01 23 45 67
B = 89 AB CD EF
C = FE DC BA 98
D = 76 54 32 10

d. Pengolahan pesan dalam blok 512 bit

Proses pengolahan pesan menjadi hashing ini, pesan dibagi menjadi sebuah blok yang masing – masing memiliki panjang 512 bit. setiap blok – blok pesan tersebut diproses bersamaan dengan penyangga yang sudah di inisialisasikan. Setiap blok – blok yang diproses dengan penyangga akan diolah sampai sebanyak 16 kali terhadap masukan, setiap

kali proses dasar menggunakan element T yang hasilnya menjadi keluaran 128 bit dan ini disebut Proses MD5. Seperti pada gambar berikut



Gambar 2.12 Alur MD5 mendapatkan Hash

2.10 Citra Digital

2.10.1 Pengertian Citra Digital

Citra adalah sebuah gambar yang ditinjau dari sudut pandang sistematis pada bidang dwimatra, sumber cahya menerangkan objek dan mementulkan kembali sebagian berkas cahya yang ditangkap oleh sebuah optik. Optik mengeluarkan sebuah gambar/foto yang terekam dari sebuah objek (Munir 2004).

Citra merupakan sebuah representasi/ gambar dengan kemiripan dari suatu objek, citra merupakan sebuah bentuk perekaman data berupa objek yang didapat melalui optif yang menghasilkan foto (Totok Sutoyo, Edy Mulyanto, Vincent Suhartono, Oky Dwi Nurhayati 2009).

2.10.2 Jenis – Jenis Citra Digital

Citra memiliki beberapa jenis yang terkandung didalam sebuah citra dengan bentuk yang berbeda - beda. Menurut (Munir 2004) ada 3 jenis citra yaitu.

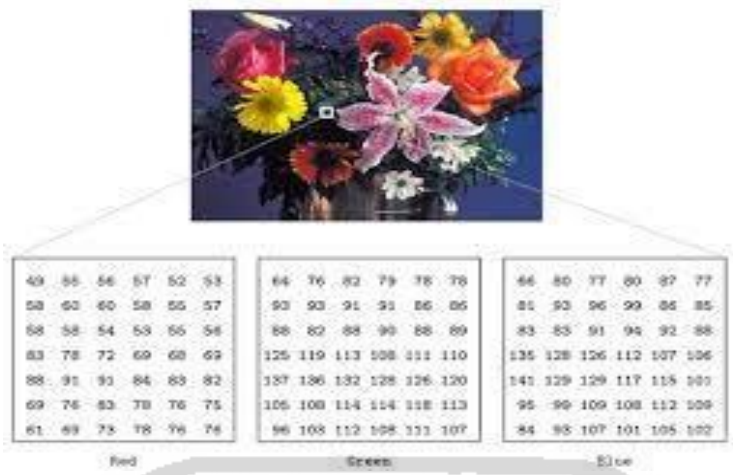
a. Citra Biner : Memiliki bentuk citra berwarna hitam atau putih. Citra biner memiliki nilai bit per-pixel 0 dan 1, yang dimana nilai 0 berwarna hitam dan nilai 1 memiliki warna putih. Citra biner ini masih dipergunakan hingga sekarang untuk dipergunakan sebagai tanda-tangan, sidik jari atau rancangan arsitektur. Ilustrasi gambar Citra biner.

Gambar 2.13 Bentuk citra biner

b. Citra Abu – abu : Citra abu – abu bisa disebut Citra warna 8-bit yang memiliki pewarnaan dengan nilai 0 sampai 256. Nilai 0 sebagai hitam sedangkan warna dari 0 - 256 menyatakan warna keabuan. Citra abu – abu masih digunakan untuk citra medis dan lain – lain. Ilustrasi gambar citra abu – abu sebagai berikut.

Gambar 2.14 Citra abu – abu (Citra 8 bit)

c. Citra RGB : Dalam Citra RGB memiliki 3 komponen warna dalam 1 pixel yang dimana nilai warna [255, 255, 255] yang disebut Citra warna 24-bit, dengan memiliki perwarnaan Red, Green, dan Blue dalam 1 warna memiliki 8 bit. 3 warna tersebut merupakan dasar dari sebuah warna yang jika dibungkan mendapatkan sebuah warna baru. RGB banyak digunakan disekarang ini untuk kepentingan Foto dan lain – lain yang ingin melihat sebuah gambar penuh warna. Ilustrasi gambar citra RGB sebagai berikut.



Gambar 2.15 Citra RGB (24-Bit)

2.11 Steganografi

2.11.1 Pengertian Steganografi

Steganografi memiliki banyak pengertiannya masing – masing, beberapa sumber mengetakan pengertian steganografi

a. Menurut (Munir 2004) mengatakan : Steganografi merupakan sebuah teknik menyembunyikan pesan data rahasia didalam sebuah wadah (media) digital sehingga keberadaan data rahasia tersebut tidak diketahui orang
b. Menurut (Ariyus 2006) mengatakan : Steganografi merupakan cabang ilmu yang mempelajari bagaimana menyimpan informasi rahasia didalam informasi

Jadi bisa dikatan bahwa steganografi merupakan sebuah teknik dari kelanjutan kriptografi yang artinya sama – sama digunakan untuk menjaga kerahasiaan dengan cara berbeda yang dimana kriptografi mengganti sebuah data rahasia menjadi *ciphertext* sedangkan steganografi memasukan data rahasia ke dalam sebuah objek.

2.11.2 Sejarah Steganografi

Sejarah teknik steganigrafi dimulai pada 4000 tahun yang lalu, yang dilahirkan pada peradaban mesir kuno yang dimana tulisan ini menceritakan kehidupan majikannya. Oleh karena itulah, tulisan mesir kuno yang menggunakan gambar sebagai tulisan dianggap sebagai steganografi pertama di dunia.

Setelah bangsa mesir menggunakan gambar sebagai tulisan, yunani merupakan bangsa yang menggunakan steganografi setelah mesir. Herodotus mendokumentasikan konflik antara Persia dan yunani, yang dimana Pada abad 50, Raja xerxes mengirimkan pesan kepada

pasukannya untuk menyerang yunani, akan tetapi pesan tersebut berhasil diterjemahkan oleh ahli tulis yang menyebabkan Persia dikalahkan oleh yunani. Penyebab kekalahan Persia dikarenakan Demartus yang seorang yunani mengabarkan bahwa Raja xerxes mau menyerbu yunani. Untuk mengabarkan ke pihak yunani dan tidak kelihatan oleh Raja xerxes, Darmatus menulis pesan dengan cara mengisi tabung kayu dengan lilin. Darmatus menulis sebuah pesan dipapan kayu lalu dimasukan ke tabung kayu yang di isi dengan lilin.

Selain cerita tentang bangsa mesir dan yunani, ada cerita lain di negara cina dengan menggunakan cara yang berbeda. Negara cina tidak menggunakan sebuah kertas untuk mengirimkan pesan, melainkan sebuah objek manusia sebagai tempat penyimpanan pesan tersebut. Manusia yang dijadikan objek penyembunyian pesan dengan cara menggunduli kepala, lalu dituliskan sebuah pesan yang berisi " La Wan " setelah rambut manusia yang dijadikan objek penyembunyian tumbuh dan menutupi tulisan tersebut, lalu dikirimkan pesan tersebut kepada orang lain. (Ariyus 2009).

2.11.3 Konsep – Konsep Steganografi

Dengan perkembangan teknologi saat ini, maka wadah yang digunakan juga berbeda tidak seperti dulu yang menggunakan kepala manusia atau tabung kayu yang menjadikan wadahnya. Dengan steganografi sekarang maka harus mengetahui media yang dapat digunakan sebagai wadah penampungnya. Beberapa media yang digunakan dalam teknik sebuah steganografi (Ariyus 2009).

a. Teks : Media Teks dapat digunakan dalam algoritma steganografi dengan menggunakan teknik NLP, sehingga teks yang telah disisipkan dengan pesan rahasia tidak akan curigai orang
b. Audio : Audio dapat digunakan dalam agoritma steganografi, format ini biasanya berkas dengan ukuran yang relative besar sehingga bisa menampung pesan rahasia dalam jumlah besar
c. Citra : Citra termasuk dalam wadah steganografi, format ini merupakan salah satu format file yang sering dipertukarkan dalam dunia internet.
d. Video : Format ini memang format yang ukuran filenya relatif sangat besar, tetapi jarang digunakan karena ukurannya yang terlalu besar mengurangi kepraktisannya.

Dengan mengetahui media yang digunakan dalam penyembunyiannya, maka harus mengetahui skema dalam proses steganografi dalam melakukan penyembunyian datanya. Dalam pendapat dari (Munir 2004) steganografi membutuhkan dua property yaitu wadah

penampung dan data rahasia yang akan disembunyikan, lalu ada beberapa istilah yang digunakan Dengan illustrasi konsep seperti ini seperti ini.

Gambar 2.16 Ilustrasi proses steganografi

a. Embedded Massage (pesan rahasia) : Embedded Massage merupakan pesan yang disembunyikan/pesan rahasia
b. Cover Object atau Stego Cover : Cover Object merupakan sebuah wadah yang digunakan untuk menyembunyikan pesan rahasia yang berupa Teks, Audio, Citra Digital, Video
c. Stego Object : Stego Object atau bisa disebut Objek wadah penampung yang memiliki sebuah pesan rahasia didalam wadah tersebut.

2.11.4 *Least Significant Bit* (LSB)

Least Significant Bit (LSB) merupakan sebuah metode yang digunakan dalam ilmu steganografi yang melakukan teknik subtitusi. Penyembunyian metode ini dilakukan dengan mengganti sebuah bit – bit data yang terkandung didalam citra digital, dengan cara mengganti representasi warna pixel – pixel yang ada di citra digital. 1 pixel citra berukuran 1 sampai 3 byte dan susunan bit dalam sebuah byte (1 byte = 8 bit), ada sebuah bit yang paling berarti (Most Significant Bit atau MSB) dan ada bit yang tidak paling berarti (Least Significant Bit atau LSB) dalam citra digital.

Contoh bit yang paling berarti (MSB) dan sebuah bit tidak berarti (LSB) misalnya 11010010, bit angka 1 (yang digaris bawahi) adalah bit MSB dan bit dengan angka 0 (yang digaris bawahi) merupakan bit LSB. Bit yang paling cocok dalam penyembunyian pesan adalah LSB, karena mengubah nilai byte tersebut satu lebih tinggi atau satu lebih rendah dari nilai sebelumnya. Jika sebuah warna digunakan berwarna biru, maka dengan pergantian bit LSB tidak mengubah warna secara berarti. Dengan keuntungan ini dimanfaatkan untuk penyembunyian, karena mata manusia tidak membedakan perubahan warna yang begitu kecil (Munir 2004).

Ilustrasi proses Steganografi menggunakan metode LSB pada sebuah gambar, dengan pixel – pixel sebagai berikut.

Table 2.8 Contoh binary pixel gambar

00000000	00000000	00000001	00000001	00000001	00000001	00000001	00000001
00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Lalu menyisipkan sebuah pesan ke dalam binary gambar yang ada di Table 2.8, sebagai contoh pesan yang dimasukan karakter "s", maka karakter sebut diubah ke binary terlebih dahulu

Table 2.9 ASCII karakter "s"

Karakter	**ASCII (Decimal)**	**Hexadecimal**	**Binary**
S	115	73	01110011

Maka binary dari karakter "s" dengan nilai 01110011 gabungkan ke binary yang ada didalam gambar menggunakan Logika True False. Logika yang digunakan yaitu operasi AND. Maka ilustrasi penyembunyiannya seperti berikut.

Table 2.10 Penggabungan binary pixel dan karakter

00000000	00000001	00000001	00000001	00000000	00000000	00000001	00000001
00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

2.11.5 Ukuran Data Yang Disembunyikan

Ukuran data yang disembunyikan tergantung pada ukuran citra penampung yang digunakan, maka semakin besar citra yang digunakan maka semakin banyak pesan yang dapat ditampung ke dalam sebuah citra. Dengan menggunakan rumus untuk menghitung jumlah maksimal yang dapat disembunyikan pada citra yaitu :

$$\text{Maksimal Data} = \frac{\square\square\square\square\square\square\square\square\ \square\square\square\square\square\square\ \square\ \square\square\square\square\square\square\ \square\square\square\square\square\square}{8\ \square\square\square\ /1\ \square\square\square\square\square\square\square\square\square} \qquad (2.15)$$

Dengan contoh menggunakan wadah penampung dengan citra 8 bit yang memiliki ukuran 256 x 256, yang artinya panjang citra memiliki ukuran 256 pixel dan lebar citra memiliki ukuran 256 pixel, sehingga menghasilkan 65536 pixel. Hasil tersebut dibagi 1

karakter, karana 1 karakter terdiri 8 bit maka karakter yang dapat disembunyikan pada media penampung setiap 8 pixel. Sehingga maksimal jumlah karakter yang dapat ditampung media dengan ukuran 256 x 256 pixel yaitu 8.192 byte/ 8.192 karakter yang dapat ditampung

Sedangkan jika menggunakan citra 24 bit, yang artinya memiliki 3 warna dalam 1 gambar. Maka dengan menggunakan ukuran pixel yang sama 65536, menghasilkan 196608 pixel. 196608 pixel ini didapat dengan cara mengkalikan citra sebelumnya dengan 3 warna dalam 1 gambar (65536 x 3). Lalu hasil dari penggunaan cita 24 bit dibagi dengan 8, menghasilan 24576 byte/ 24576 karakter yang dapat ditampung 1 citra dengan ukuran 256 x 256.

2.12 Android Studio

Android studio adalah sebuah *Intregeted Development Enviroment* (IDE) yang resmi dalam pengembangan aplikasi android. Android studio sendiri menawarkan banyak fiture dalam meningkatkan pembuatan sebuah aplikasi android seperti sistem build gradle, emulator, memberikan pengenalan fiture dasar android studio, dapat mengembangkan aplikasi kesemua perangkat android, Framework dan pengujian lengkap didukung menggunakan Bahasa C++ dan Java. Dengan android studio para pengembang bisa dengan leluasa berkarya menciptakan sebuah aplikasi tersendiri dan Open Source yang artinya terbuka untuk digunakan oleh semua orang.

2.13 Java

Bahasa Java di ciptakan oleh Sun Microsystem pada tahun 1995-an. Java merupakan Bahasa pemrograman yang multi platform dan multi device. Dengan menggunakan java ini, dapat menjalankan di semua komputer dan perangkat yang support pada java. Java memiliki beberapa fungsi dalam pembuatan yaitu Bahasa yang digunakan sederhana, dapat di pakai pada sistem Operasi manapun, mendukung native methode, dan adanya fiture GUI. Tidak hanya fungsi saja yang dimiliki java, java juga memiliki beberapa arsitektur dan arsitektur java dibagi menjadi 3 yaitu.

a. J2ME (java micro Edition) arsitektur yang dipergunakan pembuatan aplikasi handphone
b. J2SE (java 2 Standard Edition) arsitektur untuk membuat aplikasi yang sederhana atau standar
c. J2EE (Java 2 Enterprise Edition) arsitektur yang digunakan untuk membuat aplikasi yang sangat rumit.

2.14 Android

Android merupakan sistem operasi untuk perangkat bergerak (mobile) yang dikembangkan oleh android Inc. pada tahun 2005, android secara resmi dibeli oleh perusahaan yang bernama google. Dengan dibelinya android oleh google, google tetap memberikan source code (Sumber kode) sehingga android termasuk dalam software open source yang membuat banyak orang berkontribusi dalam mengembangkan aplikasi – aplikasi saat ini. Banyak perangkat handphone menggunakan OS (Sistem Operasi) android. Untuk setiap versi OS (Sistem Operasi) android dinamai dengan nama makanan berdasarkan huruf alphabet. Ada beberapa nama – nama yang digunakan dalam OS (Sistem Operasi) android dari yang lama hingga yang terbaru yaitu : Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, Kitkat, Lollipo, Marshmellow, Nougat, Oreo dan Pie.

2.15 ASCII

ASCII merupakan sebuah kode standar amerika yang digunakan sebagai kode standar internasional untuk digunakan sebagai pertukaran informasi. ASCII sendiri memiliki tabel yang dimana terdapat sebuah karakter huruf dan karakter simbol dengan total jumlah 256 karakter. ASCII merupakan sebuah bahasa komputer dengan nilai bit antara 0 dan 1, setiap karakter memiliki 8 bit. Didalam Tabel terdiri beberapa kolom yang mengambarkan sebuah karakter yaitu Nilai ASCII (*Decimal*), Nilai Unik (*Hexadecimal*) dan keterangan info karakter dari nilai ASCII maupun nilai Unik.

2.16 Penelitian Terdahulu

Berikut ini merupakan penilitian – penelitian terdahulu yang berkaitan dengan ilmu kriptografi dan steganografi dengan metode – metode yang digunakan untuk penelitian ini.

a. Penelitian yang dilakukan (Wibowo, Nilawati, and Suharnawi 2014). Penelitian ini berutujuan mengamankan pesan dengan menggunakan metode kriptografi affine. Hasil menunjukan bahwa algoritma affine dengan menggunakan kunci kombinasi, dapat diimplementasikan.
b. Penelitian yang dilakukan (Babu 2017). Penelitian ini melakukan modifikasi affine cipher dengan menambahkan karakter mencapai total 36 karakter dan karakter *plaintext* disusun terbalik. Hasil penelitian menunjukan, karakter penambahan mancapai total 36 karakter memiliki 12 kunci invers yang didapat, serta dengan konsep membalikan pesan terlebih dahulu menghasilkan chipertext yang terbalik juga.

c. Penelitian yang dilakukan (Juliadi et al. 2013). Penelitian ini melakukan modifikasi metode affine cipher dengan menambahkan jumlah karakter yang dipakai. Total karakter yang dipakai 40 karakter, dengan total karakter 40 memperbesar jumlah proses kriptanalisi memecahkan kunci untuk mendapatkan hasil pesan. Lalu affine cipher digabung dengan Vigenere yang menjadikan *ciphertext* sulit dipecahkan karena butuh 3 kunci untuk memecahkannya. Hasil dari penelitian ini dengan 2 metode kriptografi klasik dapat memperkuat sebuah pesan.

d. Penelitian yang dilakukan (Sari, Sulindawaty, and Sihotang 2017). Penelitian ini menggabungkan sebuah metode kriptografi dan Steganografi dalam pengamanan pesan. Teknik yang digunakan untuk mengamankan pesan menggunakan metode Hill Cipher dengan kunci 2 x 2 yang menghasilkan *ciphertext* dengan kunci tetap, *ciphertext* tersebut di sisipkan ke dalam gambar dengan menggunakan metode Steganografi sehingga pesan tersebut menyatu dengan gambar. Hasil dari penelitian ini membuat aplikasi yang berhasil mengamankan sebuah pesan ke dalam sebuah gambar.

e. Penelitian yang dilakukan (Barmawi and Hamdani 2016). Penelitian ini menggunakan algoritma *Hill Cipher* untuk mengamankan dokumen file, yang dimana menggunakan kunci matriks 2 x 2 dengan jumlah 95 karakter yang digunakan. Hasil penelitiannya mengatakan bahwa kekurangan algoritma yang mereka gunakan tidak menggunakan 256 karakter yang dapat di enkripsi, sehingga tidak memunculkan sebuah karakter simbol yang lebih acak

f. Penelitian yang dilakukan (Puspita and Wayahdi 2015). Penelitian ini melakukan penggabungan 3 metode dalam melakukan enkripsi terhadap sebuah pesan. Metode Hill Cipher dipenelitian ini menggunakan kunci 2 x 2 dengan jumlah total 255 karakter yang menghasilkan sebuah *ciphertext* dengan karakter simbol. Hasil penelitiannya mengatakan 3 metode ini cukup kuat dari segi keamanannya dan menyarankan menggunakan Matriks kunci yang lebih besar.

g. Penelitian yang dilakukan (Abidin et al. 2016). Penelitian ini melakukan percobaan dengan menggunakan aplikasi yang dibuat berbasis web dengan aplikasi lain dengan nama " Cryptool ", perbedaan yang didapat untuk berbasis web hanya untuk berupa data teks sedangkan untuk aplikasi Cryptool lebih bervariasi dan dapat memiliki tipe *hexadecimal* dan binary. Penelitian ini menghasilkan sebuah kesimpulan kekuatan *Advanced Encrypt Standard* terdapat pada pendistribusian kunci.

h. Penelitian (Anwar 2017). Penelitian ini berutujuan membuat aplikasi penyembunyian pesan dengan menggunakan algoritma AES sebagai enkripsi pesan dan menggunakan algoritma steganografi untuk menyembunyikan pesan Objek gambar. Penelitian ini membuat aplikasi berbasis Desktop dengan menghasilkan sebuah aplikasi yang berhasil diterapkan.

BAB III
METODOLOGI DAN PERANCANGAN

3.1 Analisis Kebutuhan

Analisi kebutuhan adalah suatu kegiatan yang menganalisis kebutuhan – butuhan dalam tahap perancangan aplikasi yang diperlukan guna untuk membangun suatu sistem perangkat lunak berbasis android. Pada tahap penelitian ini akan dilakukan membutuhkan kebutuhan dalam proses input data, data proses dan hasil proses data.

Aplikasi Multiple Kriptografi dan Steganografi ini digunakan untuk melakukan pengamanan pesan teks yang terenkripsi dan disembunyikan dalam media gambar yang menghasilkan proses yang baik ataupun tidak baik. Untuk melakukannya maka pesan teks dienkripsi dengan metode – metode yang sudah di tetapkan dengan cara mengubah sebuah pesan teks ke *ciphertext* lalu diproses kembali untuk melakukan penyembunyian pesan ke dalam media gambar. Untuk melakukan dekripsi pesan yang ada didalam gambar maka pengguna harus menggunakan aplikasi yang dibuat penelitian ini.

3.1.1 Analisis Kebutuhan Input

Input adalah suatu masukan data yang akan diolah sehingga menghasilkan sebuah proses informasi yang diinginkan. Kebutuhan yang dibutuhkan dalam proses input diantaranya

a. Masukkan sebuah media gambar yang berfungsi sebagai tempat penyimpanan informasi
b. Masukkan sebuah pesan teks yang ingin di enkripsi
c. Masukkan sebuah kunci (kunci dalam bentuk teks dengan maksimal 16 kata).
d. Masukkan sebuah kunci pertama yang digunakan untuk algoritma *Affine Cipher*
e. Masukkan sebuah kunci kedua yang dibutuhkan untuk shift/pergeseran

3.1.2 Analisis Kebutuhan Proses.

Proses adalah suatu pengolahan data yang didapat dari proses input untuk mengubah data menjadi data yang di inginkan. Kebutuhan yang dibutuhkan dalam proses aplikasi ini diantaranya

a. Proses pengecekan suatu input yang dimana jika menu input tidak lengkap maka sistem memberitahukan pengguna.

b. Proses enkripsi pesan teks menggunakan metode affine cipher dengan menggunakan 2 kunci dan Dekripsi proses pengubahan *ciphertext* yang dihasilkan Hill Cipher ke bentuk pesan semula.
c. Proses hasil *ciphertext* dari metode affine cipher diolah kembali dengan menggunakan metode Hill Cipher dengan kunci yang sudah ditetapkan yang berfungsi sebagai enkripsi dan untuk dekripsi, *ciphertext* diambil dari AES (Advanced Encrypt Standard).
d. Proses enkripsi dan dekripsi kunci dengan menggunakan metode Caesar Cipher yang digunakan untuk proses metode AES (Advanced Encrypt Standard).
e. Proses Eksnpansi kunci metode AES (Advanced Encrypt Standard) yang didapat dari proses Caesar
f. Proses Enkripsi Dan dekripsi AES (Advanced Encrypt Standard).
g. Proses *Embedding* yang bergunakan untuk memasukan *ciphertext* ke dalam sebuah media gambar
h. Proses *Extaracting* yang berguna untuk mengeluarkan isi pesan yang tersembunyi dalam media gambar.
i. Proses mendapatkan nilai Hash dari Media gambar yang terdapat dari proses *embedding*
j. Proses pengecekan File integritas sebelum dan sesudah pada File *stego image*.

3.1.3 Analisis Kebutuhan Output

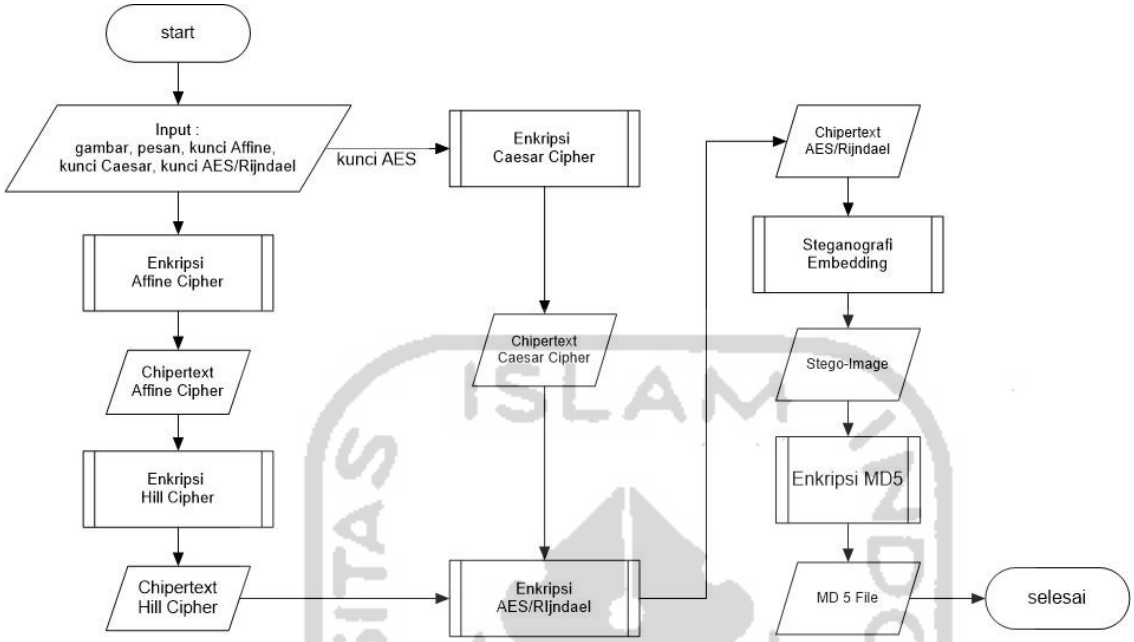
Output adalah sebuah keluaran dari hasil proses – proses data yang sudah di kelola. Kebutuhan yang dibutuhkan dalam proses Output ini dalam aplikasi yaitu

a. Gambar yang telah dilakukan dalam proses embedding.
b. Hidden-Massage yang telah disembunyikan didalam sebuah media gambar
c. Hashing dari File Hidden Message dapat digunakan
d. Pesan dapat dibaca kembali setelah di extract dari gambar.
e. Status kecocokan Hashing dari File Hidden Message yang dikirim

3.2 Diagram Alir

Diagram alir adalah sebuah langkah – langkah rancangan alur dari sebuah program yang dimana berisikan sebuah Flowchart. Flowchart merupakan sebuah alur yang terdiri dari bagan – bagan dengan simbol tertentu dalam menggambar proses sebuah program secara detail dan memberikan informasi hubungan antara suatu proses dengan proses lain hingga akhir.

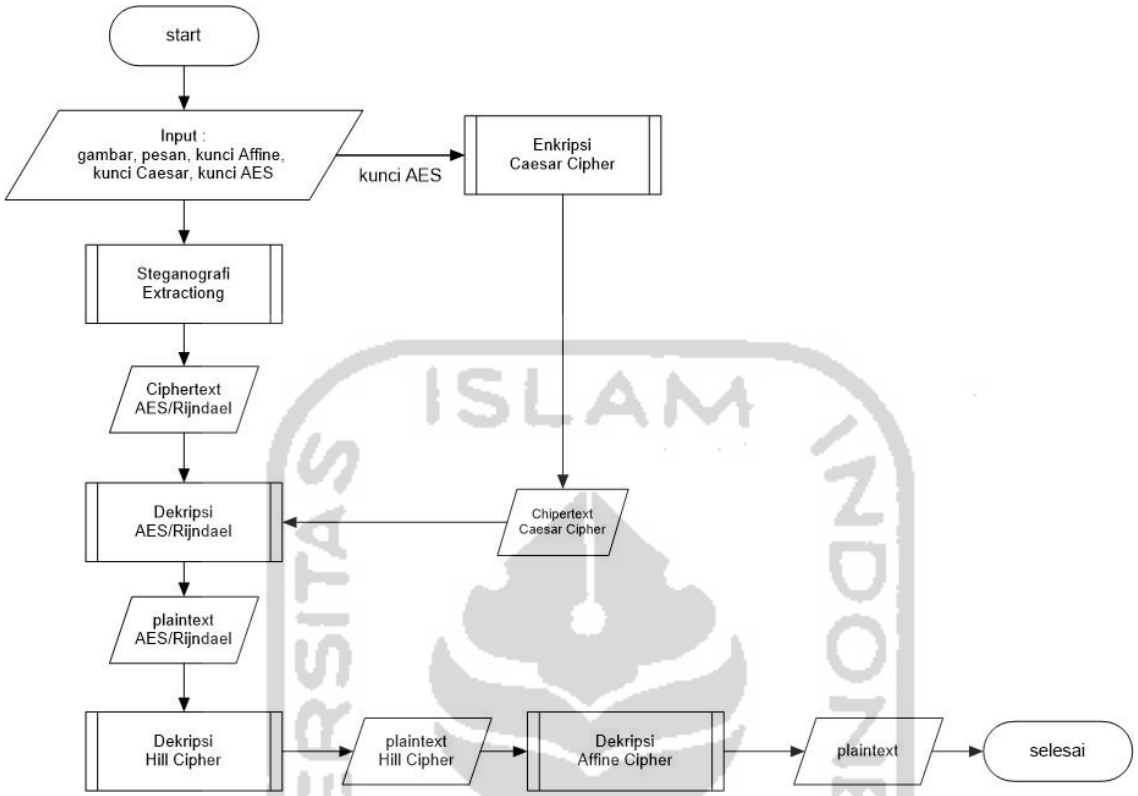
3.2.1 *Flowchart* Enkripsi dan *Embedding* (Encoding)



Gambar 3.1 *Flowchart* enkripsi dan *Embedding*

Seperti yang terlihat pada Gambar 3.1, menjelaskan bahwa untuk melakukan proses *embedding* sebuah pesan ke dalam sebuah gambar harus melalui banyak alur yang dimana sebuah pesan tersebut harus melalui proses enkripsi. Setiap pesan yang diubah menjadi *ciphertext* harus melalui proses metode – metode kriptografi yang digunakan, yaitu AffineCipher, Hill Cipher, dan AES. Hasil dari Proses tersebut lalu melakukan Proses *embedding* pesan *ciphertext* ke dalam sebuah objek gambar yang nantinya menghasilkan sebuah Stego-image. Stego-image ini merupakan gambar yang sudah disisipkan sebuah *ciphertext* (pesan rahasia). Setelah itu, melakukan enkripsi MD5 yang berguna mendapatkan nilai identik dari File Stego-image.

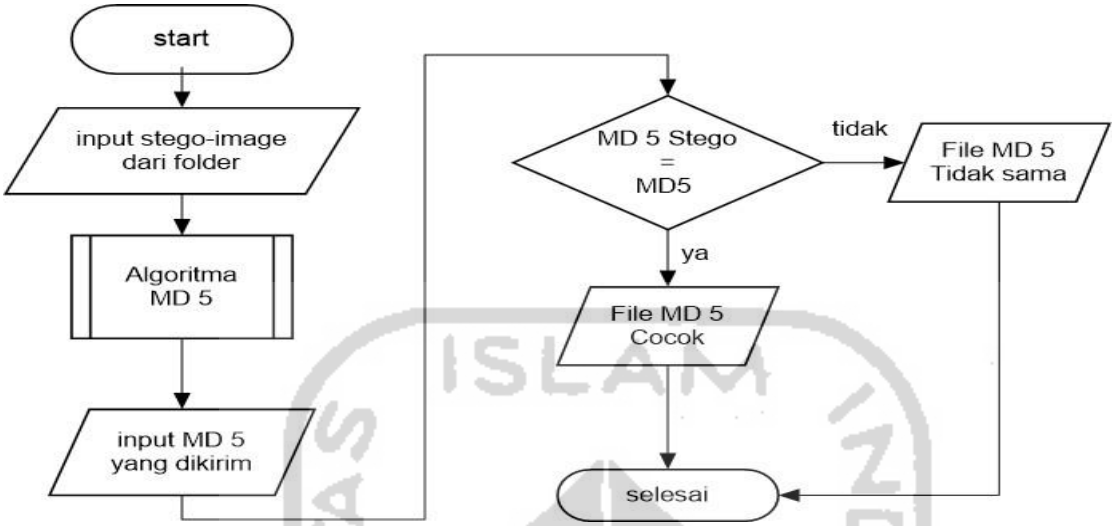
3.2.2 *Flowchart* dekripsi dan *Extracting* (Decoding)



Gambar 3.2 *Flowchart* dekripsi dan *extracting*

Pada Gambar 3.2 yang dapat dilihat, Proses dalam mendapatkan pesan kembali maka harus melakukan input seperti pada proses Encrypt. Tetapi pada proses ini alur pengambilan pesan terbalik dari proses encrypt, yang dimana proses awal mengekstraksi pesan yang ada didalam gambar, lalu di dekripsi dengan metode – metode kriptografi yang digunakan dengan urutan yang berbeda dari proses enkripsi dan *embedding* yaitu AES/Rijndael, Hill Cipher, dan Affine Cipher. Maka pesan yang tersembunyi didalam gambar dapat di baca kembali dengan teks yang sama pada proses penyembunyiannya.

3.2.3 Flowchart File Checksum



Gambar 3.3 *Flowchart* File Checksum

Gambar 3.3 merupakan alur dari proses pengecekan integritas File *Stego Image* yang dikirim melalui media. Tahap awal proses, menginput *stego image* yang diambil dari folder, setelah itu *stego image* di enkripsi menggunakan MD 5 untuk mendapatkan nilai indetik.

Setelah MD 5 File *Stego image* didapat, wajib memasukan MD 5 yang dikirim oleh si pemberi untuk mengetahui nilai indetiknya sama atau tidak.

3.2.4 Flowchart proses *Affine Cipher*

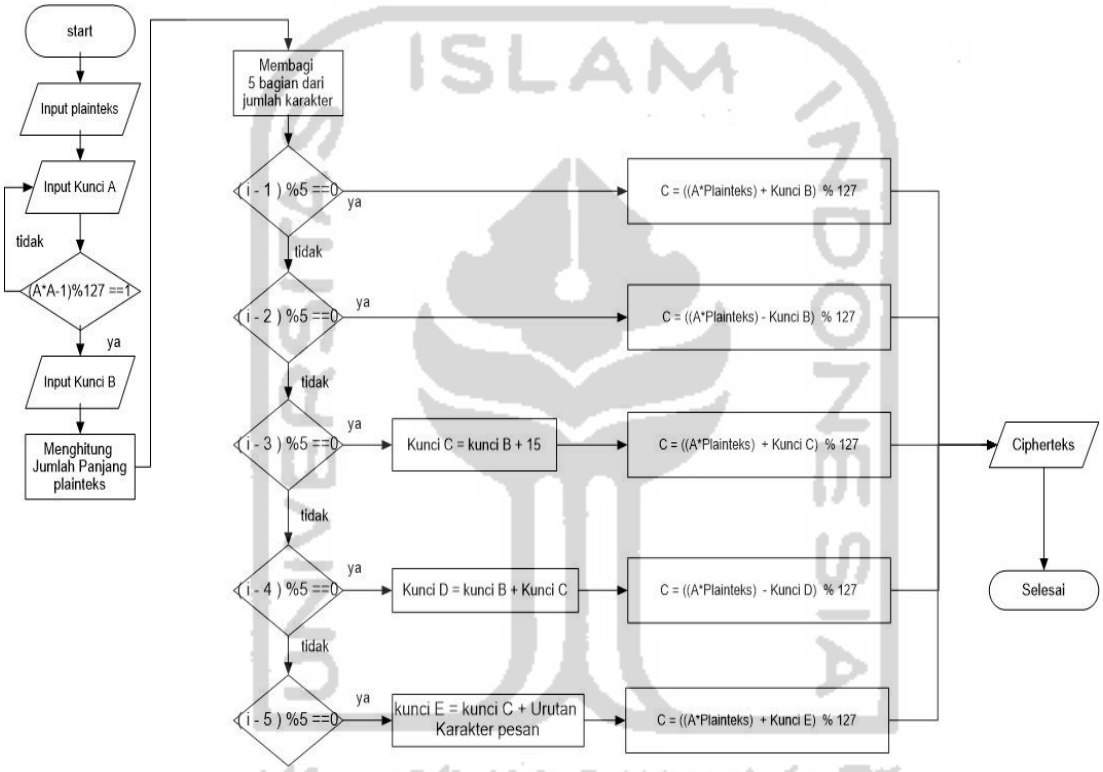
Flowchart Proses Affine Cipher ini menggambarkan alur dari proses enkripsi dan dekripsi yang terdapat pada Flowchart encode dan decode dengan secara sistematis. Seperti penjelasan pada bab sebelumnya, Metode *affine cipher* memiliki 2 kunci yang digunakan untuk melakukan enkrip dan dekrip sebuah pesan. Algoritma *Affine cipher* ini memiliki banyak kelemahan jika hanya menggunakan karakter dengan total 26 karakter, dikarenakan ruang kunci yang digunakan memiliki hanya 12 kunci yang dapat digunakan untuk melakukan enkrip dan dekrip. 12 kunci tersebut yang memiliki nilai invers, yang dimana nilai invers tersebut digunakan pada proses dekrip. Lalu proses perhitungan karakter menjadi *ciphertext* dengan metode sama menghasilkan nilai yang sama.

Maka untuk menutupi kelemahaan tersebut, pada penelitian ini *affine cipher* yang digunakan dimodifikasi dengan total 127 karakter agar mendapatkan jumlah ruang kunci yang

lebih besar, dengan menggunakan total 127 karakter maka kunci yang dapat digunakan dengan total 126 kunci yang memiliki invers dan perhitungannya dibagi menjadi 5 proses untuk mendapatkan hasil *chipertext* yang lebih acak. 5 proses tersebut memiliki kunci pergeseran yang berbeda – beda.

Maka Alur dalam melakukan Proses Enkripsi dan Dekripsi akan di jabarkan dibawah ini dengan langkah – langkahnya sebagai berikut.

a. *Flowchart* enkripsi *Affine cipher*



Gambar 3.4 *Flowchart* enkripsi *Affine Cipher*

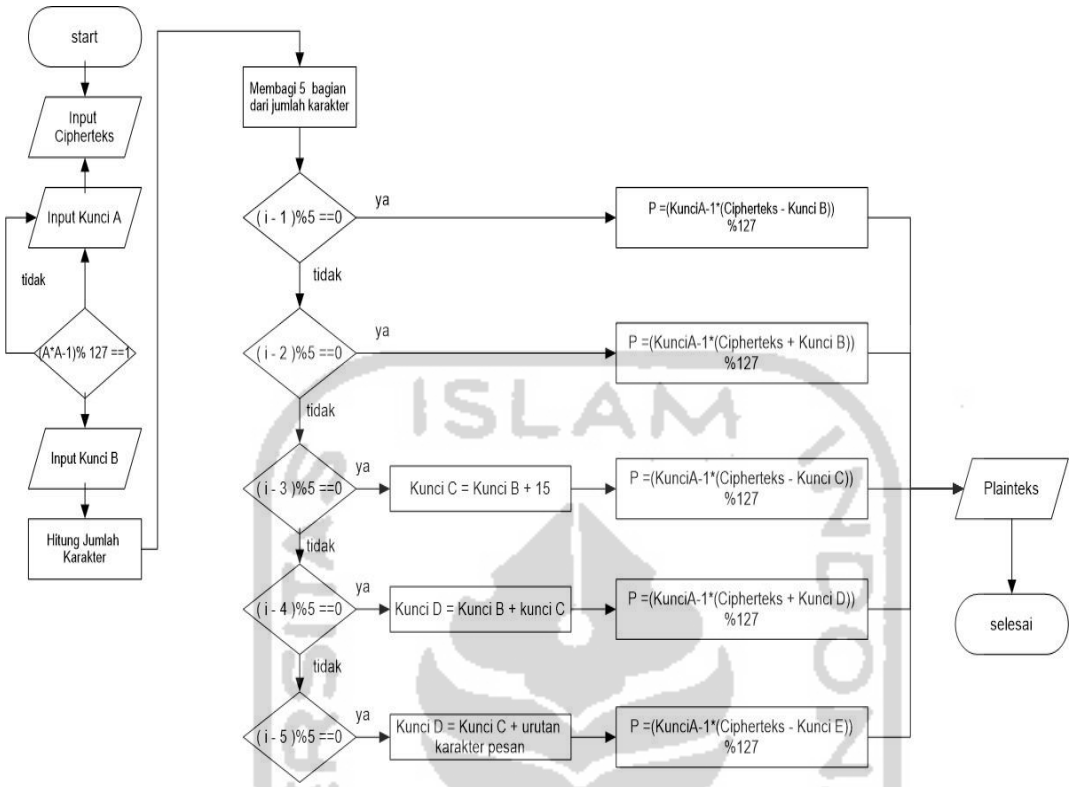
Pada Gambar 3.4 ini proses berjalannya program dalam melakukan enkripsi sebuah pesan menjadi sebuah *ciphertext*. Dalam melakukan proses enkripsi maka harus memenuhi syarat – syarat yang sudah dibuat sehingga dapat menghasilkan hasil keluaran yang semestinya.

Maka berikut Langkah – langkah dalam melakukan proses enkripsi dengan menggunakan algoritma metode *Affine Cipher* :

1. Masukkan Kunci Pertama (Kunci A)
2. Masukkan Kunci Kedua (Kunci B)

3. Periksa kunci pertama apakah memiliki nilai Invers dari modulo 127. Jika Tidak, maka menginput ulang kunci yang memiliki nilai invers modulo 127.
4. Menghitung panjang/jumlah Karakter pesan (*plaintext*).
5. Mengurutkan panjang/jumlah karakter lalu menghitung jumlah karakter dengan modulo 5
6. mengubah karakter pesan menjadi *numerik* ASCII
7. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 1) mod 5 == 0 maka dihitung dengan persamaan C = (*Kunci A* * *Pesan* + *Kunci B*) mod 127
8. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 2) mod 5 == 0 maka dihitung dengan persamaan C = (*Kunci A* * *Pesan* - *Kunci B*) mod 127
9. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 3) mod 5 == 0 maka mencari kunci C terlebih dahulu dengan rumus Kunci C = kunci B + 15, setelah didapat maka menghitung dengan rumus C = (*Kunci A* * *Pesan* + *KunciC*) mod 127
10. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 4) mod 5 == 0 maka mencari kunci D terlebih dahulu dengan rumus Kunci C = kunci B + Kunci C, setelah didapat maka menghitung dengan rumus C = (*Kunci A* * *Pesan* - *KunciD*) mod 127
11. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 5) mod 5 == 0 maka mencari kunci E terlebih dahulu dengan rumus Kunci D = kunci C + Urutan Karakter pesan, setelah didapat maka menghitung dengan rumus C = (*Kunci A* * *Pesan* + *KunciE*) mod 127
12. Mengubah *numerik* ASCII menjadi Karakter sehingga menghasilkan *Ciphertext*.

b. *Flowchart* dekripsi *Affine Cipher*



Gambar 3.5 *Flowchart* dekripsi *Affine Cipher*

Pada Gambar 3.5 proses alur dalam melakukan dekrip sebuah *ciphertext* menjadi sebuah Pesan kembali, dalam prosesnya tidak jauh beda dengan proses pada Gambar 3.4. Akan tetapi perbedaanya pada perhitungannya, yang dimana kunci A yang digunakan yaitu invers yang didapat dari kunci A dan kunci B juga prosesnya berbeda.

Maka berikut ini penjelasan proses langkah – langkahnya melakukan dekrip *ciphertext* menjadi pesan (*plaintext*) kembali dengan metode Affine Cipher :

1. Masukkan Pesan Rahasia (*ciphertext*)
2. Masukkan Kunci Pertama (Kunci)
3. Masukkan Kunci Kedua (Kunci B)
4. Periksa Kunci Pertama memiliki Invers modulo 127, Jika tidak maka melakukan input ulang. Jika memiliki Invers modulo 127 maka digunakan pada proses selanjutnya dengan persamaan *Kunci* A^{-1}.
5. Menghitung panjang/jumlah karakter pesan rahasia (*ciphertext*)

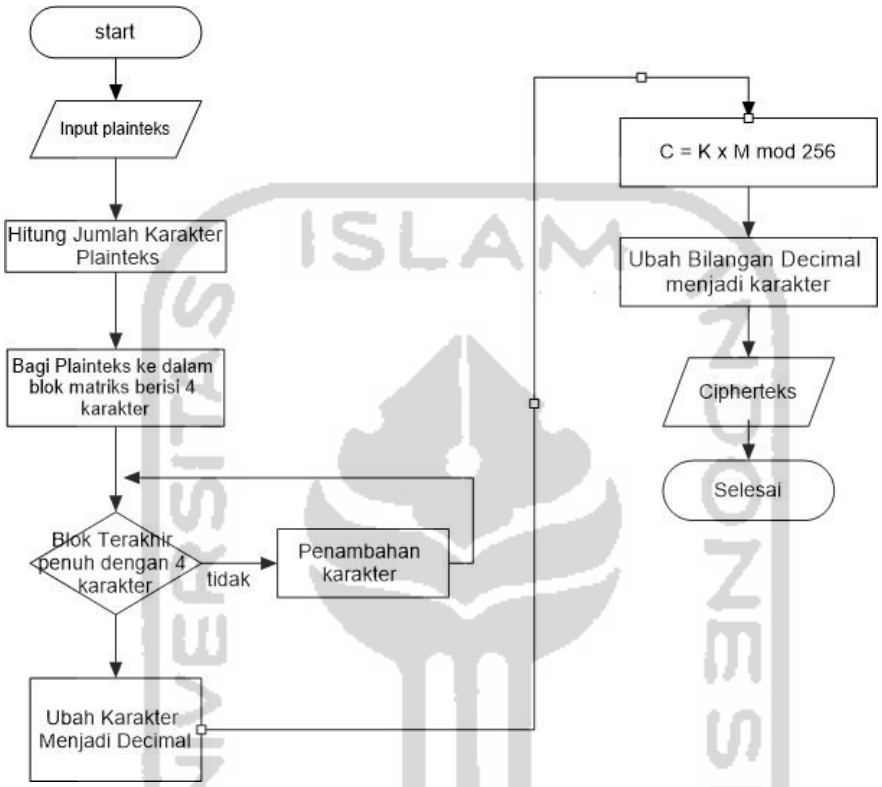
6. Mengurutkan panjang/jumlah karakter lalu menghitung jumlah karakter dengan modulo 3
7. Mengubah pesan rahasia menjadi *numerik* ASCII.
8. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 1) mod 5 == 0 maka dihitung dengan persamaan *C* = (*Kunci A* * *Pesan* - *Kunci B*) mod 127
9. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 2) mod 5 == 0 maka dihitung dengan persamaan *C* = (*Kunci A* * *Pesan* + *Kunci B*) mod 127
10. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 3) mod 5 == 0 maka mencari kunci C terlebih dahulu dengan rumus Kunci C = kunci B + 15, setelah didapat maka menghitung dengan rumus *C* = (*Kunci A* * *Pesan* - *KunciC*) mod 127
11. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 4) mod 5 == 0 maka mencari kunci D terlebih dahulu dengan rumus Kunci C = kunci B + Kunci C, setelah didapat maka menghitung dengan rumus *C* = (*Kunci A* * *Pesan* + *KunciD*) mod 127
12. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 5) mod 5 == 0 maka mencari kunci E terlebih dahulu dengan rumus Kunci D = kunci C + Urutan Karakter pesan, setelah didapat maka menghitung dengan rumus *C* = (*Kunci A* * *Pesan* - *KunciE*) mod 127
13. Mengubah kembali *numerik* ASCII menjadi karakter yang menghasilkan pesan kembali (*plaintext*).

3.2.5 Flowchart proses *Hill Cipher*

Flowchart Proses Hill Cipher ini menggambarkan alur dari proses enkripsi dan dekripsi yang terdapat pada Flowchart *encode* dan *decode* dengan secara sistematis. Seperti penjelasan pada bab sebelumnya, *Hill Cipher* menggunakan rumus perkalian matriks antara kunci dengan pesan, yang dimana perkalian antara baris dan kolom. Algoritma *Hill Cipher* memiliki banyak kelemahan jika menggunakan matriks kunci berukuran 2 x 2 dengan total 26 karakter, yang artinya setiap kunci hanya menggunakan nilai antara 1 sampai 26 dan pada penelitian (Puspita and Wayahdi 2015) yang dilakukan, menggunakan kunci matriks 2 x 2 dengan total 256 karakter yang dapat digunakan dan menyarankan menggunakan kunci lebih besar.

Sehingga pada penelitian ini, *Hill Cipher* yang digunakan dimodifikasi dengan menggunakan matriks kunci berukuran 4 x 4 dengan nilai tetap, kemudian jumlah karakter yang digunakan dengan jumlah total 256 karakter agar menghasilkan banyak karakter acak.

a. *Flowchart* enkripsi *Hill Cipher*



Gambar 3.6 *Flowchart* enkripsi *Hill Cipher*

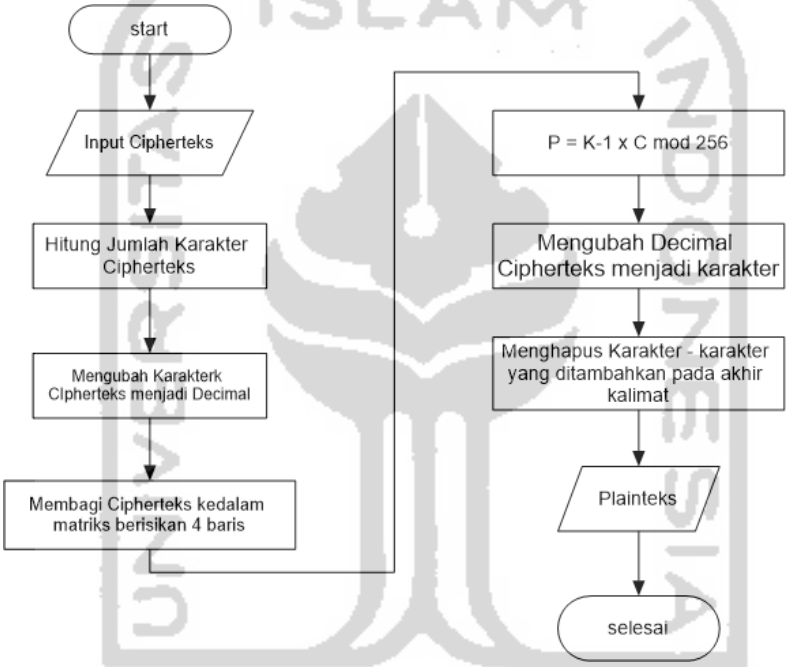
Pada Gambar 3.6 merupakan sebuah proses alur dimana pesan tersebut di enkrip dengan metode *Hill Cipher* sehingga menghasilkan sebuah *ciphertext*. Untuk melakukan proses enkripsi, maka harus melewati proses – proses yang sudah ditentukan sehingga dapat menghasilkan sebuah keluaran yang semestinya.

Dibawah ini langkah – langkah proses enkripsi alogirtma metode Hill Cipher yang dapat dijabarkan sebagai berikut.

1. Masukkan pesan (*plaintext*)
2. Masukkan Kunci Matriks
3. Hitung jumlah karakter pesan (*plaintext*).
4. Membagi pesan (*plaintext*) menjadi blok – blok karakter yang masing – masing 4 karakter

5. Jika blok – blok karakter tidak memenuhi 4 karakter maka penambahan karakter " . " sebanyak kurang blok karakter yang kurang
6. Mengubah karakter pesan (*plaintext*) menjadi *numerik* ASCII.
7. Menghitung blok – blok pesan (*plaintext*) dengan kunci Matriks dengan persamaan C = P x K mod 256 sehingga menghasilkan sebuah *numerik* ASCII lain.
8. Ubah numerik ASCII menjadi karakter kembali, maka menghasilkan karakter – karakter *ciphertext*. Flowchart Dekrip Hill Cipher

b. *Flowchart* dekripsi *Hill Cipher*



Gambar 3.7 *Flowchart* dekripsi *Hill Cipher*

Pada Gambar 3.7 merupakan proses dari Dekripsi *Hill Cipher*, yang dimana proses yang dilakukan sama seperti proses Enkrpsi *Hill Cipher* akan tetapi dalam proses ada beberapa perbedaan yang dimana kunci Matriks yang digunakan yaitu hasil Matriks Invers.

Dibawah ini langkah – langkah proses dekrip alogirtma metode Hill Cipher yang dapat dijabarkan sebagai berikut.

1. Masukkan pesan rahasia (*ciphertext*)
2. Masukkan Kunci Matriks
3. Hitung jumlah karakter pesan rahasia (*ciphertext*).

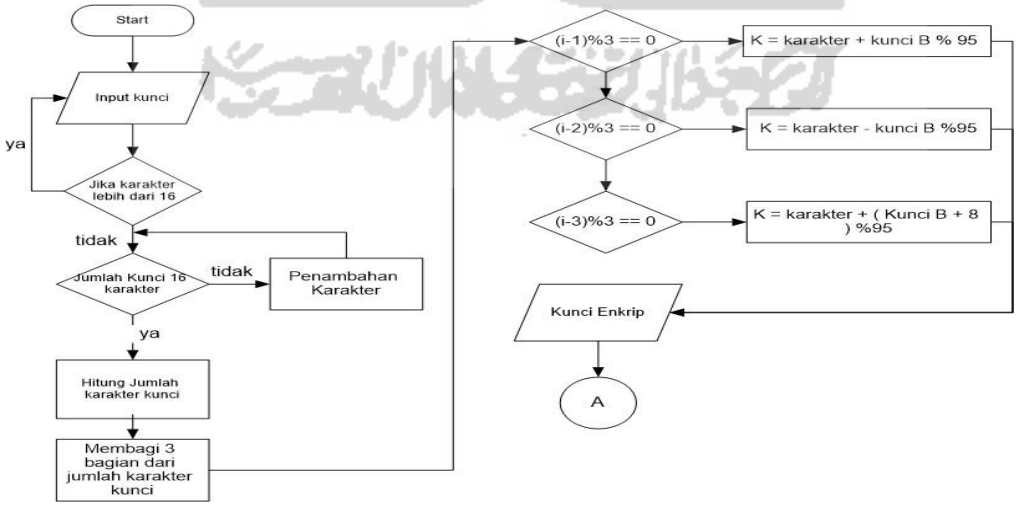
4. Membagi pesan (*plaintext*) menjadi blok – blok karakter yang masing – masing 4 karakter
5. Mengubah pesan rahasia menjadi *numerik* ASCII
6. Menghitung blok – blok pesan (*plaintext*) dengan kunci Matriks dengan persamaan C = P x K mod 256 sehingga menghasilkan sebuah *numerik* ASCII lain.
7. Mengubah *numerik* ASCII menjadi karakter kembali
8. Menghapus karakter – karakter tambahan yaitu karakater " . " sehingga menghasilkan jumlah pesan yang sama.

3.2.6 *Flowchart* proses *Caesar Cipher* dan AES

Seperti penjelasan sebelumnya, *Advanced Encrypt Standar* termasuk dalam kriptografi yang memiliki tingkat keamanan yang tinggi sehingga susah untuk di pecahkan. Akan tetapi bukan berarti *Advanced Encrypt Standar* selalu aman. Kelemahan *Advanced Encrypt Standar* yaitu termasuk kriptografi dengan algoritma " Simetris ", yang artinya kunci yang digunakan dalam melakukan proses enkrip dan dekrip menggunakan kunci yang sama.

Pada penelitian ini, *Advanced Encrypt Standar* dimodifikasi dengan metode *Caesar Cipher*, sehingga pendistribusian kunci diketahui belum dapat dipastikan dapat digunakan dalam memecahkan pesan rahasia. Maka akan dijelaskan sebagai berikut alur dari modifikasi metode *Advanced Encrypt Standar* dengan *Caesar Cipher*.

a. *Flowchart* enkripsi *Caesar Cipher*



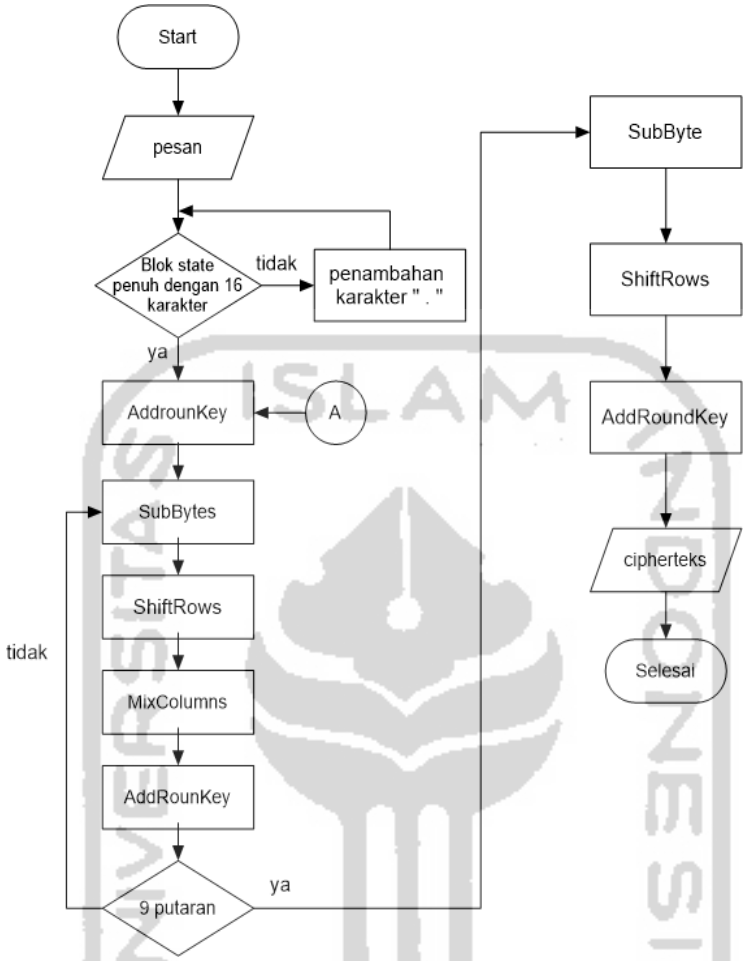
Gambar 3.8 *Flowchart* enkripsi *Caesar Cipher*

Pada Gambar 3.8 merupakan proses melakukan pengenkripsian kunci yang nantinya digunakan sebagai kunci *Advanced Encrypt Standart*. Fungsi pengekripsian kunci ini digunakan untuk pendistribusian kunci agar kunci yang terlihat bukan berarti kunci yang dapat digunakan di proses *Advanced Encrypt Standart* lain. Kunci dirubah sehingga menghasilkan *ciphertext* kunci. *Ciphertext* kunci tersebut yang digunakan sebagai kunci *Advanced Encrypt Standart*.

Dibawah ini langkah – langkah proses enkripsi kunci dengan metode *Caesar Cipher* yang dapat dijabarkan sebagai berikut.

1. Masukkan Kunci
2. Masukkan Kunci Kedua (Kunci B)
3. Menghitung panjang/jumlah Karakter kunci.
4. Periksa kunci, apakah kunci lebih dari 16 karakter atau kurang dari 16 karakter. Jika kunci lebih dari 16 karakter maka input ulang, jika kunci kurang dari 16 karakter maka penambahan karakter " . ".
5. Mengurutkan panjang/jumlah karakter kunci lalu menghitung jumlah karakter dengan modulo 3
6. Mengubah karakter kunci menjadi *numerik* ASCII
7. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 1) modulo 3 == 0 maka dihitung dengan persamaan $C = (kunci + Kunci\ B) \bmod 95$
8. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 2) modulo 3 == 0 maka dihitung dengan persamaan $C = (kunci - Kunci\ B) \bmod 95$
9. Jika panjang/jumlah karakter memenuhi dengan persamaan (Panjang Karakter – 3) modulo 3 == 0 maka dihitung dengan persamaan $C = (kunci + (Kunci\ B + 8)) \bmod 95$
10. Mengubah *numerik* ASCII menjadi Karakter sehingga menghasilkan *ciphertext* kunci.

b. *Flowchart* enkripsi *Advanced Encrypt Standard* (AES)



Gambar 3.9 *Flowchart* enkripsi AES

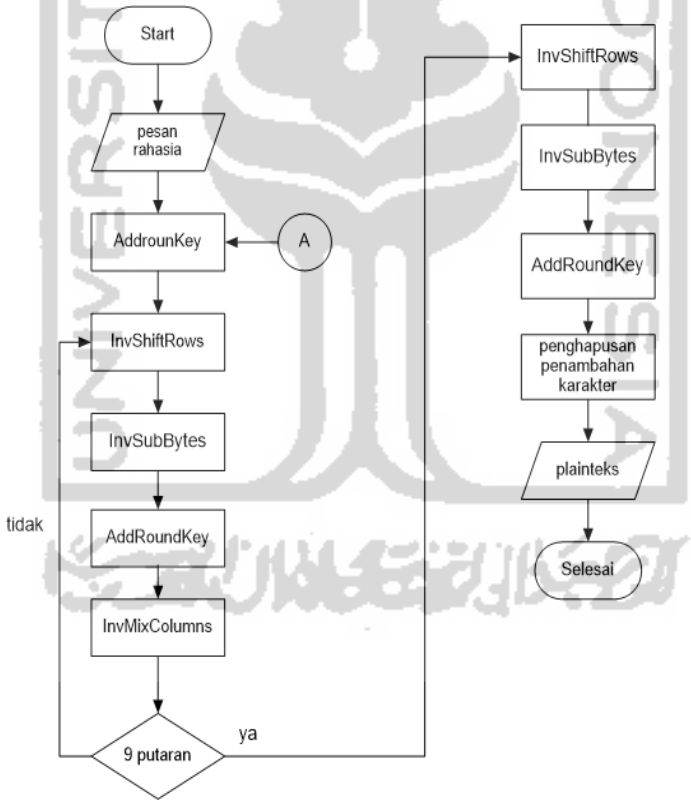
Pada Gambar 3.9 merupakan alur dari proses Enkripsi AES, yang dimana proses ini digunakan untuk Mengubah pesan menjadi pesan rahasia (*chipertext*). Didalam gambar terdapat simbol lingkaran, yang bisa disebut sebuah konektor. Konektor ini digunakan untuk menghubungkan proses dengan proses lain, yang artinya proses itu didapat dari proses lain. Lingkaran itu merupakan hasil dari proses *Caesar Cipher* yang digunakan sebagai kunci *Advanced Encrypt Standar* untuk melakukan enkripsi sebuah pesan.

Setelah didapat kunci dari *Caesar Cipher* maka berikut ini langkah – langkah proses enkripsi pesan dengan metode *Advanced Encrypt Standar* yang dapat dijabarkan sebagai berikut dengan menggunakan tabel.

Table 3.1 Proses putaran enkripsi AES

Round	Function
-	AddRound Key
0	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
1	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
2	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
3	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
4	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
5	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
6	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
7	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
8	Sub Bytes – Shift Rows – Mix Columns - AddRoundKey
9	Sub Byte – Shift Rows – Add Around Key

c. *Flowchart* dekripsi *Advanced Encrypt Standard* (AES)



Gambar 3.10 *Flowchart* dekripsi AES

Pada Gambar 3.10 merupakan proses alur dalam melakukan dekripsi pesan rahasia menjadi pesan kembali, dalam proses dekripsi AES yaitu melakukan input pesan rahasia

dan kunci. Kunci yang digunakan harus melalalui proses enkripsi metode *Caesar Cipher* yang nantinya dapat digunakan untuk melakukan proses dekripsi.

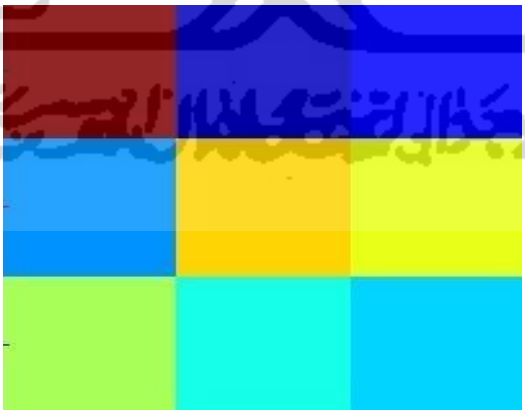
Setelah didapat kunci dari *Caesar Cipher* maka berikut ini langkah – langkah proses dekripsi pesan rahasia (*ciphertext*) menjadi pesan (*plaintext*) dengan metode *Advanced Encrypt Standar* yang dapat dijabarkan sebagai berikut dengan menggunakan tabel.

Table 3.2 Proses putaran dekripsi AES

Round	Function
-	AddRound Key
0	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
1	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
2	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
3	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
4	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
5	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
6	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
7	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
8	InvShiftRows – InvSubByte – AddRoundKey – InvMix Columns
9	InvShiftRows – InvSubByte – AddRound Key

3.2.7 *Flowchart* proses Steganografi

Seperti penjelasan pada bab sebelumnya, metode Steganografi *Least Significant Bit* (LSB) digunakan untuk menyembunyikan data informasi ke dalam sebuah gambar yang dimana konsep alurnya diilustrasikan sebagai berikut.



Gambar 3.11 Ilustrasi wadah gambar Objek

Gambar 3.11 merupakan sebuah ilustrasi gambar yang digunakan untuk melakukan penyembunyian data dengan ukuran pixel 3 x 3 yang memiliki nilai RGB. Dengan

menggunakan metode steganografi LSB, pixel – pixel tersebut digunakan sebagai wadah untuk meyembunyikan bit – bit pesan. Dalam prosesnya dicontohkan sebagai berikut.

Sebelum melakukan proses penyembunyian bit – bit data pesan, wadah tersebut harus diketahui berapa data maksimal yang dapat digunakan. Pada ilustrasi, wadah yang digunakan seperti pada Gambar 3.11 yang dimana memiliki ukuran pixel 3 x 3. Dengan mengetahui ukuran pixel tersebut, dapat dihitung untuk mencari batasan maksimal data, yang dirumuskan seperti pada persamaan 2.15. dengan contoh sebagai berikut.

$$\text{Maksimal Data} = \frac{(3 \times 3) \times 3}{8 \text{ bit} / 1 \text{ byte}} = 3{,}37 \text{ byte (3 karakter)}$$

.

Seperti dapat dilihat, maksimal data yang dapat disembunyikan yaitu 3,37 byte. Artinya dengan ukuran gambar 3 x 3 hanya dapat menampung dengan maksimal 3 karakter saja. Perhitungan ini didapat dari ukuran pixel gambar (3 x 3), representative Citra 24 bit (3 citra Biner), serta data yang disembunyikan perkarakter (1 byte /8 bit).

Setelah data maksimal karakter yang dapat ditampung didapat, maka proses yang dilakukan yaitu mengambil nilai bit – bit pixel gambar dan bit – bit karakter pesan. Pesan yang digunakan yaitu “Aku”, berikut proses ilustrasi steganografi melakukan penyembunyian data

Pertama – tama mengubah gambar pixel tersebut menjadi ukuran biner dengan nilai RGB yang di ilustrasikan pada gambar berikut.

R = 01011100 G = 00110011 B = 10110010	R = 00001111 G = 11100110 B = 00011110	R = 11111000 G = 00011110 B = 01010011
R = 10110010 G = 01011100 B = 00110011	R = 11100110 G = 00011110 B = 00001111	R = 00011110 G = 01010011 B = 11111000
R = 00110011 G = 10110010 B = 01011100	R = 00011110 G = 00001111 B = 11100110	R = 01010011 G = 11111000 B = 00011110

Gambar 3.12 Ilustrasi Bit RGB Wadah Gambar

Setelah mendapatkan bit – bit RGB dari pixel gambar, selanjut yaitu mengubah bit – bit karakter pesan yang di ilustrasikan pada tabel berikut.

Table 3.3 Mengubah Karakter Pesan menjadi bit

A	k	u
01000001	01101011	01110101

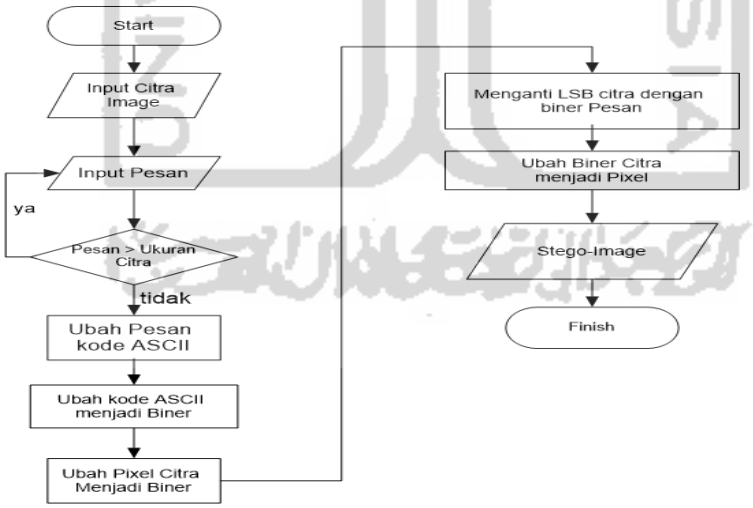
Setelah mendapatkan nilai bit pesan dan gambar, langkah selanjut yaitu melakukan proses penyisipan yang di ilustrasikan pada gambar berikut.

R = 01011100 G = 00110011 B = 10110010	R = 00001110 G = 11100110 B = 00011110	R = 11111000 G = 00011111 B = 01010010
R = 10110011 G = 01011101 B = 00110010	R = 11100111 G = 00011110 B = 00001111	R = 00011111 G = 01010010 B = 11111001
R = 00110011 G = 10110011 B = 01011100	R = 00011111 G = 00001110 B = 11100111	R = 01010011 G = 11111000 B = 00011110

Gambar 3.13 Ilustrasi penyisipan karakter

Dengan perhitungan data maksimal di awal proses, dapat mengetahui berapa karakter yang dapat ditampung. Oleh sebab itu pada penelitian saat ini, maka alur dalam melakukan proses *embedding* dan *extracting* gambar dengan pesan akan dijelaskan dibawah ini.

a. *Flowchart Embedding* pesan



Gambar 3.14 *Flowchart embedding* steganografi

Pada Gambar 3.14 merupakan sebuah proses alur *Embedding*, yang dimana pesan tersebut di sembunyikan ke dalam gambar dengan menggunakan metode Steganografi *Least Significant Bit* sehingga menghasilkan sebuah *Stego-image*. Untuk melakukan proses

Embedding pesan, maka harus melewati proses – proses yang sudah ditentukan sehingga dapat menghasilkan sebuah keluaran yang semestinya.

Dibawah ini merupakan langkah – langkah proses Steganografi LSB (*Least Significant Bit*) yang dapat dijabarkan sebagai berikut :

1. Masukkan gambar dari gallery
2. Masukkan Pesan yang ingin disembunyikan
3. Periksa pesan apakah panjang pesan lebih besar dari total jumlah pixel. Jika iya, maka menginput kembali pesan dengan total jumlah pixel.
4. Mengubah pesan menjadi kode biner ASCII
5. Mengubah pixel citra menjadi biner
6. Melakukan pergantian biner akhir pixel dengan biner pesan dengan biner pixel.
7. Mengubah biner pixel menjadi pixel citra kembali
8. Menghasilkan sebuah Stego Image.

b. *Flowchart Extracting* pesan

Gambar 3.15 *Flowchart extracting* steganografi

Pada Gambar 3.15 merupakan proses dari *Extracting*, yang dimana ini digunakan untuk melakukan pengambilan kembali pesan yang tersebunyi didalam sebuah pixel – pixel yang terdapat didalam gambar.

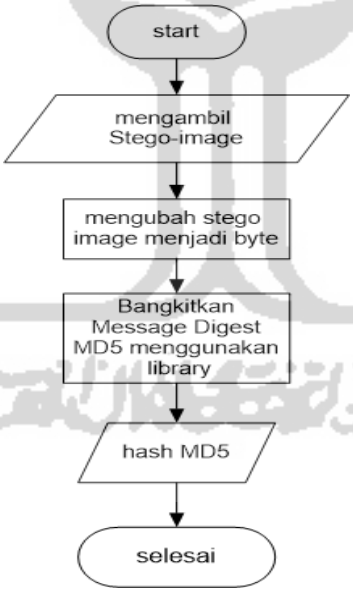
Dibawah ini langkah – langkah proses dalam melakukan *Extracting* pesan yang dapat dijabarkan sebagai berikut

1. Input Stego image
2. Mengubah pixel citra stego image menjadi binary
3. Mengambil LSB citra biner pesan dan menghasilkan biner pesan kembali
4. Mengubah biner pesan menjadi kode ASCII
5. Mengubah Kode ASCII menjadi karakter kembali

3.2.8 Flowchart proses MD5

Seperti pada penjelasan pada bab sebelumnya, Metode *Message Digest* 5 ini bisa berfungsi sebagai algoritma yang digunakan untuk menandakan keaslian sebuah file yang dimana prosesnya melakukan perubahan sebuah file menjadi notasi hex yang bernilai identik, yang dapat digunakan untuk melakukan pengecekan file keasliannya.

Pada penelitian ini, maka alur dalam melakukan proses enkripsi File menggunakan algortima MD 5 akan dijelaskan dibawah ini



Gambar 3.16 *Flowchart* MD 5

Pada Gambar 3.16 merupakan proses mendapatkan nilai identik dari enkripsi MD5 , yang digunakan untuk mendapatkan sebuah notasi hex sebagai nilainya.

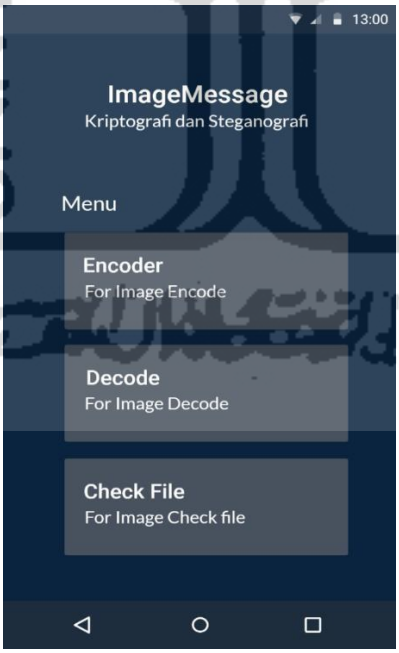
Dibawah ini langkah – langkah proses dalam melakukan *Extracting* pesan yang dapat dijabarkan sebagai berikut

1. Mengambil Stego Image dari hasil proses *Embedding*
2. Mengubah Stego Image menjadi byte
3. Memproses Byte Stego Image dengan algoritma MD5 yang tersedia di library
4. Mendapatkan nilai identik (*Hashing*) dari file Stego Image

3.3 Rancangan Antar Muka

Rancangan antar muka berguna untuk tujuan mencari sebuah *user interface* yang optimal dari pembangunan sebuah aplikasi dengan faktor – faktor permasalahan yang terjadi dan kebutuhan yang ada, sehingga menghasilkan aplikasi yang baik. Dalam perencangan ini, membangun sebuah aplikasi dengan melakukan kombinasi antara perangkat lunak dan perangkat keras untuk diimplementasikan antarmuka yang dikembangkan sehingga menghasilkan sebuah *user interface* yang *friendly* bagi pengguna.

3.3.1 Antarmuka Menu Utama

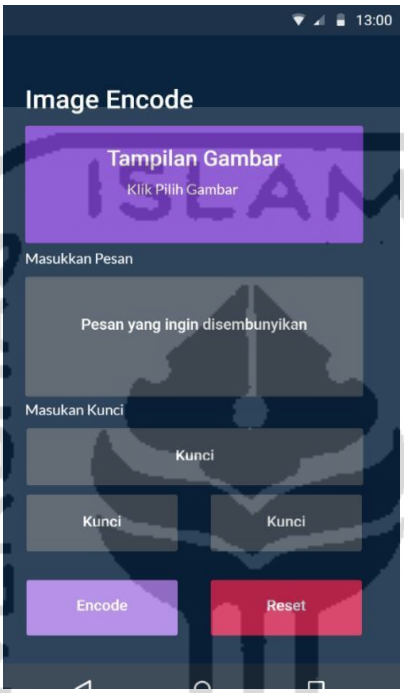


Gambar 3.17 Menu Utama Aplikasi

Desain Antar Muka menu utama merupakan halaman yang muncul saat pengguna membuka aplikasi *Image Message*, didalam menu utama terdapat 3 tombol yang berfungsi

untuk mengarahkan ke menu – menu lain yang dimana terdapat 3 menu, yaitu menu encode, menu decode, dan menu *Check File*. Perancangan menu encode dan menu decode dapat dilihat pada Gambar 3.18 dan Gambar 3.20.

3.3.2 Antarmuka Menu Encode



Gambar 3.18 Menu Encode Aplikasi

Desain antarmuka menu encode ini, menjelaskan tentang kegunaan – kegunaan dari tampilan aplikasi, yang digunakan untuk menjalankan sebuah proses melakukan encode yang terdapat Gambar 3.1, antarmuka ini berisikan sebuah masukkan yang akan mendapatkan hasil sebuah proses keluaran, yang dimana dalam desain ini terdapat sebuah tampilan form dan tombol. Untuk fungsinya, maka akan di jelaskan dibawah ini:

a. Button pilih gambar

   Button pilih gambar ini berfungsi untuk mengambil gambar dari dalam gallery handphone yang nantinya digunakan pada saat proses Embedding. Setelah gambar dipilih maka akan menampilkan gambar pada “ Tampilan Gambar ”.

b. Form input masukkan pesan

   EditText masukkan ini berfungsi sebagai masukkan sebuah teks pesan yang digunakan untuk melakukan proses perubahaan pesan menjadi pesan acak.

c. Form input kunci

EditText ini berfungsi sebagai masukkan kunci dengan bentuk karakter yang digunakan untuk proses perubahaan karakter pesan menjadi karakter acak, kunci ini digunakan pada proses Caesar Cipher lalu digunakan pada proses AES sebagai kunci

d. Form input kunci pertama

EditText kunci pertama ini berfungsi sebagai masukkan kunci dengan bentuk *numerik* atau bisa dibilang penomoran, kunci pertama ini digunakan pada saat proses melakukan enkrip dan dekrip dengan metode Affine Cipher

e. Form input kunci kedua

EditText kunci kedua ini berfungsi sebagai input kunci dengan bentuk *numerik*, perbedaan kunci kedua dengan kunci pertama, yaitu digunakan pada proses enkrip dan dekrip metode Affine Cipher dan Caesar Cipher.

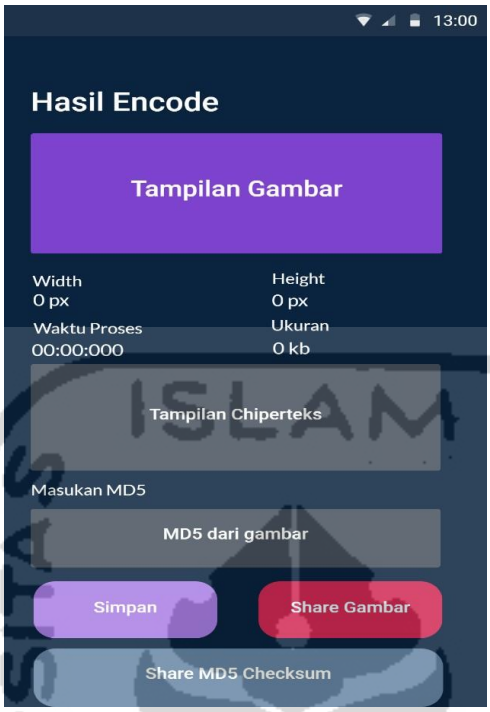
f. Button encode/enkrip pesan

Button encode/enkrip ini berfungsi untuk melakukan proses – proses penyisipan pesan ke dalam sebuah gambar, pada proses ini program akan melakukan pengecekan terlebih dahulu sebelum proses dilakukan.

g. Button reset/mengulang

Button reset/mengulang ini berfungsi untuk menghapus semua inputan jika terjadi beberapa kesalahan, sehingga lebih praktis dengan tombol ini.

3.3.3 Antarmuka Menu Hasil Encode



Gambar 3.19 Menu hasil Encode

Desain antarmuaka menu hasil encode pada Gambar 3.19 ini berfungsi menampilkan hasil yang didapat dari proses encode terpenuhi, yang dimana hasilnya akan ditampilkan pada menu ini. Terdapat beberapa tampilan dengan fungsi yang berbeda yaitu *Imageview*, *Textview*, dan *Button*. Untuk lebih jelasnya, maka akan dijelaskan dibawah ini fungsinya

a. Image view

   Image view disini berfungsi menampilkan stego image/ wadah penampung hasil dari encode.

b. Textview informasi

   Textview informasi disini menampilkan informasi tentang lebar dan tinggi wadah penampung, proses waktu berlangsungnya encode, dan ukuran gambar wadah penampung.

c. Textview *ciphertext*

   Textview *ciphertext* disini menampilkan sebuah pesan rahasia yang sudah di enkripsi pada proses sebelumnya.

d. Textview *Chekcsum* MD5

   Textview Checksum MD5 ini, menampilkan informasi hash yang didapat dari wadah penampung/ *Stego-image* yang digunakan untuk pegecekan file.

e. Button simpan

Button simpan ini berfungsi untuk menyimpan gambar *stego-image* ke dalam folder yang ada di handphone.

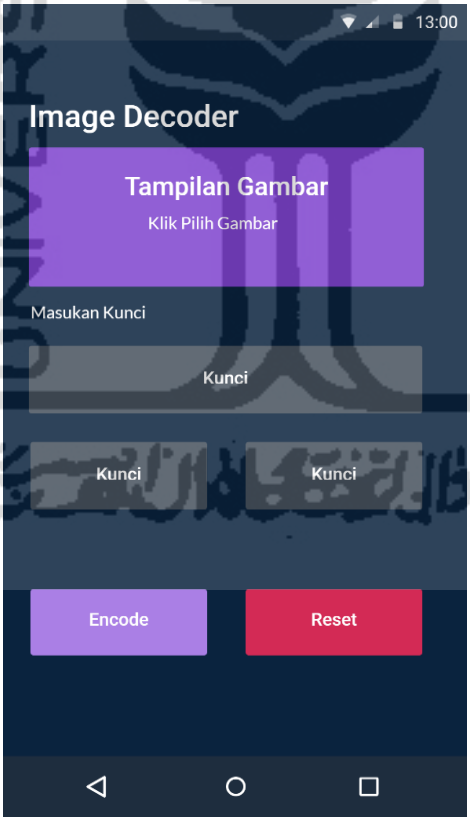
f. Button Share Image

Button Share Image ini berfungsi untuk melakukan share ke media social yang dimana pada saat button share image di klik maka gambar *stego-image* saja yang akan dikirim ke beberapa media yang dapat digunakan.

g. Button Share MD5

Button Share MD5 ini berfungsi untuk melakukan sharing/berbagi informasi hash yang didapat dari *stego-image* yang dimana, button ini hanya membagi informasi dari Textview Checksum MD5.

3.3.4 Antarmuka Menu Decode



Gambar 3.20 Menu decode aplikasi

Desain Antar Muka menu Decode ini menjelaskan tentang kegunaan – kegunaan dari tampilan yang digunakan untuk menjalankan sebuah proses melakukan decode yang terdapat

pada Gambar 3.2, antarmuka ini birisikan sebuah masukkan yang dapat menghasilkan sebuah proses keluaran, yang dimana dalam desain ini terdapat sebuah tampilan form dan tombol. Maka akan di jelaskan dibawah ini:

a. Button pilih gambar

   Button pilih gambar ini berfungsi untuk mengambil gambar dari dalam gallery handphone yang nantinya digunakan pada saat proses ekstrakting sebuah pesan yang terdapat didalam gambar

b. Form input kunci

   EditText ini berfungsi sebagai masukkan kunci dengan bentuk karakter yang digunakan untuk proses perubahaan karakter pesan menjadi karakter acak, kunci ini digunakan pada proses Caesar Cipher lalu digunakan pada proses AES sebagai kunci

c. Form input kunci pertama

   EditText kunci pertama ini berfungsi sebagai masukkan kunci dengan bentuk *numerik* atau bisa dibilang penomoran, kunci pertama ini digunakan pada saat proses melakukan enkrip dan dekrip dengan metode Affine Cipher

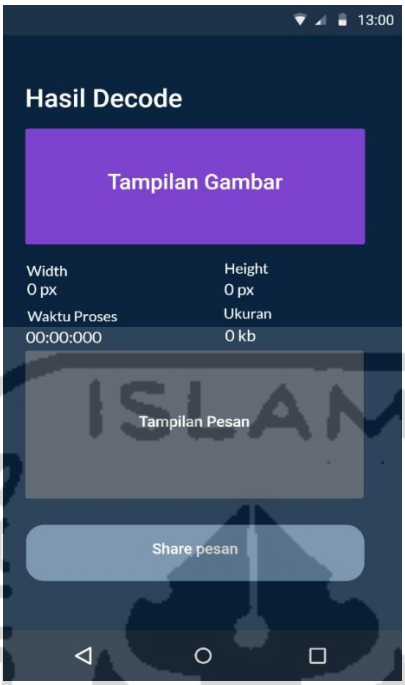
d. Form input kunci kedua

   EditText kunci kedua ini berfungsi sebagai input kunci dengan bentuk *numerik*, perbedaan kunci kedua dengan kunci pertama, yaitu digunakan pada proses enkrip dan dekrip metode Affine Cipher dan Caesar Cipher.

e. Button Decode/dekrip pesan

   Button decode/dekrip ini berfungsi untuk melakukan proses – proses ekstraksi pesan yang terdapat didalam gambar, pada proses ini program akan melakukan pengecekan terlebih dahulu sebelum proses dilakukan.

3.3.5 Antarmuka Menu Hasil Decode



Gambar 3.21 Antar muka hasil decode

Desain antarmuaka menu hasil decode ini berfungsi menampilkan hasil proses extracting dan dekripsi, yang diamana hasilnya akan ditampilkan pada menu ini. Terdapat tampilang dengan fungsi yang berbeda, yaitu ImageView,TextView dan Button. Untuk lebih jelasnya, maka akan dijabarkan dibawah ini fungsi - fungsinya

a. Image view

   Image view disini berfungsi menampilkan stego image/ wadah penampung hasil decode.

b. Textview informasi

   Textview informasi disini menampilkan informasi tentang lebar dan tinggi wadah penampung, proses waktu berlangsungnya encode, dan ukuran gambar wadah penampung.

c. Button

   Button disini berfungsi untuk memberikan pesan yang didapat pada stego image untuk dapat dikirim ke berbagai platform yang digunakan

3.3.6 Antarmuka Menu File Checksum

Gambar 3.22 Antar muka File Checksum

Desain antarmuaka menu file checksum ini berfungsi pengecekan keaslian sebuah file stego image yang dikirim melalui media pengirim, guna untuk mengetahui terjadi perubahan pada file stego image tersebut. Dalam tampilan ini memiliki beberapa fungsi yang berguna untuk mengetahui kondisi file, dimana fungsi tersebut ImageView, TextView, EditText, dan Button. Fungsi ini memiliki kerjanya masing – masing, berikut penjelasan dari fungsi – fungsi tersebut.

a. Image View

   Image view disini berfungsi melakukan pengambilan gambar stego image dari folder, yang nantinya akan ditampilkan pada aplikasi

b. Textview MD 5

   Textview disini berfungsi menampilkan nilai identik MD 5 dari stego image yang sudah di input

c. EditText Input MD5

EditText disini berfungsi untuk memasukan MD 5 yang diberi pengirim ke pada penerima, guna untuk membandingan dengan Original MD 5

d. Button Process

Button disini berfungsi melakukan proses perbandingan antara MD 5 original dengan MD 5 input untuk melihat apakah file itu berubah apa tidak

e. Button Reset

Button disini berfungsi untuk menghapus inputan gambar maupun inputan MD 5 jika terjadi kesalahan pada proses input.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan menjelaskan hasil dari implementasi dan pengujian aplikasi yang diperoleh berdasarkan pada penjelasan – penjelasan yang sudah di paparkan pada bab – bab sebelumnya. Pembahasan dalam bab ini meliputi kebutuhan perangkat lunak, kebutuhan perangkat keras, implementasi antarmuka, *source code algoritma* dan hasil dari pengujian serta analisis pengujian yang dimana mengenai penggunaan multiple kriptografi dan Steganografi yang ada didalam judul penelitian ini.

4.1 Kebutuhan Perangkat Lunak dan Perangkat Keras

Dalam pembuatan dan pengujian aplikasi Multiple kriptografi dan Steganografi pada penelitian ini, menggunakan spesifikasi perangkat lunak dan perangkat keras yang digunakan seperti dibawah ini.

4.1.1 Spesifikasi Perangkat Lunak

Pada proses perancangan dan pengembangan sistem aplikasi, terdapat beberapa komponen perangkat lunak yang digunakan sebagai alat bantu dalam pembuatan aplikasi multiple kriptgrafi dan steganografi ini, diantaranya

a. Windows 10 (64-bit)

   Windows 10 64-bit adalah sistem operasi yang digunakan pada perangkat laptop untuk membuat maupun perancangan aplikasi yang akan digunakan pada ponsel android

b. Android 9

   Android 9 yang memiliki kode nama " Pie " adalah Sistem operasi yang digunakan pada perangkat ponsel dalam menjalankan aplikasi yang dibuat

c. Bahasa pemrograman java

   Pemrograman java adalah sebuah bahasa pemrograman yang digunakan untuk membangun sistem aplikasi steganografi untuk dapat digunakan proses - prosesnya

d. Aplikasi Android Studio

   Android Studio merupakan sebuah textEditor yang ter-install diperangkat laptop, yang berguna untuk membangun aplikasi berbasis android hingga dapat digunakan.

4.1.2 Spesifikasi Perangkat Keras

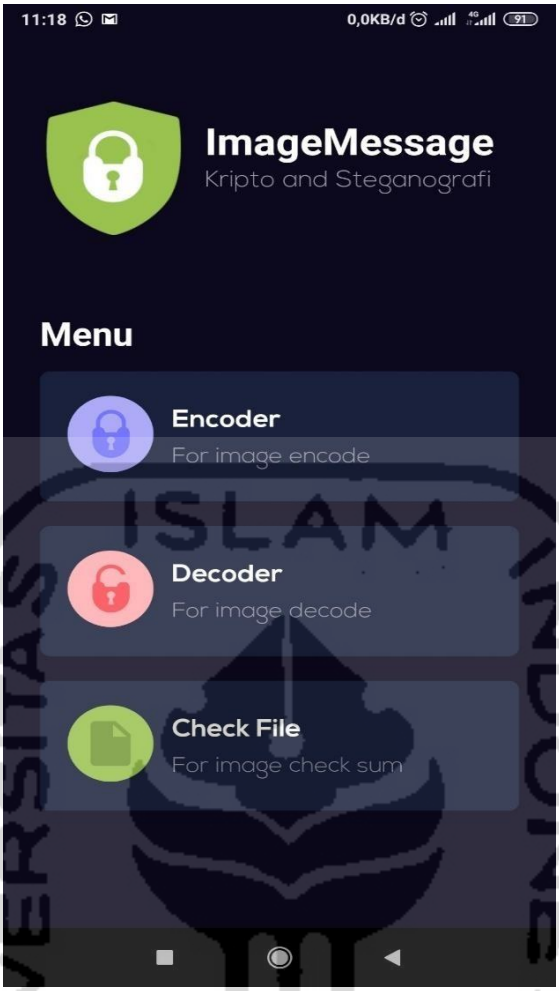
Pada proses pengembangan dan penggunaan aplikasi, terdapat beberapa komponen perangkat keras yang digunakan untuk pembuatan aplikasi dan menjalan aplikasi yang dibuat. Perangkat keras tersebut dibutuhkan untuk mendapatkan sebuah hasil yang maksimal. Terdapat 2 perangkat keras yang digunakan pada penelitian ini, antara lain Laptop dan Ponsel.

Berikut spesifikasi perangkat keras yang digunakan dalam pembuatan dan penggunaan aplikasi multiple kriptografi dan steganografi ini.

a. Spesifikasi Perangkat Keras Laptop
    1. Processor Intel Core i5-8900H dengan VGA Nvidia Geforce GTX 1050
    2. RAM 8 GB
    3. Harddisk 1 TB
    4. Monitor 12 Inc
b. Spesifikasi Perangkat Keras Ponsel
    1. Processor Octa-core Max 1.8GHz/ snapdragon636
    2. RAM 3 GB
    3. Penyimpanan 32 GB
    4. Lebar Layar 5.99 Inc

4.2 Implementasi Antar Muka

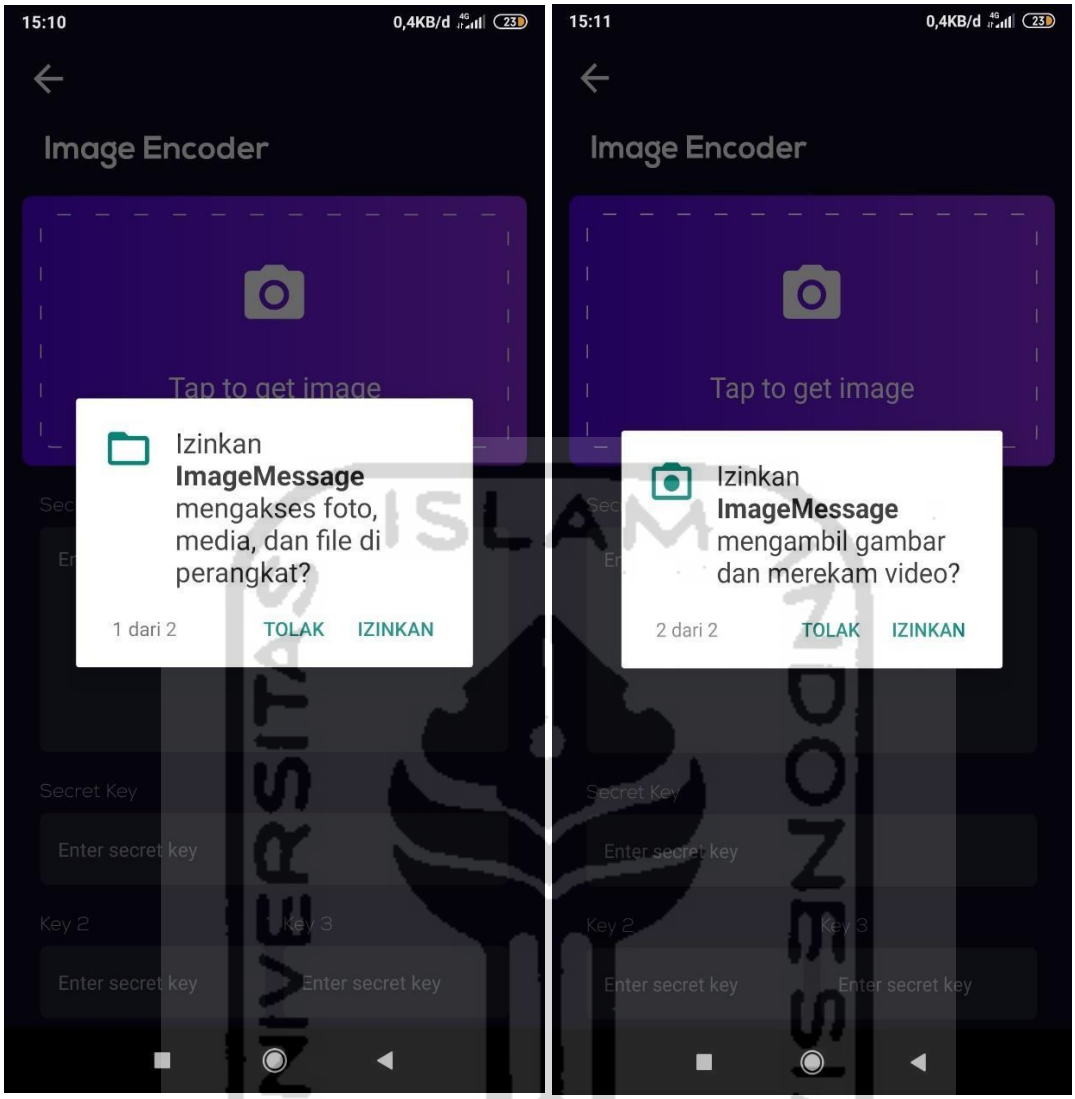
Pada implentasi antarmuka, penelitian ini membuat 3 buah halaman antarmuka yang terdiri dari halaman encode, halaman decode, dan halaman *file checksum*. Setiap halaman menandakan sebuah proses yang akan dilakukan pengguna untuk menggunakan aplikasi tersebut. Bentuk antarmuka halaman utama dapat dilihat seperti ini.



Gambar 4.1 Halaman Utama Aplikasi Image Message

Dari Gambar 4.1 dapat dilihat pada halaman utama aplikasi terdapat 3 menu utama yang digunakan untuk melakukan proses multiple kriptografi dan steganografi. Pertama, tombol " Menu Encode", tombol ini digunakan untuk ke menu proses "*Enkripsi dan Embedding*" yang dimana digunakan untuk melakukan pengenkripsian pesan dan disembunyikan ke dalam gambar. Kedua, tombol "Menu Decode", tombol ini digunakan untuk ke menu proses "*Extraction dan Dekripsi*" yang digunakan untuk melakukan ekstraksi pesan pada gambar, lalu melakukan dekrip pesan yang disembunyikan. Serta terdapat menu *Check File* yang digunakan untuk memasuki menu pengecekan keaslian File dengan membandingan nilai identik dari MD5.

Sebelum dapat mengakses menu – menu yang tersedia, sistem akan meminta akses perizinan untuk menggunakan fasilitas – fasilitas yang terdapat di handphone. Maka tampilan antarmuka perizinan yang digunakan aplikasi seperti ini.

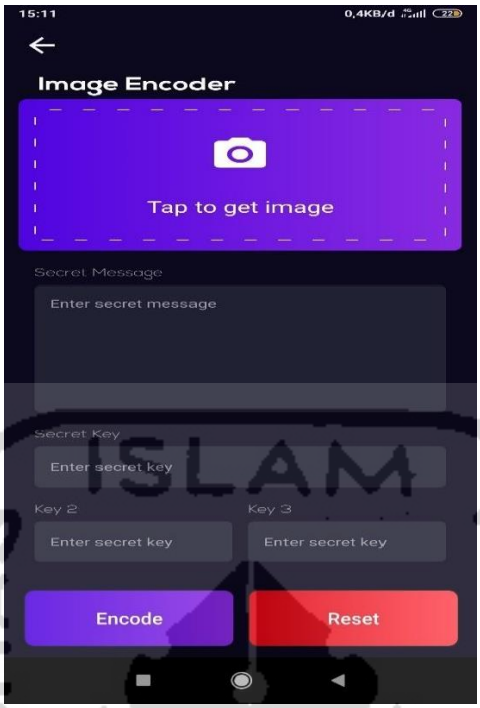


Gambar 4.2 Izin Akses Aplikasi

Perizinan aplikasi pada Gambar 4.2 digunakan untuk meminta hak akses perizinan yang akan digunakan dalam kebutuhan aplikasi, yang dimana digunakan untuk mengakses fasilitas – fasilitas data yang ada diperangkat *handphone.* Dengan fungsi ini, maka pengguna aplikasi sudah mengizinkan aplikasi untuk mengakses Foto media, mengambil gambar dan mengakses file yang terdapat di perangkat.

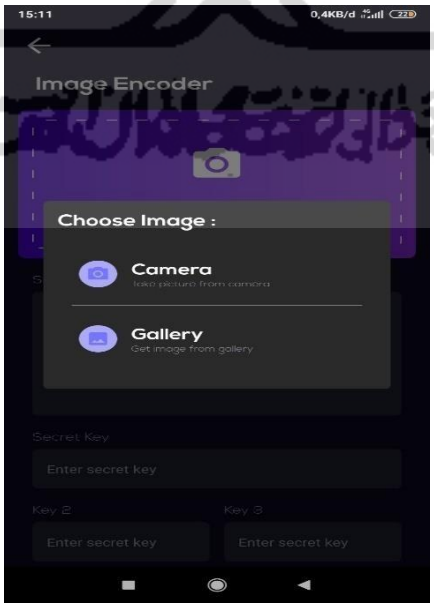
4.2.1 Implementasi Enkripsi dan *Embedding* (Encode)

Proses Enkripsi dan Embedding atau yang bisa disebut encode digunakan untuk melakukan proses pengubahan pesan dan penanaman pesan ke dalam sebuah objek yang dimana dalam prosesnya dilakukan dengan mengikuti aturan – aturannya. Antarmuka Encode dapat dilihat seperti ini jika pengguna memasuki “ Menu Encode ”.



Gambar 4.3 Halaman menu encode

Gambar 4.3 merupakan tampilan menu encode yang dimana terdapat tombol pilih gambar, masukan pesan, dan masukkan kunci. Dalam aturannya pengguna wajib mengisi semua proses yang dibutuhkan untuk dapat melakukan proses penyembunyian pesan terhadap objek gambar. Jika tombol pilih gambar di tekan, maka tampilannya akan seperti ini.



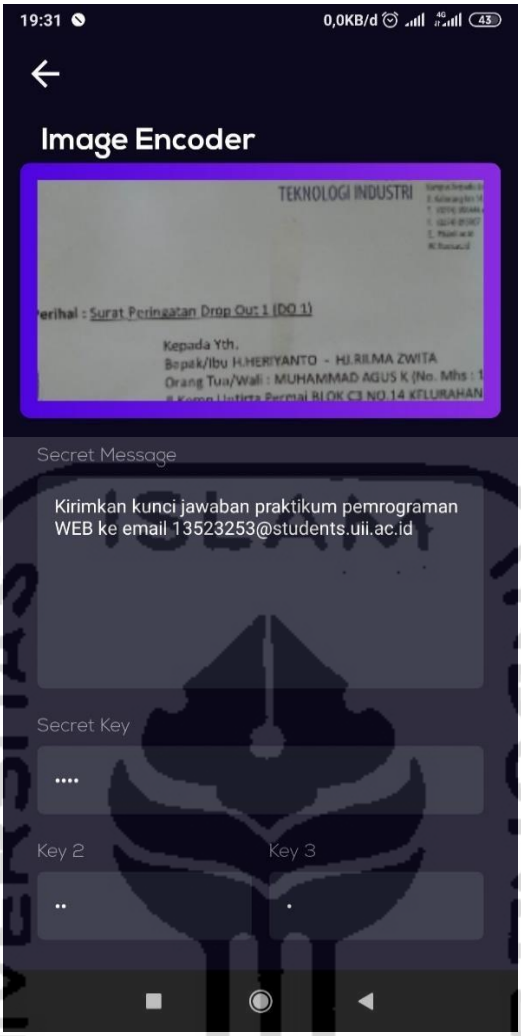
Gambar 4.4 *Pop-Up* pilih gambar

Pada Gambar 4.4 merupakan antarmuka untuk melakukan proses pencarian gambar, yang dimana terdapat 2 pilihan yang dapat digunakan untuk mencari gambar yaitu pilih gambar melalui kamera atau dari *gallery handphone*.

Jika yang digunakan melalui hak akses kamera, maka aplikasi akan mengakses (mengaktifkan) kamera yang ada di *handphone*. Jika yang digunakan melalui hak akses *gallery*, maka aplikasi akan mengakses *gallery handphone* untuk mencari gambar yang akan digunakan. Maka seperti ini jika gallery handphone digunakan

Gambar 4.5 *Pop-Up Gallery Handphone*

Gambar 4.5 merupakan proses antarmuka *gallery handphone* yang dapat diakses. 3 menu yang terdapat pada gambar tergantung aplikasi yang ada di perangkat *handphone*. Perangkat yang digunakan pada penelitian ini menggunakan *handphone Xiaomi Redmi Note* 5 yang dimana terdapat 3 aplikasi *gallery* yang digunakan yaitu aplikasi galeri, Foto, dan *File manager*.

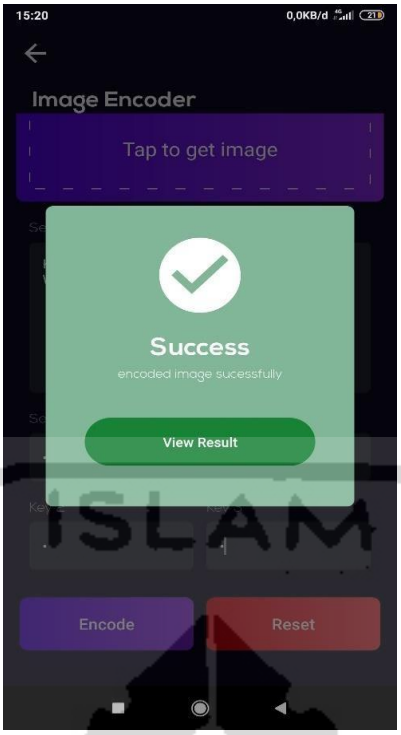


Gambar 4.6 Proses enkripsi dan *Embedding*

Gambar 4.6 merupakan antarmuka menu encode yang dimana jika semua aturan – aturannya sudah terpenuhi, maka selanjut dapat mengakses 2 tombol yang tersedia. 2 tombol tersebut yaitu " *Encode* " dan " *Reset* ".

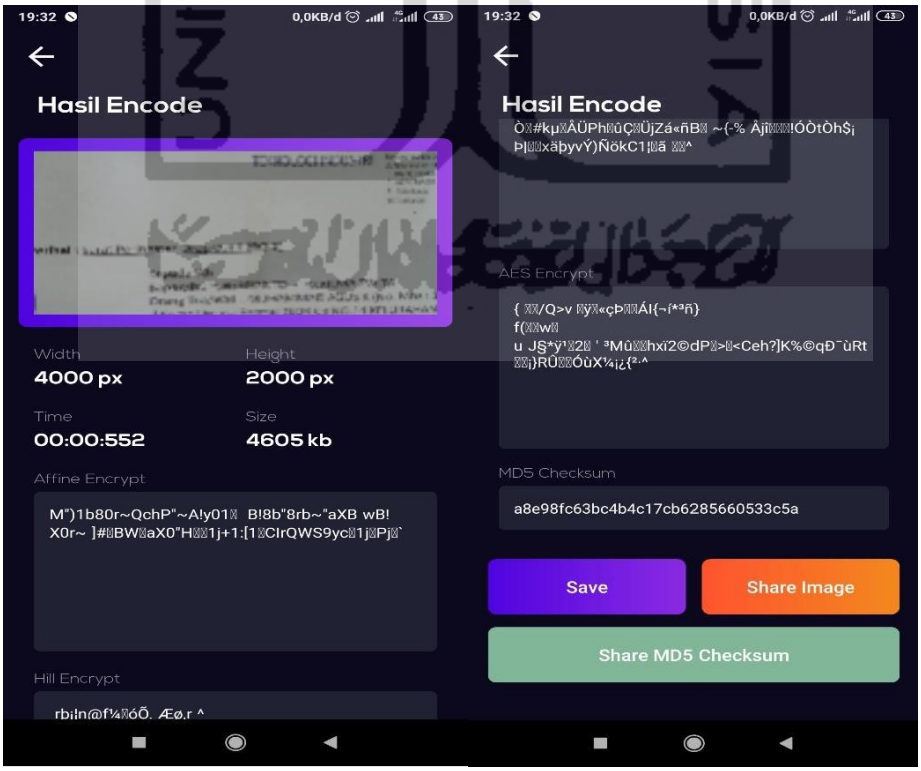
Tombol " *Reset* " digunakan pada saat kesalahan pada melakukan penginputan data, yang dimana dapat menghapus semua kesalahan input yang terjadi. Sedangkan tombol " *Encode* " digunakan untuk melakukan proses pengenkripsian pesan dan penyembunyian pesan terhadap objek gambar yang digunakan.

Disini menggunakan kunci " agus ", " 16 ", dan " 5 ", setelah itu melakukan *encode* maka akan menampilkan sebuah notifikasi untuk melihat hasil proses program yang dijalankan aplikasi, maka antarmuka notifikasi hasil proses berhasil melakukan *encode* seperti ini



Gambar 4.7 Notifikasi proses encode

Pada Gambar 4.7 merupakan antarmuka notifikasi keberhasilan melakukan proses *encode* yang dimana terdapat tombol " *View Result* " yang digunakan untuk melihat hasil.

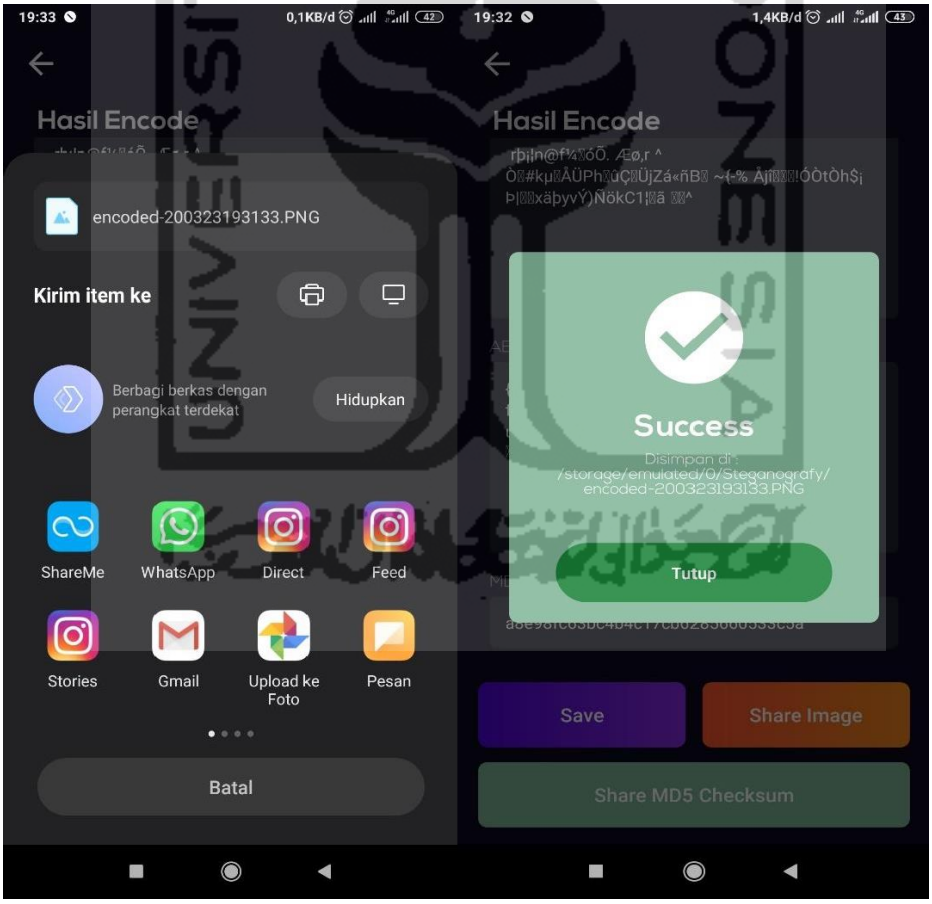


Gambar 4.8 Hasil proses Encode

Gambar 4.8 merupakan antarmuka hasil dari proses encode, yang dimana menampilkan kembali gambar yang digunakan. Diantarmuka hasil ini memberikan informasi ukuran pixel, waktu proses melakukan enkripsi dan *embedding*, besar ukuran gambar, informasi hasil dari metode yang digunakan seperti *affine cipher*, *hill cipher*, dan AES (*Advanced Encrypt Standar*), serta memberikan informasi nilai identic dari gambar yang sudah di *Embedding*.

Hasil proses encode ini terdapat 3 tombol yang digunakan yaitu " *Save* ", " *share image* ", dan " *Share MD5* ". untuk proses share dibuat 2, dikarenakan pada proses pengiriman melalui Bluetooth atau sejenisnya. Program mengalami terjadinya error, sehingga dibuat 2 proses.

Save disini untuk menyimpan hasil proses encode ke folder dengan nama folder " Steganografi", untuk kedua tombol *share* disini berfungsi mengirim gambar atau teks MD 5 ke aplikasi – aplikasi yang dapat digunakan.

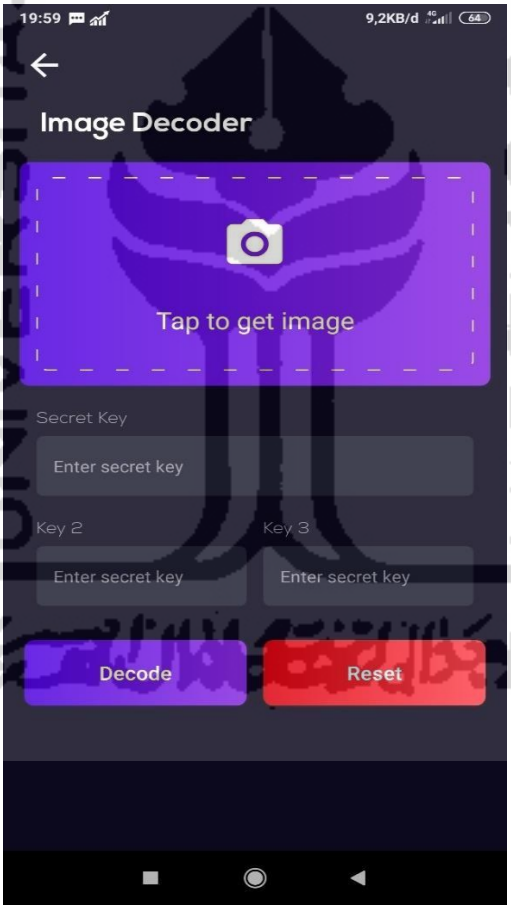


Gambar 4.9 Share atau Save

Seperti pada tampilan yang terdapat pada Gambar 4.9. Tombol " *share* " dapat mengirim langsung lewat email, media social, ataupun aplikasi pengiriman via Bluetooth, sedangkan untuk tombol " *save* " memberikan informasi lokasi penyimpanan hasil gambar yang sudah di *encode*.

4.2.2 Implementasi *Extracting* dan Dekrip (Decode)

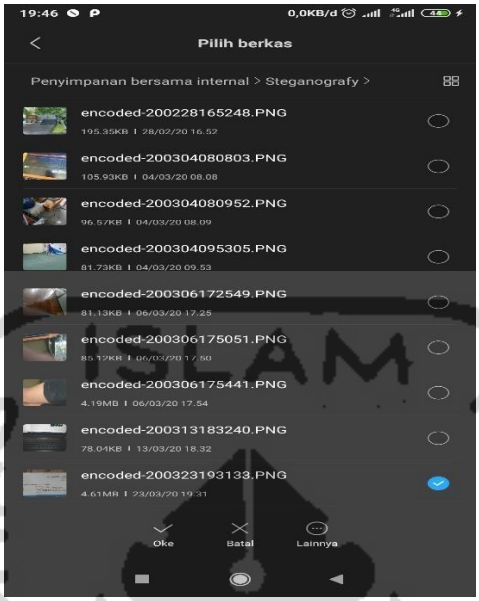
Proses Extracting dan Dekrip atau yang bisa di sebut *decode* digunakan untuk melakukan proses pelepasan pesan pada citra gambar dan mengembalikan pesan acak dapat dibaca kembali dengan aturan – aturan yang digunakan pada proses *encode*. Antarmuka *decode* dapat dilihat seperti ini jika pengguna memasuki " Menu *decode* ".



Gambar 4.10 Halaman Menu Decode

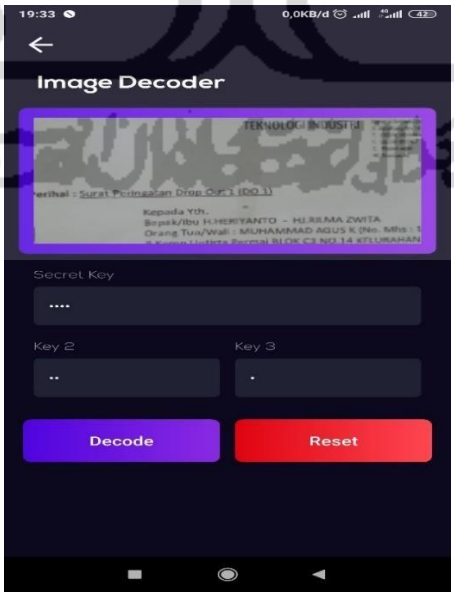
Gambar 4.10 merupakan antarmuka halaman menu *decode* yang dimana terdapat tombol mencari gambar stego-image yang ada didalam gallery atau file penyimpanan, inputan kunci dan terdapat 2 tombol yaitu " *Decode* " dan " *Reset* ". seperti tombol " *Reset* " yang terdapat di

menu encode, sama – sama mengulang inputan kembali. Sedangkan untuk tombol " *decode* " digunakan untuk melakukan pengambilan pesan kembali yang terdapat di stego-image.



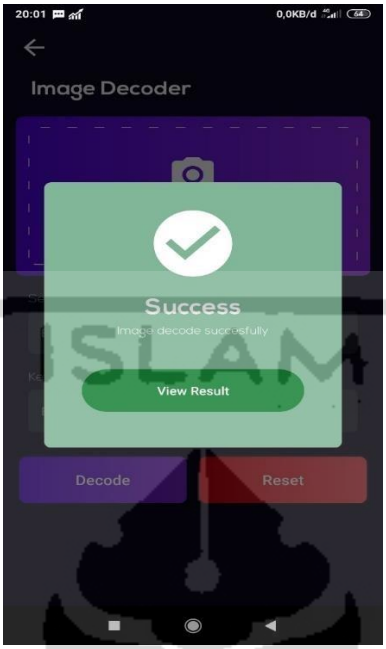
Gambar 4.11 Pemilihan gambar *stego-image*

Seperti Gambar 4.11 merupakan proses tampilan pengambilan gambar *stego-image* yang tersimpan pada folder " Steganografi " yang ada diberkas file manager. Setelah mendapatkan stego-image maka gambar akan ditampilkan kembali di menu encode, seperti ini.



Gambar 4.12 Antarmuka Menu Decode

Setelah inputan gambar dan kunci terpenihi seperti yang ditampilkan pada Gambar 4.12 maka, tombol decode dapat digunakan untuk melakukan pengambilan pesan.



Gambar 4.13 Notifikasi *Success* Decode

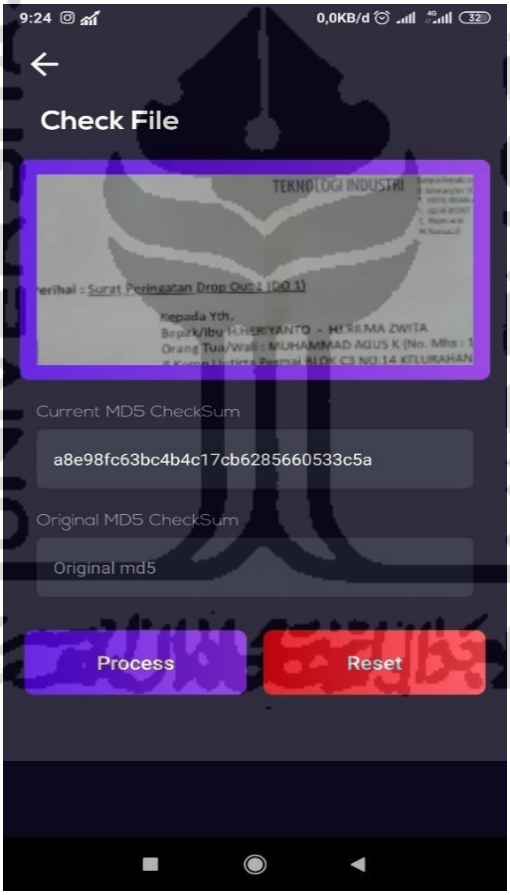
Seperti Gambar 4.13 merupakan antarmuka informasi yang memberitahukan bahwa pesan dapat dibaca kembali, yang dimana informasi tersebut harus menakan tombol " View Result " sehingga dapat mengetahui hasil proses Extracting dan Dekrip pada Stego-image.

Gambar 4.14 Hasil proses Decode

Gambar 4.14 merupakan tampilan antarmuka hasil proses *decode* yang dimana jika " *result view* " ditekan maka menghasilkan informasi dari semua proses decode. Informasi decode juga memberikan informasi ukuran pixel, waktu proses *Extracting* dan dekrip, ukuran gambar dan Informasi pesan yang tersebunyi yang ada didalam *stego-image*.

4.2.3 Implementasi File Check

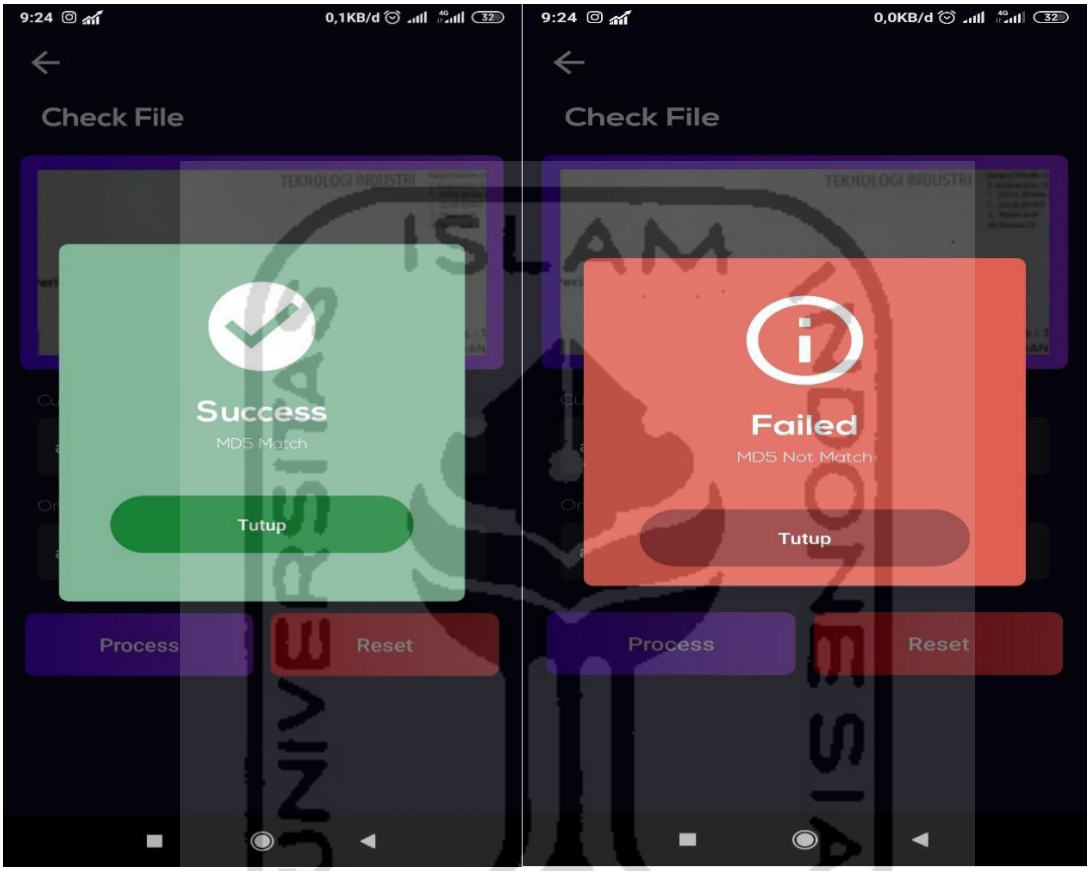
Proses Chec File ini digunakan untuk melakukan proses pengambilan nilai identik dari stego image yang diterima dengan mencocokan nilai identik yang diberikan oleh si pemberi untuk mengetahui terjadinya perubahan dalam file tersebut. Berikut implementasi antarmuka File check berfungsi dengan semestinya.



Gambar 4.15 Mengambil nilai Identik Stego Image

Gambar 4.15 merupakan tampilan dari proses aplikasi untuk melakukan proses pengecekan keaslian file yang dikirim oleh pemberi. Pada prosesnya, pengguna wajib memasukan gambar untuk mendapatkan original MD5 dari stego image tersebut, yang nantinya aplikasi akan menampilkan MD 5 di kolom textView dengan nama *Current MD5 Checksum*. Di kolom

tersebut merupakan hasil pengambilan dari file stego image, setelah itu pengguna memasukan original MD 5 yang diberi oleh pengirim. Ketika pengguna menekan tombol *Process*, maka program berjalan untuk memberikan hasil kecocokan. Berikut gambar hasil *process* jika terjadi kecocokan maupun tidak cocok



Gambar 4.16 Informasi hasil kecocokan File

Gambar 4.16 merupakan tampilan dari proses Check File yang memberikan informasi kecocokan file yang dikirim. Ketika MD 5 dari stego image dengan input MD 5 original cocok, maka program menampilkan notifikasi "*Success*". Jika MD5 tidak cocok maka program memberikan informasi "*Failed*".

4.3 Source Code Implementasi Program

Berdasarkan hasil rancangan teori yang digambarkan pada alur flowchart pada bab sebelumnya, tahapan ini merupakan implementasi kode program dari proses algoritma – algoritma yang digunakan dengan mengubah ke dalam bentuk bahasa pemgorgraman java.

Konsep penjalasan kode program ini dengan memberikan informasi fungsi kode baris yang digunakan. Maka berikut kode program dari teori kriptografi dan steganografi yang dibuat

4.3.1 Implementasi Source Code Enkripsi dan Embedding

Berdasarkan alur yang dibuat pada Gambar 3.1 untuk mendapatkan sebuah *stego image*, harus melalui tahapan proses enkripsi terlebih dahulu. Dimana dalam prosesnya memalui tahap proses enkripsi Affine Cipher, Hill Cipher, Caesar Cipher dan AES.

Setelah proses enkripsi selesai, program menjalankan proses embedding yang digunakan untuk menyembunyikan pesan rahasia tersebut ke dalam objek gambar untuk mendapatkan hasil *Stego image*. Hasil *stego image* tesebut dienkripsi lagi untuk mendapatkan sebuah nilai hash pada proses pengiriman nanti. Berikut kode program aplikasi menjalankan algoritma tersebut dalam bentuk kode program

4.3.1.1 Source Code Affine Cipher Enkripsi

Berdasarkan alur dari Gambar 3.4 dalam menjalankan proses enkripsi, terdapat tahapan – tahapan yang harus dilalui yaitu pengecekan kunci dan proses perhitungan enkripsi dengan rumus dari persamaan 2.6 . Berikut langkah awal kode program *Affine Cipher* dalam melakukan proses pengecekan kunci.

```
1   int modulus = 127;
2   public int inversModular(int kunci_a) {
3       for (int i = 0; i<modulus; i++){
4           if ((kunci_a*i)%modulus == 1){
5               return i;
6           }
7        }
9     return 0;
10  }
```

Gambar 4.17 Kode program inversModular/pengecekan kunci

Gambar 4.17 merupakan kode program yang digunakan pada saat proses pengecekan kunci. Fungsi ini digunakan untuk mengetahui kunci tersebut memiliki nilai invers atau tidak. Jika memiliki invers yang memenuhi invers modular, maka kunci dapat digunakan untuk proses enkripsi dan jika tidak maka kunci tidak dapat digunakan.

kode program tersebut juga digunakan untuk mencegah terjadinya perbedaan antara kunci enkripsi dengan kunci dekripsi. Sehingga pengguna tidak akan melakukan kesalahan, jika kunci yang digunakan pada proses enkripsi dapat digunakan juga pada proses dekripsi.

Setelah itu, alur berikut yaitu melakukan proses enkripsi, Dimana dalam 1 kode program memiliki fungsi berdasarkan alur yang dibuat. Berikut kode program *Affine Cipher* melakukan proses enkripsi.

```
1  private String ProsesEnkrip(String pesan, int kunci_a, int kunci_b) {
2      StringBuilder str = new StringBuilder();
3      String hasilEnkrip="";
4      kunciC = kunci_b + 15;
5      kunciD = kunci_b + kunciC;
6
7      for (int i = 0; i < pesan.length(); i++) {
8          kunciE = kunciC + i;
9
10         if ((i - 1)%5 == 0 ) {
11             hasilRotasi = kunci_a*((int)pesan.charAt(i)) + kunci_b;
12             modular = floorMod(hasilRotasi, modulus);
13             char a1 = (char) ((char) modular);
14             hasilEnkrip = String.valueOf(a1);
15             str.append(hasilEnkrip);
16         }
17         if ((i - 2)%5 == 0) {
18             hasilRotasi = kunci_a*((int)pesan.charAt(i)) - kunci_b;
19             modular = floorMod(hasilRotasi, modulus);
20             char a1 = (char) ((char) modular );
21             hasilEnkrip = String.valueOf(a1);
22             str.append(hasilEnkrip);
23         }
24         if ((i - 3)%5 == 0) {
25             hasilRotasi = kunci_a*((int)pesan.charAt(i)) + kunciC;
26             modular = floorMod(hasilRotasi, modulus);
27             char a1 = (char) ((char) modular );
28             hasilEnkrip = String.valueOf(a1);
29             str.append(hasilEnkrip);
30         }
31         if ((i - 4)%5 == 0) {
32             hasilRotasi = kunci_a*((int)pesan.charAt(i)) - kunciD;
33             modular = floorMod(hasilRotasi, modulus);
34             char a1 = (char) ((char) modular );
35             hasilEnkrip = String.valueOf(a1);
36             str.append(hasilEnkrip);
37         }
38         if ((i - 5)%5 == 0) {
39             hasilRotasi = kunci_a*((int)pesan.charAt(i)) + kunciE;
40             modular = floorMod(hasilRotasi, modulus);
41             char a1 = (char) ((char) modular );
42             hasilEnkrip = String.valueOf(a1);
43             str.append(hasilEnkrip);
44         }
45     }
46     return str.toString();
47 }
```

Gambar 4.18 Kode program enkripsi *Affine Cipher*

Gambar 4.18 merupakan kode program *Affine Cipher* dalam menjalankan proses enkripsi berdasarkan pada alur Gambar 3.4. Setelah proses cek kunci terpenuhi. kode baris program 1 – 47 menjalankan proses sisanya. Berikut penjelasan baris kode pada Gambar 4.18.

Kode baris program 7 berfungsi melakukan perulangan berdasarkan perhitungan jumlah karakter pesan. Kode baris 10,17,24,31 dan 38 merupakan proses pembagian kelompok berdasarkan jumlah karakter yang didapat dari kode baris program 7.

kode baris program 11 – 15, 18 – 22, 25 – 29, 32 – 36, dan 39 - 43 berfungsi dalam melakukan proses perhitungan dengan rumus persamaan 2.6, yang dimana pada proses perhitungannya dibagi – bagi dengan kunci pergeseran yang berbeda – beda, yang dapat dilihat pada kode baris program 11, 18, 25, 32 dan 39 untuk mendapat sebuah karakter baru yang menghasilkan sebuah pesan *Chipertext* yang akan diproses pada algoritma *Hill Cipher*

4.3.1.2 Source Code Hill Cipher Enkripsi

Berdasarkan alur dari Gambar 3.6 dalam melakukan enkripsi sebuah pesan, terdapat proses yang harus terpenuhi yaitu proses dalam melakukan pengecekan karakter, proses pembagian karakter, proses perubahan karakter ke ASCII decimal, serta proses enkripsi. Berikut kode program *Hill Cipher* dalam melakukan pengecekan karakter dan pembagian karakter.

```
1   static String pisahkanTeks(String teks){
2       String teksnya = teks;
3       if (teksnya.length() % 4 == 0){
4           teksnya = teks;
5       }else if (teksnya.length() % 4 == 1){
6           teksnya = teks + "." + "." + ".";
7       }else if (teksnya.length() % 4 == 2){
8           teksnya = teks + "." + ".";
9       }else{
10          teksnya = teks + "." ;
11      }
12      teks2karakter = new String[teksnya.length() / 4];
13      for (int i = 0; i < teks2karakter.length; i++){
14          teks2karakter[i] = teksnya.substring(i * 4, i * 4 + 4);
15      }
16      return teksnya;
17  }
```

Gambar 4.19 Kode program pengecekan dan pembagian karakter

Gambar 4.19 merupakan kode program yang digunakan untuk melakukan pengecekan dan pembagian karakter ke dalam sebuah blok array matriks. Fungsi pembagian *chipertext* ke dalam blok array matriks ini digunakan untuk proses perhitungan pada algoritma *Hill Cipher* yang dirumuskan pada persamaan 2.12. Berikut fungsi kode baris program yang terdapat pada Gambar 4.19.

Kode baris program 3 – 11 berfungsi melakukan pengecekan jumlah karakter, yang dimana jika karakter tidak berkelipatan 4, maka ada penambahan karakter tergantung

kekurangan dari karakter tersebut. Untuk kode baris program 12 – 15 berfungsi melakukan pembagian blok – blok array yang berisikan 4 karakter, yang berguna untuk proses perhitungan matriks kunci pada enkripsi maupun dekripsi

Setelah proses pengecekan karakter dan pembagian karakter dalam blok – blok matriks terpenuhi, proses selanjutnya yaitu melakukan proses enkripsi dengan algoritma *Hill Cipher*. Berikut kode program proses algoritma Hill Cipher dalam melakukan proses enkripsi.

```
1  static String[][] perhitunganKunci(String[][] angka, int[][] matrix){
2      int B1K1 = matrix[0][0], B1K2 = matrix[0][1], B1K3 = matrix[0][2],
3          B1K4 = matrix[0][3];
4      int B2K1 = matrix[1][0], B2K2 = matrix[1][1], B2K3 = matrix[1][2]
5          B2K4 = matrix[1][3];
6      int B3K1 = matrix[2][0], B3K2 = matrix[2][1], B3K3 = matrix[2][2]
7          B3K4 = matrix[2][3];
8      int B4K1 = matrix[3][0], B4K2 = matrix[3][1], B4K3 = matrix[3][2]
9          B4K4 = matrix[3][3];
10     hasilHitungKunci = new String[angka.length][4];
11     for (int n = 0; n < angka.length; n++) {
12         int konvert = Integer.parseInt(angka[n][0]);
13         int konvert1 = Integer.parseInt(angka[n][1]);
14         int konvert2 = Integer.parseInt(angka[n][2]);
15         int konvert3 = Integer.parseInt(angka[n][3]);
16         int hasil = (B1K1 * konvert) + (B1K2 * konvert1) + (B1K3 * konvert2)
17                   + (B1K4 * konvert3);
18         int hasil1 = (B2K1 * konvert) + (B2K2 * konvert1) + (B2K3 * konvert2)
19                    + (B2K4 * konvert3);
20         int hasil2 = (B3K1 * konvert) + (B3K2 * konvert1) + (B3K3 * konvert2)
21                    + (B3K4 * konvert3);
22
23         int hasil3 = (B4K1 * konvert) + (B4K2 * konvert1) + (B4K3 * konvert2)
24                    + (B4K4 * konvert3);
25
26         hasil = hasil % modulo;
27         hasil1 = hasil1 % modulo;
28         hasil2 = hasil2 % modulo;
29         hasil3 = hasil3 % modulo;
30
31         if (hasilHitungKunci[n][0] == null){
32             hasilHitungKunci[n][0] = String.valueOf(hasil);
33         }if(hasilHitungKunci[n][1] == null){
34             hasilHitungKunci[n][1] = String.valueOf(hasil1);
35         }if(hasilHitungKunci[n][2] == null){
36             hasilHitungKunci[n][2] = String.valueOf(hasil2);
37         }if(hasilHitungKunci[n][3] == null){
38             hasilHitungKunci[n][3] = String.valueOf(hasil3);
39         }
40     }
41      return hasilHitungKunci;
42 }
```

Gambar 4.20 Kode program enkripsi dan dekripsi *Hill Cipher*

Gambar 4.20 merupakan kode program algoritma Hill Cipher dalam menjalankan proses enkripsi maupun dekripsi untuk mendapatkan sebuah *plaintext* maupun *chipertext*. Kode program tersebut memiliki beberapa fungsi diantara lain yaitu mengubah karakter menjadi

kode ASCII decimal, serta perhitungan perkalian antara kode ASCII decimal dengan Matriks kunci. Berikut fungsi kode baris program yang terdapat pada Gambar 4.20.

Kode baris program 12 – 15 berfungsi mengubah *String array* karakter pesan menjadi kode ASCII decimal, yang dimana kode ASCII decimal tersebut akan di kalikan dengan kunci Matriks. Kode baris program 16 – 24 merupakan proses perhitungan antara kode ASCII decimal dengan kunci Matriks yang menghasilkan sebuah *ciphertext* yang akan digunakan pada proses enkripsi *Advanced Encrypt Standard*.

4.3.1.3 Source Code Caesar Cipher

Sebelum melakukan proses enkripsi AES, berdasarkan alur pada Gambar 3.1 Gambar 3.1 *Flowchart* enkripsi dan *Embedding*, algoritma *Caesar Cipher* digunakan untuk mengenkripsi sebuah kunci masukan yang akan digunakan pada algoritma *Advanced Encrypt Standard*.

Pada proses pengenkripsiannya, terdapat proses – proses yang harus terpenuhi yang dimana proses tersebut berdasarkan alur pada Gambar 3.8. berikut kode program *Caesar Cipher* mengenkripsi kunci yang akan digunakan pada algoritma AES.

```
1   static String prosesCekKunci(String kunci){
2       StringBuilder str = new StringBuilder(kunci);
3       if (kunci.length()<=16) {
4           for (int i = 0; i < (16 - kunci.length()); i++)
5           {
6               str.append(".");
7           }
8       }else{
9           str = new StringBuilder(str.substring(0, 15));
10      }
11      return str.toString();
12  }
```

Gambar 4.21 Kode program cek kunci

Gambar 4.21 merupakan kode program yang berfungsi untuk melakukan pengecekan jumlah karakter kunci sebelum dilakukan proses enkripsi dengan algoritma *Caesar Cipher*. Pengecekan ini berguna untuk memenuhkan sebuah *state* ekspansi kunci di algoritma AES-128 bit yang memiliki total 16 karakter. Berikut fungsi dari kode program tersebut

Proses pengecekannya ditunjukan 3 – 10. Dimana jika karakter kunci AES kurang dari 16 maka akan ada penambahan karakter, sampai total karakter kunci AES tersebut mencapai 16 karakter. Jika karakter memiliki 16 karakter, maka proses langsung berjalan dengan jumlah karakter input.

Setelah pengecekan terpenuhi, berikut kode program algoritma *Caesar Cipher* melakukan proses enkripsi kunci yang akan digunakan pada Algoritma AES-128 bit .

```
1   static String prosesCaesarCipher(String kunciEncrypt, int kunciB){
2       StringBuilder str = new StringBuilder();
3       String hasilEnkrip="";
4
5       for (int i = 0; i < kunciEncrypt.length(); i++) {
6           if ((i-1)%3 == 0) {
7               hasilRotasi = (int)kunciEncrypt.charAt(i) - 32 - kunciB;
8               modular = floorMod(hasilRotasi, modulus);
9               char a1 = (char) ((char) modular + 32);
10              hasilEnkrip = String.valueOf(a1);
11              str.append(hasilEnkrip);
12          }if ((i-2)%3 == 0) {
13              hasilRotasi = (int)kunciEncrypt.charAt(i) - 32 + kunciB;
14              modular = floorMod(hasilRotasi, modulus);
15              char a1 = (char) ((char) modular + 32);
16              hasilEnkrip = String.valueOf(a1);
17              str.append(hasilEnkrip);
18          }
19          if ((i-3)%3 == 0) {
20              hasilRotasi = (int)kunciEncrypt.charAt(i) - 32 - kunciC;
21              modular = floorMod(hasilRotasi, modulus);
22              char a1 = (char) ((char) modular + 32);
23              hasilEnkrip = String.valueOf(a1);
24              str.append(hasilEnkrip);
25          }
26      }
27      return str.toString();
28  }
```

Gambar 4.22 Kode progam enkripsi *caesar cipher*

Gambar 4.22 merupakan kode program *Caesar cipher* melakukan pengenkripsian kunci yang akan digunakan pada algoritma AES-128 bit. Kode baris program 5 melakukan perulangan berdasar jumlah karakter kunci AES, yang berfungsi untuk mengenkripsi semua karakter kunci tersebut.

Kode baris program 6, 12, dan 19 berfungsi membagikan proses dengan perhitungan kunci pergeseran yang berbeda tergantung urutan dari karakter kunci. Kode baris program 7 – 11, 13 – 17, dan 20 – 24 merupakan proses perhitungan untuk mendapatkan *Chipertext* kunci yang akan digunakan pada algoritma *Advanced Encrypt Standard*.

4.3.1.4 Source Code Enkripsi AES-128 bit

Berdasarkan alur pada Gambar 3.9 dalam melakukan proses enkripsi sebuah pesan, terdapat proses yang harus terpenuhi, yaitu masukan sebuah kunci hasil enkripsi dari algoritma *Caesar Cipher* yang didapat pada dari Gambar 4.22 dan sebuah pengecekan jumlah karakter pesan *ciphertext* dari Gambar 4.20.

Dikarenakan proses pengecekan jumlah karakter kunci sudah dilakukan, berikut kode program pengecekan jumlah karakter pesan *chipertext* sebelum proses enkripsi algoritma AES-128 bit.

```
static String prosesCekPesan(String pesan){
    String pesan_n = pesan;
    StringBuilder str = new StringBuilder();
    if (pesan_n.length()% 16 == 1) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + "."
+ "." + "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 2) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + "."
+ "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 3) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + "."
+ "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 4) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + "."
+ "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 5) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + "."
+ "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 6) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + "."
+ ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 7) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 8) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 9) {
        pesan_n = pesan + "." + "." + "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 10) {
        pesan_n = pesan + "." + "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 11) {
        pesan_n = pesan + "." + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 12) {
        pesan_n = pesan + "." + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 13) {
        pesan_n = pesan + "." + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 14) {
        pesan_n = pesan + "." + ".";
        str.append(pesan_n);
    }else if (pesan_n.length()% 16 == 15) {
        pesan_n = pesan + ".";
        str.append(pesan_n);
    }else{
        pesan_n = pesan;
        str.append(pesan_n);
    }
    return str.toString();
```

```
60  }
```

Gambar 4.23 Kode program cek jumlah karakter pesan

Gambar 4.23 merupakan kode program yang ada di algoritma AES untuk melakukan proses pengecekan jumlah karakter pesan *ciphertext*, proses pengecekan karakter ini, berguna untuk mengetahui keseluruhan karakter *ciphertext* memiliki kelipatan 16 atau tidak. Jika tidak, maka ada penambahan karakter sampai kelipatan 16, dikarenakan pada proses enkripsi AES terdapat sebuah *state* yang memiliki blok – blok penampung sebanyak 16 karakter.

Setelah pengecekan jumlah karakter pesan memiliki kelipatan 16 karakter, berikut kode program untuk mendapatkan kunci ekspansi setiap putaran sebelum melakukan proses enkripsi. berikut kode program ekspansi kunci AES 128 bit.

```
1   public void setKeys(byte[] key) {
2       final int BLOCK_SIZE = 16;
3       final int BC = BLOCK_SIZE / 4;
4       final int Klen = key.length;
5       final int Nk = Klen / 4;
6       numRounds = 10;
7       int i, j, r;
8       final int ROUND_KEY_COUNT = (numRounds + 1) * BC;
9
10      byte[] w0 = new byte[ROUND_KEY_COUNT];
11      byte[] w1 = new byte[ROUND_KEY_COUNT];
12      byte[] w2 = new byte[ROUND_KEY_COUNT];
13      byte[] w3 = new byte[ROUND_KEY_COUNT];
14
15      Ke = new byte[numRounds + 1][BLOCK_SIZE];
16      Kd = new byte[numRounds + 1][BLOCK_SIZE];
17
18      for (i=0, j=0; i < Nk; i++) {
19          w0[i] = key[j++];
20          w1[i] = key[j++];
21          w2[i] = key[j++];
22          w3[i] = key[j++];
23      }
24
25      byte t0, t1, t2, t3, old0;
26      for (i = Nk; i < ROUND_KEY_COUNT; i++) {
27          t0 = w0[i-1];
28          t1 = w1[i-1];
29          t2 = w2[i-1];
30          t3 = w3[i-1];
31          if (i % Nk == 0) {
32              old0 = t0;
33              t0 = (byte)(S_box[t1 & 0xFF] ^ rcon[i/Nk]);
34              t1 = (byte)(S_box[t2 & 0xFF]);
35              t2 = (byte)(S_box[t3 & 0xFF]);
36              t3 = (byte)(S_box[old0 & 0xFF]);
37          }
38          else if ((Nk > 6) && (i % Nk == 4)) {
39              t0 = S_box[t0 & 0xFF];
40              t1 = S_box[t1 & 0xFF];
41              t2 = S_box[t2 & 0xFF];
42              t3 = S_box[t3 & 0xFF];
43          }
```

```
44          w0[i] = (byte)(w0[i-Nk] ^ t0);
45          w1[i] = (byte)(w1[i-Nk] ^ t1);
46          w2[i] = (byte)(w2[i-Nk] ^ t2);
47          w3[i] = (byte)(w3[i-Nk] ^ t3);
48      }
49
50      for (r = 0, i = 0; r < numRounds + 1; r++) {
51          for (j = 0; j < BC; j++) {
52              Ke[r][4*j] = w0[i];
53              Ke[r][4*j+1] = w1[i];
54              Ke[r][4*j+2] = w2[i];
55              Ke[r][4*j+3] = w3[i];
56              Kd[numRounds - r][4*j] = w0[i];
57              Kd[numRounds - r][4*j+1] = w1[i];
58              Kd[numRounds - r][4*j+2] = w2[i];
59              Kd[numRounds - r][4*j+3] = w3[i];
60              i++;
61          }
62      }
63  }
```

Gambar 4.24 Kode program ekspansi kunci AES-128 bit

Gambar 4.24 merupakan algoritma yang berfungsi untuk mendapatkan kunci putaran (*key round*) yang digunakan untuk enkripsi dan dekripsi, kunci yang dimasukan merupakan kunci *chipertext* hasil dari enkripsi algoritma *Caesar Cipher*. Seperti teori yang dijelaskan dalam proses ekspansi kunci AES-128 bit memiliki 44 Word untuk 10 putaran. Kode baris program 8 digunakan untuk mencari proses tersebut.

Untuk kode program 10 – 13 berfungsi mendefinisikan suatu temporary nilai *byte*. Sedangkan untuk kode baris program 15 – 16 mendefinisikan nilai byte yang didapat pada putaran tertentu untuk digunakan enkripsi dan dekripsi. kode baris program 18 – 23 berfungsi memasukan nilai *byte input* ke dalam sebuah *state*.

Kode baris program 26 – 48 merupakan proses perhitungan untuk mendapatkan kunci ekspansi yang akan digunakan. Kode baris program 27 – 30 merupakan proses RotWord, yang berfungsi menggeserkan setiap *byte* yang ada didalam *state*. Untuk kode baris program 31 – 43 berfungsi untuk proses *SubBytes* dan *RCon* yang dimana terdapat kondisi. Untuk kondisi menghasilkan nilai 0 maka kode baris program 33 – 35 melakukan proses *RCon*, jika kondis selain 0 maka kode baris program 39 – 42 melakukan proses *SubBytes*

Kode baris program 44 – 47 berfungsi melakukan *XOR* kunci dengan kunci sebelumnya. Kode baris program 50 – 62 berfungsi menyimpan semua proses ke dalam penyimpanan sementara untuk digunakan pada proses enkripsi dan dekripsi. 52 – 55 digunakan untuk proses enkripsi, sedangkan 56 – 59 digunakan untuk proses dekripsi.

Setelah itu, program akan menjalankan proses enkripsi berdasarkan alur yang sudah dibuat pada Gambar 3.9. Berikut kode program enkripsi AES 128 bit.

```
1  public byte[] encrypt(byte[] plain) {
2      byte [] a = new byte[BLOCK_SIZE];
3      byte [] ta = new byte[BLOCK_SIZE];
4      byte [] Ker;
5      int    i, j, k, row, col;
6      Ker = Ke[0];
7
8      for (i = 0; i < BLOCK_SIZE; i++){
9          a[i] = (byte)(plain[i] ^ Ker[i]);
10     }
11
12     for (int r = 1; r < numRounds; r++) {
13         Ker = Ke[r];
14         for (i = 0; i < BLOCK_SIZE; i++){
15             ta[i] = S_box[a[i] & 0xFF];
16         }
17         for (i = 0; i < BLOCK_SIZE; i++) {
18             row = i % COL_SIZE;
19             k = (i + (row_shift[row] * COL_SIZE)) % BLOCK_SIZE;
20             a[i] = ta[k];
21         }
22         for (col = 0; col < NUM_COLS; col++) {
23             i = col * COL_SIZE;
24             ta[i]   = (byte)(mul(2,a[i]) ^ mul(3,a[i+1]) ^ a[i+2] ^ a[i+3]);
25             ta[i+1] = (byte)(a[i] ^ mul(2,a[i+1]) ^ mul(3,a[i+2]) ^ a[i+3]);
26             ta[i+2] = (byte)(a[i] ^ a[i+1] ^ mul(2,a[i+2]) ^ mul(3,a[i+3]));
27             ta[i+3] = (byte)(mul(3,a[i]) ^ a[i+1] ^ a[i+2] ^ mul(2,a[i+3]));
28         }
29         for (i = 0; i < BLOCK_SIZE; i++){
30            a[i] = (byte)(ta[i] ^ Ker[i]);
31         }
32     }
33
34     Ker = Ke[numRounds];
35     for (i = 0; i < BLOCK_SIZE; i++){
36         a[i] = S_box[a[i] & 0xFF];
37     }
38     for (i = 0; i < BLOCK_SIZE; i++) {
39         row = i % COL_SIZE;
40         k = (i + (row_shift[row] * COL_SIZE)) % BLOCK_SIZE;
41         ta[i] = a[k];
42     }
43     for (i = 0; i < BLOCK_SIZE; i++){
44         a[i] = (byte)(ta[i] ^ Ker[i]);
45     }
46     return (a);
47 }
```

Gambar 4.25 Kode program enkripsi AES-128 bit

Gambar 4.25 merupakan kode program dari algoritma AES dalam proses pengenkripsiannya, yang dimana dalam kode program tersebut memiliki tahapan – tahapan proses yang dibutuhkan pada pengekripsian. Berikut tahapan – tahapan proses enkripsi AES.

Kode baris program 8 – 10 merupakan proses *AddRoundKey* untuk mendapatkan *state* baru yang akan digunakan pada putaran pertama, proses ini melakukan XOR antara pesan *ciphertext* dengan kunci *ciphertext* yang didapat pada Gambar 4.22.

Kode baris program 12 – 32 merupakan Round/putaran algoritma AES, yang dimana terdapat tahapan – tahapan proses *SubBytes, ShiftRows, MixColumns,* dan *AddRoundKey*. Kode baris program 14 – 16 merupakan proses *SubBytes* antara *state* dengan tabel S-Box. Kode baris program 17 – 21 merupakan proses *ShiftRows*, yang berfungsi menggeserkan pada baris *state*. Kode baris program 22 – 28 merupakan proses *MixColumns*, yang berfungsi untuk melakukan *XOR* antara *state* yang dihasilkan *ShiftRows* dengan *state* konstan (yang sudah ditentukan). Untuk kode baris program 29 – 31 merupakan proses *AddRoundKey* yang terdapat didalam putaran yang berfungsi mendapatkan *state* baru dari hasil XOR antara *state* hasil *MixColumns* dengan kunci ekspansi yang didapat pada Gambar 4.24.

Kode baris program 35 – 45 merupakan tahapan akhir proses AES-128 bit melakukan putaran terakhir, yang dimana terdapat tahapan proses *SubBytes, ShiftRows*, dan *AddRoundKey*. Kode baris program 35 – 37 merupakan proses *SubBytes,* yang berfungsi untuk mencari *Hex* baru antara *State* dari putaran yang didapat dengan Tabel S-Box. Kode baris program 38 – 42 merupakan proses *ShiftRows* di putaran terakhir. Kode baris program 43 – 45 merupakan proses terakhir *addRoundKey* untuk mendapatkan *chipertext* terakhir dari proses enkripsi.

4.3.1.5 Source code Embedding Steganografi

Berdasarkan alur pada Gambar 3.14 dalam melakukan proses *embedding* sebuah *ciphertext* yang didapat dari algoritma AES-128. Terdapat proses – proses yang harus dilakukan, berikut kode program steganografi dalam melakukan proses *Embedding*.

```
1  public static List<Bitmap> encodeMessage(List<Bitmap> splitted_images,
2                                            String          encrypted_message,
3  ProgressHandler progressHandler) {
4
5      List<Bitmap> result = new ArrayList<>(splitted_images.size());
6
7      encrypted_message = encrypted_message + END_MESSAGE_COSTANT;
8      encrypted_message = START_MESSAGE_COSTANT + encrypted_message;
9
10
11     byte[] byte_encrypted_message = encrypted_message.
12                                     getBytes(Charset.forName("ISO-8859-1"));
13
14     MessageEncodingStatus message = new MessageEncodingStatus(
15                                    byte_encrypted_message, encrypted_message);
16
17     if (progressHandler != null) {
18     progressHandler.setTotal(encrypted_message.getBytes(Charset.forName("ISO-
19     8859-1")).length);
20     }
21
22     Log.i(TAG, "Message length " + byte_encrypted_message.length);
```

```
23
24      for (Bitmap bitmap : splitted_images) {
25        if (!message.isMessageEncoded()) {
26            int width = bitmap.getWidth();
27            int height = bitmap.getHeight();
28
29            int[] oneD = new int[width * height];
30            bitmap.getPixels(oneD, 0, width, 0, 0, width, height);
31
32            int density = bitmap.getDensity();
33
34            byte[] encodedImage = encodeMessage(oneD, width, height,
35                                                message, progressHandler);
36            int[] oneDMod = Utility.byteArrayToIntArray(encodedImage);
37
38            Bitmap encoded_Bitmap = Bitmap.createBitmap(width, height,
39                                    Bitmap.Config.ARGB_8888);
40            encoded_Bitmap.setDensity(density);
41
42            int masterIndex = 0;
43
44            for (int j = 0; j < height; j++){
45               for (int i = 0; i < width; i++) {
46                   encoded_Bitmap.setPixel(i, j, Color.argb(0xFF,
47                                oneDMod[masterIndex] >> 16 & 0xFF,
48                                oneDMod[masterIndex] >> 8 & 0xFF,
49                                oneDMod[masterIndex++] & 0xFF));
50                }
51            }
52            result.add(encoded_Bitmap);
53       } else {
54         result.add(bitmap.copy(bitmap.getConfig(), false));
55       }
56     }
57      return result;
58  }
```

Gambar 4.26 Kode program *embedding* gambar dan pesan

Gambar 4.26 merupakan kode program steganografi melakukan proses *embedding.* Kode program memiliki fungsi untuk mendapatkan ukuran pixel dari gambar yang dipilih dan proses pengembalian gambar menjadi sebuah *stego image*. Berikut fungsi dari setiap baris kode yang terdapat pada Gambar 4.26.

Kode baris program 5 berfungsi membuat variabel bertipe *arrayList* berfungsi menyimpan data dengan jumlah besar yang berguna untuk hasil akhir dari proses ini. Kode baris program 7 dan 8 berfungsi penanda karakter awal dan diakhir pesan untuk proses decode. Kode baris program 11 digunakan untuk mengambil data *byte* dari pesan yang akan disisipkan, kode baris program 14 berfungsi mengambil nilai byte dari proses kode baris 11 dan disimpan pada konstruktor untuk digunakan pada proses selanjutnya. Kode baris baris 17 – 20 berfungsi menambahkan durasi proses dialog.

Kode baris program 24 – 56 merupakan kode program yang berfungsi melakukan perulangan dari karakter pesan yang akan dimasukan ke dalam gambar. Sedangkan untuk kode

baris 25 berfungsi pengecekan proses *embedding* selesai atau belum, jika belum akan melanjutkan proses yang dimana proses awalnya mengambil nilai lebar dan tinggi gambar yang ada dikode program 26 dan 27, setelah itu kode baris program 29 dan 30 membuat variabel baru dan mendifinisikan pixel array dari proses 26 dan 27. Kode baris program 31 berfungsi untuk mendapat density image/gambar yang dihasilkan dari split kode baris 24.

Kode baris program 34 berfungsi memangil function yang digunakan dalam proses *embedding* pesan ke dalam gambar, yang akan disimpan pada byte array. Setelah disusun kembali, lalu Megubah array tadi menjadi integer array pada kode baris program 36 yang berfungsi untuk mengatur proses bitmap. Kode baris program 38 membuat bitmap/gambar yang sudah di *embedding* pesan. Kode baris program 44 – 51 berfungsi mengatur nilai pixel dari bitmap yang dibuat pada baris 38.

Setelah *embedding* selesai maka akan menjalankan kode baris program 52 – 54 yang berfungsi menambah pixel yang telah diatur ke dalam list result dan dikembalikan kembali menjadi tampilan gambar.

Pada kode baris program 34 merupakan proses penyisipan pesan ke dalam pixel yang dimana prosesnya akan digambar pada kode program berikut

```
1  private static byte[] encodeMessage(int[] integer_pixel_array, int
2  image_columns, int image_rows,MessageEncodingStatus messageEncodingStatus,
3                                              ProgressHandler progressHandler){
4
5      int channels = 3;
6      int shiftIndex = 4;
7      byte[] result = new byte[image_rows * image_columns * channels];
8      int resultIndex = 0;
9
10   for (int row = 0; row < image_rows; row++) {
11     for(int col = 0; col < image_columns; col++) {
12         int element = row * image_columns + col;
13         byte tmp;
14         for (int channelIndex = 0; channelIndex < channels; channelIndex++)
15         {
16          if (!messageEncodingStatus.isMessageEncoded()) {
17          tmp = (byte) ((((integer_pixel_array[element] >>
18                binary[channelIndex]) & 0xFF) & 0xFC) |
19                ((messageEncodingStatus.getByteArrayMessage()
20                [messageEncodingStatus.getCurrentMessageIndex()]
21                >> toShift[(shiftIndex++)% toShift.length]) & 0x3));
22
23             if (shiftIndex % toShift.length == 0) {
24                 messageEncodingStatus.incrementMessageIndex();
25                 if (progressHandler != null)
26                     progressHandler.increment(1);
27             }
28
29             if (messageEncodingStatus.getCurrentMessageIndex() ==
30                      messageEncodingStatus.getByteArrayMessage().length)
31                     {
32                         messageEncodingStatus.setMessageEncoded();
```

```
33                          if (progressHandler != null)
34                              progressHandler.finished();
35                      }
36              } else {
37                      tmp = (byte) ((((integer_pixel_array[element] >>
38                      binary[channelIndex]) & 0xFF)));
39               }
40                    result[resultIndex++] = tmp;
41              }
42          }
43      }
44        return result;
45  }
```

Gambar 4.27 Kode program penyisipan bit pesan

Gambar 4.27 merupakan algoritma steganografi menyisipkan pesan ke dalam bit – bit pixel gambar. Kode baris program 1 – 45 merupakan proses steganografi LSB melakukan penyisipan. kode baris program 5 merupakan definisi dari 3 huruf RGB, sedangkan kode baris program 6 mendifinisikan proses pergeserang/shift indeks sebanyak 4. Kode baris program 7 mendefinisikan *byte image* bertipe *byte array*. Kode baris program 8 mendifinisikan penomoran array yang dimulai dari nilai 0 terhadap baris dan kolom.

Kode baris program 10 – 43 merupakan proses penyisipan pesan ke dalam gambar dengan jumlah perulangan terhadap baris gambar, sedangkan kode baris program 11 berfungsi melakukan perulangan terhadap kolom gambar. Kode baris program 12 berfungsi untuk mengetahui tempat lokasi *pixel* yang akan disisipkan. Kode baris program 13 mendefinisikan variabel temporary *byte*.

Kode baris program 14 – 41 berfungsi melakukan perulangan sebanyak 3 kali sesuai yang terdiri dari RGB. Kode baris program 16 berfungsi pengecekan *true* and *false* status pesan yang didapat dari proses Gambar 4.26 kode baris program 14, jika false maka menjalankan program dengan kondisi yang ada pada baris program 17 – 35. Kode baris program 17 – 21 berfungsi menggeserkan bit pixel sebanyak 2 bit dan diganti dengan bit dari pesan.

Kode baris program 23 – 27 berfungsi untuk menambah proses durasi progress dialog untuk menyelesaikan proses pergeserang bit – bit yang terjadi pada kode baris program 17 – 21. Sedangkan kode baris program 29 – 35 berfungsi untuk menutup progress dialog dan mengganti kondisi messageEncodeStatus yang terdapat pada kode baris program 16 menjadi true untuk menjalankan program dari kode baris 36 – 39 yang dimana kode baris program 37 dan 38 menyimpan pixel hasi dari pergeseran ke dalam variabel temporary.

Kode baris program 40 berfungsi untuk meyimpan pixel dari temp ke dalam variabel result untuk dikembalikan ke function pada Gambar 4.26.

4.3.1.6 Source code algoritma MD 5

Source code algoritma MD 5 merupakan proses yang digunakan pada tahapan akhir dari proses dari alur pada Gambar 3.1. Setelah proses penyatuan kembali *stego image* yang didapat dari proses pada Gambar 4.26. Maka proses selanjutnya yaitu enkripsi file *stego Image.*

Berdasarkan alur dari Gambar 3.16 dalam melakukan enkripsi sebuah file *stego image* untuk mendapatkan nilai identik, harus melalui proses – prosesnya yang dibutuhkan untuk menghasil sebuah *Hash*, yang nantinya akan digunakan oleh menu pengecekan File. Berikut kode program algoritma MD 5 menjalankan fungsinya

```
1  public static String MD5_Hash(Bitmap stegoImage) {
2      MessageDigest m = null;
3
4      ByteArrayOutputStream baos = new ByteArrayOutputStream();
5      stegoImage.compress(Bitmap.CompressFormat.PNG, 100, baos);
6      byte[] bitmapBytes = baos.toByteArray();
7
8      try {
9          m = MessageDigest.getInstance("MD5");
10     } catch (NoSuchAlgorithmException e) {
11         e.printStackTrace();
12     }
13
14     m.update(bitmapBytes,0,bitmapBytes.length);
15     String hash = new BigInteger(1, m.digest()).toString(16);
16     return hash;
17 }
```

Gambar 4.28 Algoritma MD 5

Gambar 4.28 merupakan proses mendapatkan nilai *Hash* dari sebuah File *Stego Image*, proses awalnya mengubah data biner file *stego image* menjadi byte array yang ditunjukan pada kode baris program 4 – 6.

Setelah itu melakukan proses MD 5 yang ditunjukan pada kode baris program 8 – 12. Fungsi kode baris ini untuk memanggil library *message digest* yang dimana menggunakan algoritma MD5. Lalu kode baris 14 – 15 merupakan proses algoritma MD5 melakukan generate untuk mendapatkan sebuah Hash dari file *stego image*.

4.3.2 Implementasi Source Code Extracting dan Dekripsi

Dalam mengimplementasikan proses *extracting* dan dekripsi dibutuhkan beberapa *source code* program untuk membangun hasil dari perangkat lunak ini, yang memiliki alur proses seperti pada Gambar 3.2. Tahapan – tahapan yang diperlukan untuk menghasilkan sebuah keluaran yang dibutuhkan, harus melalui beberapa proses metode yang digunakan. Pada

pengimplementasian ini, alur tahapannya dimulai dari proses Steganografi melakukan *extracting*. Setelah itu, kriptografi yang melakukan penyempurnaan pesan kembali untuk dapat dibaca kembali, melalui proses Dekripsi algoritma Kombinasi *AES* dengan *Caesar Cipher, Hill Cipher,* dan *Affine Cipher*.

Berikut *Source code* algoritma pengambilan pesan kembali dari wadah *stego image*, yang akan dijabarkan sebagai berikut.

4.3.2.1 Source Code Extracting

Seperti konsep alur yang dibuat pada Gambar 3.15 pertama kali yang harus dilakukan setelah *stego image* dimasukan yaitu mengambil nilai pixel dari file *stego image*. Berikut kode program pengambilan nilai pixel dari File *Stego Image*.

```
1   public static String decodeMessage(List<Bitmap> encodedImages) {
2
3       MessageDecodingStatus messageDecodingStatus = new
4                                                     MessageDecodingStatus();
5
6       for (Bitmap bit : encodedImages) {
7           int[] pixels = new int[bit.getWidth() * bit.getHeight()];
8           bit.getPixels(pixels, 0, bit.getWidth(), 0, 0, bit.getWidth(),
9                   bit.getHeight());
10
11          byte[] b;
12          b = Utility.convertArray(pixels);
13
14          decodeMessage(b, bit.getWidth(),
15                        bit.getHeight(),messageDecodingStatus);
16
17          if (messageDecodingStatus.isEnded())
18              break;
19      }
20
21      return messageDecodingStatus.getMessage();
22  }
```

Gambar 4.29 Kode program *extracting Stego-Image*

Gambar 4.29 merupakan kode program yang ada didalam algoritma Steganografi, *function* ini memiliki 2 fungsi yaitu pengambilan nilai pixel dari *stego image* yang nantinya akan di proses pada *function* lain dan yang kedua yaitu mengambil pesan untuk ditampilkan pada aplikasi. Setelah itu, berikut penjelasan fungsi kode baris program yang terdapat Gambar 4.29.

Kode baris program 3 berfungsi untuk menampung nilai *string* yang berguna untuk mengetahui status akhir dari function ini, kode baris program 6 – 19 melakukan perulangan nilai bit pixel dari parameter *encodeImage.*

Kode baris 7 berfungsi mengambil total keseluruhan *stego image* dari hasil perkalian tinggi dan lebar filenya. Kode baris program 9 – 8 berfungsi mengambil nilai bit pixel dari array pada kode baris 7. Kode baris 12 berfungsi mengubah *byte array* menjadi *integer array*, fungsi ini digunakan untuk *function* pada proses pengambilan bit pesan yang terdapat di *stego image*.

Kode baris program 14 merupakan proses *extraction*, yang dimana prosesnya dilakukan pada *function* yang ada didalam Steganografi. Kode baris 17 – 18 berfungsi melakukan pengecekan hasil dari proses 14. Jika proses terpenuhi, maka mengakhiri perulangan yang nantinya menjalankan kode akhir dari baris program 21 yang berfungsi mengambil isi pesan tersebut untuk ditampilkan pada aplikasi.

Setelah mengetahui fungsi dari *function* tersebut. Berikut kode program pengambilan pesan dari *stego image*, dimana hasil masukannya dari kode baris program 14.

```
1  private static void decodeMessage(byte[] byte_pixel_array, int image_columns,
2                                    int    image_rows,    MessageDecodingStatus
3  messageDecodingStatus) {
4
5      Vector<Byte> byte_encrypted_message = new Vector<>();
6
7      int shiftIndex = 4;
8      byte tmp = 0x00;
9
10     for (byte aByte_pixel_array : byte_pixel_array) {
11         tmp = (byte) (tmp | ((aByte_pixel_array << toShift[shiftIndex
12                 % toShift.length]) & andByte[shiftIndex++ %
13                 toShift.length]));
14      if (shiftIndex % toShift.length == 0) {
15          byte_encrypted_message.addElement(tmp);
16          byte[] nonso = { byte_encrypted_message.elementAt
17                           (byte_encrypted_message.size() - 1)
18                         };
19          String str = new String(nonso, Charset.forName("ISO-8859-1"));
20
21          if(messageDecodingStatus.getMessage().endsWith(END_MESSAGE_COSTANT))
22             {
23               Log.i("TEST", "Decoding ended");
24               byte[] temp = new byte[byte_encrypted_message.size()];
25               for (int index = 0; index < temp.length; index++){
26                    temp[index] = byte_encrypted_message.get(index);
27             }
28
29               String stra = new String(temp, Charset.forName("ISO-8859- 1"));
30               messageDecodingStatus.setMessage(stra.substring(0,
31               stra.length() - 1));
32               messageDecodingStatus.setEnded();
33
34               break;
35             } else {
36               messageDecodingStatus.setMessage(messageDecodingStatus.
37               getMessage() + str);
38
39               if (messageDecodingStatus.getMessage().length() ==
40                  START_MESSAGE_COSTANT.length()
```

```
41                      && !START_MESSAGE_COSTANT.equals
42                      (messageDecodingStatus.getMessage()))
43                  {
44                      messageDecodingStatus.setMessage("");
45                      messageDecodingStatus.setEnded();
46                      break;
47                  }
48              }
49              tmp = 0x00;
50          }
51      }
52      if (!Utility.isStringEmpty(messageDecodingStatus.getMessage()))
53          try {
54               messageDecodingStatus.setMessage(messageDecodingStatus.
55               getMessage().substring(START_MESSAGE_COSTANT.length(),
56               messageDecodingStatus.getMessage().length() -
57               END_MESSAGE_COSTANT.length()));
58          } catch (Exception e) {
59              e.printStackTrace();
60          }
61  }
```

Gambar 4.30 Kode program pengambilan bit pesan di *stego image*

Gambar 4.30 merupakan kode program *function* pengambilan bit – bit pesan yang terdapat pada pixel *stego image*, yang dimana kode baris memiliki fungsinya. Berikut fungsi dari kode program Gambar 4.30.

Kode baris program 5 berfungsi mendefinisikan variabel vector dari pesan yang akan di ekstrak. Kode baris program 7 berfungsi untuk mengidefinisikan shiftIndeks untuk dipergunakan dalam pergeseran bit. kode baris program 8 berfungsi mendefinisikan variabel temporary bertipe byte.

Kode baris program 10 - 51 berfungsi untuk melakukan perulangan berdasarkan jumlah pixel dari gambar/image. Kode baris program 11 – 13 berfungsi mendapatkan 2 bit terakhir dari pixel. Kode baris program 14 berfungsi untuk melakukan pengecekan jika shiftIndeks mod toShiftLength = 0 maka dilakukan proses ekstrak. Kode baris program 25 berfungsi untuk menambahkan temporary byte ke dalam variabel byte_encrypt_message.

Pada kode baris program 21 – 34 berfungsi pengecekan karakter " #!@ " yang ada di proses *embedding* pada Gambar 4.26 kode baris 7 dan 8. Jika terdapat " #!@ " , maka menjalankan kode baris program 24 untuk mengambil byte *ciphertext* lalu dimasukan dalam tempat penyimpanan sementara yang diproses 25 – 27. Kode baris program 29 – 31 mengubah byte menjadi string lalu menghapus karakter tambahan, setelah itu program mendapatkan status decoding selesai pada program 32.

Setelah pesan sudah di ekstrak, langkah berikutnya yaitu proses dekripsi dengan algoritma kriptografi. Berikut *source code* implementasi kriptografi melakukan dekripsi.

4.3.2.2 Source Code Kombinasi Caesar cipher dan AES

Pada tahap melakukan proses dekripsi menggunakan algoritma AES, terdapat alur yang harus terpenuhi seperti pada Gambar 3.10, yang dimana pertama kali sebelum melakukan proses dekripsi, algoritma AES membutuhkan sebuah masukan pesan dan kunci. Pesan masukan ini, merupakan hasil dari proses *extraction* pada Gambar 4.29 dan kunci masukan ini merupakan hasil dari proses enkripsi algoritma *Caesar cipher* dengan menggunakan kode program dari Gambar 4.22.

Setelah pesan dan kunci terpenuhi, langkah selanjutnya yaitu pencarian kunci ekspansi setiap putaran proses algoritma AES ini. Proses pencarianya, kunci ekspansi ini menggunakan kode program pada Gambar 4.24.

Proses selanjutnya dari pencarian kunci ekspansi yaitu proses dekripsi algoritma AES untuk mengembalikan pesan acak tersebut. Berikut kode implementasi program algoritma AES melakukan proses dekripsi

```
1  public byte[] decrypt(byte[] cipher) {
2      byte [] a = new byte[BLOCK_SIZE];
3      byte [] ta = new byte[BLOCK_SIZE];
4      byte [] Kdr;
5      int    i, j, k, row, col;
6      Kdr = Kd[0];
7
8      for (i = 0; i < BLOCK_SIZE; i++){
9           a[i] = (byte)(cipher[i] ^ Kdr[i]);
10     }
11     for (int r = 1; r < numRounds; r++) {
12         Kdr = Kd[r];
13         for (i = 0; i < BLOCK_SIZE; i++) {
14             row = i % COL_SIZE;
15             k = (i + BLOCK_SIZE - (row_shift[row] * COL_SIZE)) % BLOCK_SIZE;
16             ta[i] = a[k];
17         }
18         for (i = 0; i < BLOCK_SIZE; i++){
19             a[i] = Si[ta[i] & 0xFF];
20         }
21         for (i = 0; i < BLOCK_SIZE; i++){
22             ta[i] = (byte)(a[i] ^ Kdr[i]);
23         }
24         for (col = 0; col < NUM_COLS; col++) {
25             i = col * COL_SIZE;
26             a[i]   = (byte)(mul(0x0e,ta[i]) ^ mul(0x0b,ta[i+1]) ^
27                             mul(0x0d,ta[i+2]) ^ mul(0x09,ta[i+3]));
28             a[i+1] = (byte)(mul(0x09,ta[i]) ^ mul(0x0e,ta[i+1]) ^
29                             mul(0x0b,ta[i+2]) ^ mul(0x0d,ta[i+3]));
30             a[i+2] = (byte)(mul(0x0d,ta[i]) ^ mul(0x09,ta[i+1]) ^
31                             mul(0x0e,ta[i+2]) ^ mul(0x0b,ta[i+3]));
32             a[i+3] = (byte)(mul(0x0b,ta[i]) ^ mul(0x0d,ta[i+1]) ^
33                             mul(0x09,ta[i+2]) ^ mul(0x0e,ta[i+3]));
34         }
35     }
36     Kdr = Kd[numRounds];
37     for (i = 0; i < BLOCK_SIZE; i++) {
38         row = i % COL_SIZE;
```

```
39          k = (i + BLOCK_SIZE - (row_shift[row] * COL_SIZE)) % BLOCK_SIZE;
40          ta[i] = a[k];
41      }
42      for (i = 0; i < BLOCK_SIZE; i++){
43          ta[i] = Si[ta[i] & 0xFF];
44      }
45      for (i = 0; i < BLOCK_SIZE; i++){
46         a[i] = (byte)(ta[i] ^ Kdr[i]);
47      }
48
49
50      return (a);
51  }
```

Gambar 4.31 Kode program dekripsi AES-128 bit

Gambar 4.31 merupakan implementasi kode program algoritma AES-128 dalam melakukan proses dekripsi. Setiap kode baris memiliki tahapan – tahapan proses melakukan dekripsi. Berikut penjelasan kode baris program pada Gambar 4.31.

Kode baris program 8 – 10 merupakan proses *AddRoundKey* sebelum melakukan putaran pertama, fungsi ini untuk mendapatkan *state* yang akan digunakan pada putaran pertama, proses ini melakukan XOR antara pesan dengan kunci *chipertext* yang didapat dari Gambar 4.22.

Kode baris program 13 – 17 merupakan tahapan *InvShiftRows,* yang berfungsi melakukan pergeseran sebuah *state*. Kode baris program 18 – 20 merupaka proses *InvSubBytes* antara *state* dengan tabel Inv-Box. Kode baris program 21 – 23 merupakan proses *AddRoundKey* yang berfungsi untuk melakukan XOR sebuah *state* pesan dengan kunci ekspansi yang didapat pada Gambar 4.24. Selanjutnya, Kode baris program 24 – 34 merupakan proses *InvMixColumns* yang berfungsi melakukan XOR antara *state* dengan *state* konstan, yang dimana state konstannya berbeda dengan state konstan enkripsi.

Kode baris program 37 – 47 merupakan tahapan akhir proses dekripsi. proses tahapan terdapat 3 program, antara lain *InvShiftRows, InvSubByte*, dan *AddRoundKey*. Kode baris program 50 berfungsi mengembalikan nilai dari proses yang terdapat pada kode baris program 45 – 47 berupa bentuk *string* pesan.

Setelah proses dekripsi algoritma AES-128 mengembalikan pesan, proses selanjut yaitu proses pengembalian karakter tambahan yang didapat pada proses enkripsi algoritma AES-128. Berikut kode program normalisasi algoritma AES-128.

```
1   public String prosesNormalisasi(String pesan){
2       StringBuilder str = new StringBuilder();
3       String zz;
4       if (pesan.substring(pesan.length()-15).equals("...............")) {
5           zz = pesan.substring(0, pesan.length() - 15);
```

```
6           str.append(zz);
7       }else if (pesan.substring(pesan.length()-14).equals(".............")) {
8           zz = pesan.substring(0, pesan.length() - 14);
9           str.append(zz);
10      }else if (pesan.substring(pesan.length()-13).equals("............")) {
11          zz = pesan.substring(0, pesan.length() - 13);
12          str.append(zz);
13      }else if (pesan.substring(pesan.length()-12).equals("...........")) {
14          zz = pesan.substring(0, pesan.length() - 12);
15          str.append(zz);
16      }else if (pesan.substring(pesan.length()-11).equals("..........")) {
17          zz = pesan.substring(0, pesan.length() - 11);
18          str.append(zz);
19      }else if (pesan.substring(pesan.length()-10).equals(".........")) {
20          zz = pesan.substring(0, pesan.length() - 10);
21          str.append(zz);
22      }else if (pesan.substring(pesan.length()-9).equals("........")) {
23          zz = pesan.substring(0, pesan.length() - 9);
24          str.append(zz);
25      }else if (pesan.substring(pesan.length()-8).equals(".......")) {
26          zz = pesan.substring(0, pesan.length() - 8);
27          str.append(zz);
28      }else if (pesan.substring(pesan.length()-7).equals("......")) {
29          zz = pesan.substring(0, pesan.length() - 7);
30          str.append(zz);
31      }else if (pesan.substring(pesan.length()-6).equals(".....")) {
32          zz = pesan.substring(0, pesan.length() - 6);
33          str.append(zz);
34      }else if (pesan.substring(pesan.length()-5).equals("....")) {
35          zz = pesan.substring(0, pesan.length() - 5);
36          str.append(zz);
37      }else if (pesan.substring(pesan.length()-4).equals("...")) {
38          zz = pesan.substring(0, pesan.length() - 4);
39          str.append(zz);
40      }else if (pesan.substring(pesan.length()-3).equals("..")) {
41          zz = pesan.substring(0, pesan.length() - 3);
42          str.append(zz);
43      }else if (pesan.substring(pesan.length()-2).equals(".")) {
44          zz = pesan.substring(0, pesan.length() - 2);
45          str.append(zz);
46      }else if (pesan.substring(pesan.length()-1).equals(".")) {
47          zz = pesan.substring(0, pesan.length() - 1);
48          str.append(zz);
49      }else{
50          zz = pesan;
51          str.append(zz);
52      }
53      return str.toString();
54  }
```

Gambar 4.32 Kode program normalisasi karakter AES

Gambar 4.32 merupakan proses penghapusan karakter tambahan/ normalisasi karakter. Fungsi ini menghapus karakter tambahan dari proses enkripsi algoritma AES-128 yang ditunjukan pada Gambar 4.23. dengan adanya *function* ini, proses *Hill Cipher* akan menerima jumlah karakter yang sama.

4.3.2.3 Source Code Dekripsi Hill Cipher

Alur proses yang terdapat pada Gambar 3.7 dalam melakukan proses dekripsi *hill cipher*, tahapan – tahapan yang diperlukan sama seperti pada saat proses enkripsi. Perbedaan disini, terdapat proses pencarian kunci *invertible matriks* yang digunakan pada saat dekripsi dan normalisasi karakter tambahan dari proses penambahan karakter yang didapat dari proses enkripsi. Berikut penjelasan alur proses implementasi dekripsi *Hill Cipher*.

Proses awal *Hill Cipher* yaitu sebuah masukan pesan yang didapat dari proses Gambar 4.32, pesan ini didapat dari proses algoritma sebelumnya. Setelah itu, proses melakukan pengecekan jumlah karakter pesan yang didapat. Kode proses tersebut terdapat pada Gambar 4.19. Setelah itu, proses dekripsi *Hill Cipher*.

Sebelum melakukan proses perhitungan dekripsi, program mencari kunci *invertible matriks* dari kunci enkripsi *Hill Cipher* terlebih dahulu yang diimplementasi pada kode program berikut.

```
1  public int hitungInvers(int[][] matrix) {
2    int determinan;
3    int a11 = matrix[0][0], a12 = matrix[0][1], a13 = matrix[0][2],
4        a14 = matrix[0][3];
5    int a21 = matrix[1][0], a22 = matrix[1][1], a23 = matrix[1][2],
6        a24 = matrix[1][3];
7    int a31 = matrix[2][0], a32 = matrix[2][1], a33 = matrix[2][2],
8        a34 = matrix[2][3];
9    int a41 = matrix[3][0], a42 = matrix[3][1], a43 = matrix[3][2];
10       a44 = matrix[3][3];
11   determinan = (a11 * a22 * a33 * a44) + (a11 * a23 * a34 * a42) +
12                (a11 * a24 * a32 * a43) - (a11 * a24 * a33 * a42) -
13                (a11 * a23 * a32 * a44) - (a11 * a22 * a34 * a43) -
14                (a12 * a21 * a33 * a44) - (a13 * a21 * a34 * a42) -
15                (a14 * a21 * a32 * a43) + (a14 * a21 * a33 * a42) +
16                (a13 * a21 * a32 * a44) + (a12 * a21 * a34 * a43) +
17                (a12 * a23 * a31 * a44) + (a13 * a24 * a31 * a42) +
18                (a14 * a22 * a31 * a43) - (a14 * a23 * a31 * a42) -
19                (a13 * a22 * a31 * a44) - (a12 * a24 * a31 * a43) -
20                (a12 * a23 * a34 * a41) - (a13 * a24 * a32 * a41) -
21                (a14 * a22 * a33 * a41) + (a14 * a23 * a32 * a41) +
22                (a13 * a22 * a34 * a41) + (a12 * a24 * a33 * a41);
23   int A11 = (a22 * a33 * a44) + (a23 * a34 * a42) + (a24 * a32 * a43) -
24             (a24 * a33 * a42) - (a23 * a32 * a44) - (a22 * a34 * a43);
25   int A12 = -(a12 * a33 * a44) - (a13 * a34 * a42) - (a14 * a32 * a43) +
26             (a14 * a33 * a42) + (a13 * a32 * a44) + (a12 * a34 * a43);
27   int A13 = (a12 * a23 * a44) + (a13 * a24 * a42) + (a14 * a22 * a43) -
28             (a14 * a23 * a42) - (a13 * a22 * a44) - (a12 * a24 * a43);
29   int A14 = -(a12 * a23 * a34) - (a13 * a24 * a32) - (a14 * a22 * a33) +
30             (a14 * a23 * a32) + (a13 * a22 * a34) + (a12 * a24 * a33);
31   int A21 = -(a21 * a33 * a44) - (a23 * a34 * a41) - (a24 * a31 * a43) +
32             (a24 * a33 * a41) + (a23 * a31 * a44) + (a21 * a34 * a43);
33   int A22 = (a11 * a33 * a44) + (a13 * a34 * a41) + (a14 * a31 * a43) -
34             (a14 * a33 * a41) - (a13 * a31 * a44) - (a11 * a34 * a43);
35   int A23 = -(a11 * a23 * a44) - (a13 * a24 * a41) - (a14 * a21 * a43) +
36             (a14 * a23 * a41) + (a13 * a21 * a44) + (a11 * a24 * a43);
37   int A24 = (a11 * a23 * a34) + (a13 * a24 * a31) + (a14 * a21 * a33) -
38             (a14 * a23 * a31) - (a13 * a21 * a34) - (a11 * a24 * a33);
```

```
39    int A31 = (a21 * a32 * a44) + (a22 * a34 * a41) + (a24 * a31 * a42) -
40              (a24 * a32 * a41) - (a22 * a31 * a44) - (a21 * a34 * a42);
41    int A32 = -(a11 * a32 * a44) - (a12 * a34 * a41) - (a14 * a31 * a42) +
42              (a14 * a32 * a41) + (a12 * a31 * a44) + (a11 * a34 * a42);
43    int A33 = (a11 * a22 * a44) + (a12 * a24 * a41) + (a14 * a21 * a42) -
44              (a14 * a22 * a41) - (a12 * a21 * a44) - (a11 * a24 * a42);
45    int A34 = -(a11 * a22 * a34) - (a12 * a24 * a31) - (a14 * a21 * a32) +
46              (a14 * a22 * a31) + (a12 * a21 * a34) + (a11 * a24 * a32);
47    int A41 = -(a21 * a32 * a43) - (a22 * a33 * a41) - (a23 * a31 * a42) +
48              (a23 * a32 * a41) + (a22 * a31 * a43) + (a21 * a33 * a42);
49    int A42 = (a11 * a32 * a43) + (a12 * a33 * a41) + (a13 * a31 * a42) -
50              (a13 * a32 * a41) - (a12 * a31 * a43) - (a11 * a33 * a42);
51    int A43 = -(a11 * a22 * a43) - (a12 * a23 * a41) - (a13 * a21 * a42) +
52              (a13 * a22 * a41) + (a12 * a21 * a43) + (a11 * a23 * a42);
53    int A44 = (a11 * a22 * a33) + (a12 * a23 * a31) + (a13 * a21 * a32) -
54              (a13 * a22 * a31) - (a12 * a21 * a33) - (a11 * a23 * a32);
55
56    matrikInvers[0][0] = A11, matrikInvers[0][1] = A12;
57    matrikInvers[0][2] = A13, matrikInvers[0][3] = A14;
58    matrikInvers[1][0] = A21, matrikInvers[1][1] = A22;
59    matrikInvers[1][2] = A23, matrikInvers[1][3] = A24;
60    matrikInvers[2][0] = A31, matrikInvers[2][1] = A32;
61    matrikInvers[2][2] = A33, matrikInvers[2][3] = A34;
62    matrikInvers[3][0] = A41, matrikInvers[3][1] = A42;
63    matrikInvers[3][2] = A43, matrikInvers[3][3] = A44;
64
65    BigInteger Multiplikatif = BigInteger.valueOf(determinan).
66    modInverse(BigInteger.valueOf(modulo));
67
68    int MultiplikatifDet = Multiplikatif.intValue();
69    for (int i = 0; i < matrikInvers.length; i++) {
70         for (int j = 0; j < matrikInvers[i].length; j++) {
71             if (matrikInvers[i][j] < 0) {
72                  matrikInvers[i][j] = modulo - (Math.abs(matrikInvers[i][j])
73                  % modulo);
74             }else{
75                       matrikInvers[i][j] = matrikInvers[i][j] % modulo;
76                  }
77                  matrikInvers[i][j] = (int) (matrikInvers[i][j] *
78                                        MultiplikatifDet);
79                  matrikInvers[i][j] = matrikInvers[i][j] % modulo;
80              }
81          }
82      }
83      return determinan;
84  }
```

Gambar 4.33 Kode program mencari kunci dekrip *Hill Cipher*

Gambar 4.33 merupakan kode program dari algoritma Hill Cipher untuk mendapatkan sebuah kunci dekripsi. Meskipun kunci enkripsi pada penelitian ini statis, tapi pada proses dekripsi kode program ini berfungsi untuk mencari kunci dekripsi. Berikut fungsi kode baris untuk mendapatkan *invertible matriks* yang terdapat pada Gambar 4.33.

Kode baris 69 – 82 merupakan proses perhitungan untuk mendapatkan nilai *invertible matriks*, yang dimana prosesnya mengkalikan antara determinan dengan Matriks Invers yang didapat pada kode baris program 23 – 63. Kode baris 83 berfungsi mengambil nilai hasil dari perhitungan tersebut sebagai kunci *invertible matriks*.

Setelah kunci didapat, pesan dan kunci tersebut diproses dengan kode program pada Gambar 4.20, yang merupakan proses dekripsi *Hill Cipher*. Proses selanjutnya setelah algoritma *Hill Cipher* melakukan dekripsi, yaitu melakukan penghapusan karakter tambahan dari hasil proses dekripsi. Karakter tambahan ini didapat dari proses enkripsi. Berikut kode program penghapusan karakter tambahan di algoritma *Hill Cipher*

```
1   static String normalisasiPesan(String text) {
2       if (text.substring(text.length()-3).equals("...")) {
3           totalHasilDeskrip = text.substring(0,text.length()-3);
4       } else if (text.substring(text.length()-2).equals("..")) {
5           totalHasilDeskrip = text.substring(0,text.length()-2);
6       } else if (text.substring(text.length()-1).equals(".")) {
7           totalHasilDeskrip = text.substring(0,text.length()-1);
8       }
9       else {
10          totalHasilDeskrip = text;
11      }
12      return totalHasilDeskrip;
13  }
```

Gambar 4.34 Penghapusan karakter tambahan Hill Cipher

Gambar 4.34 merupakan kode program yang hanya digunakan pada proses dekripsi algoritma *Hill Cipher*, fungsi ini merupakan fungsi terakhir yang digunakan untuk mendapatkan pesan semula tanpa ada karakter tambahan, dimana prosesnya ditunjukan pada kode baris program 2 – 12.

Hasil penghapusan karakter tambahan ini, akan diproses lagi dengan algoritma *Affine Cipher*.

4.3.2.4 Source Code Dekripsi Affine Cipher

Dekripsi *affine cipher* ini, merupakan proses terakhir pesan benar – benar dapat dibaca kembali. Seperti alur konsep yang dibuat pada Gambar 3.5. Tahapan awal implementasi kode program dekripsi *Affine Cipher* yaitu pencarian kunci invers yang memenuhi inver modular, yang dimana kode programnya dapat dilihat pada Gambar 4.17.

Kode program tersebut, untuk proses enkripsi hanya digunakan sebagai pencegahan kunci masukan yang tidak memiliki nilai invers, sedangkan untuk proses dekripsi digunakan untuk mengambil nilai invers yang didapat dari kunci masukkan sebagai kunci dekripsi yang didapat pada kode baris 5.

Proses selanjut setelah mendapatkan kunci invers dari masukan, yaitu proses melakukan dekripsi menjadi pesan kembali. Berikut kode program implementasi dekripsi algoritma *hill cipher*.

```
1  private String ProsesDekrip(String pesanRahasia, int kunci_a, int kunci_b) {
2      StringBuilder str = new StringBuilder();
3      String hasilDekrip="";
4      kunciC = kunci_b + 15;
5      kunciD = kunci_b + kunciC;
6      for (int i = 0; i < pesanRahasia.length(); i++) {
7          kunciE = kunciC + i;
8
9          if ((i - 1)%5 == 0) {
10             hasilRotasi = kunci_a*( (int)pesanRahasia.charAt(i) - kunci_b);
11             modular = floorMod(hasilRotasi, modulus);
12             char a2 = (char) ((char) modular);
13             hasilDekrip = String.valueOf(a2);
14             str.append(hasilDekrip);
15         }
16         if((i - 2)%5 == 0){
17             hasilRotasi = kunci_a*( (int)pesanRahasia.charAt(i) + kunci_b);
18             modular = floorMod(hasilRotasi, modulus);
19             char a2 = (char) ((char) modular);
20             hasilDekrip = String.valueOf(a2);
21             str.append(hasilDekrip);
22         }
23         if((i - 3)%5 == 0){
24             hasilRotasi = kunci_a*( (int)pesanRahasia.charAt(i) - kunciC);
25             modular = floorMod(hasilRotasi, modulus);
26             char a2 = (char) ((char) modular);
27             hasilDekrip = String.valueOf(a2);
28             str.append(hasilDekrip);
29         }
30         if((i - 4)%5 == 0){
31             hasilRotasi = kunci_a*( (int)pesanRahasia.charAt(i) + kunciD);
32             modular = floorMod(hasilRotasi, modulus);
33             char a2 = (char) ((char) modular);
34             hasilDekrip = String.valueOf(a2);
35             str.append(hasilDekrip);
36         }
37         if((i - 5)%5 == 0){
38             hasilRotasi = kunci_a*( (int)pesanRahasia.charAt(i) - kunciE);
39             modular = floorMod(hasilRotasi, modulus);
40             char a2 = (char) ((char) modular);
41             hasilDekrip = String.valueOf(a2);
42             str.append(hasilDekrip);
43         }
44     }
45     return str.toString();
46 }
```

Gambar 4.35 Kode program dekripsi *Affine Cipher*

Gambar 4.35 merupakan proses implementasi program *affine cipher* melakukan dekripsi pesan acak yang didapat dari dekripsi *hill cipher* menjadi pesan yang dapat dibaca kembali, proses kode program ini merupakan proses terakhir kriptografi melakukan dekripsi. Berikut penjelasan kode program yang terdapat pada Gambar 4.35.

Dapat dilihat, konsep kode program dekripsi *affine cipher* ini tidak berbeda jauh dengan kode program enkripsi pada Gambar 4.18. Perbedaan disini, merupakan sebuah masukan kunci dan penjumlahan aritmetika integer kunci pergeseran yang ada dikode baris program 10, 17,

21, 31, dan 38. Kunci yang disimbolkan dengan penulisan “kunci_a” disini merupakan kunci invers hasil dari proses pencarian kunci dari kunci masukan dan untuk proses kunci pergeseran disini, aritmetika integer dimulai dari “ - ” dan “ + ”. Sedangkan untuk proses enkripsi, kunci pergeseran dimulai dari “ + ” dan “ - ”.

4.3.3 Implementasi Source Code File Checksum

Berdasarkan alur pada Gambar 3.3 dalam melakukan proses pengecekan integritas sebuah file *Stego Image*, memiliki beberapa tahapan – tahapan diantara lain yaitu tahapan proses pengecekan file stego image dengan algoritma MD5 dan tahapan perbandingan *Hash* dari file *stego image* dengan *Hash* yang di masukan oleh si penerima. Berikut kode program proses pengecekan file stego image.

Dalam proses pengecekan nilai MD 5 dari sebuah file Stego image sama dengan proses yang terdapat pada Gambar 4.28. Prosesnya mengambil byte array stego image, lalu di proses dengan library *Message Digest* yang menghasilkan sebuah *hash*. Setelah itu, dapat melakukan pengecekan keaslian dengan kode program sebagai berikut.

```
1  process_button.setOnClickListener(new View.OnClickListener() {
2      @Override
3      public void onClick(View v) {
4        if(currentText.getText().toString().equals(originalText.getText()
5           .toString())){
6            DialogSuccess();
7          }else{
8            DialogError();
9          }
10     }
11 });
```

Gambar 4.36 Kode program File Checksum

Gambar 4.36 merupakan proses dari menu *File checksum* dalam melakukan proses pengecekan ke aslian data yang ditunjukan pada baris program 4. Kode baris ini berfungsi membandingkan setiap karakter *Hash* dari *file* dengan *Hash* yang diinput, dimana jika Hash memiliki ke cocokan, kode baris program 6 memberikan informasi kecocokan nilai *Hash* (Identik). Sedangkan untuk kode program 8 berfungsi jika kedua karakter *Hash* tidak cocok.

4.4 Pengujian Aplikasi

Pengujian aplikasi ini bertujuan untuk mengetahui keberhasilan program yang dibuat dari setiap proses – prosesnya dalam melakukan tujuannya, yang dimana pada tahap ini akan

menguji keberhasilan hasil dari proses sistem dalam melakukan enkripsi dan *embedding* (Encode), hasil dari *extracting* dan dekripsi (Decode), hasil dari perbandingan antara *Cover Image* dengan *Stego Image*, dan serta pengujian integritas file stego image yang dikirim lewat aplikasi. Pengujian ini menggunakan password " informatika ", " 16 ", dan " 5 " dengan total karakter pesan yang berbeda - beda. Maka hasil pengujian aplikasi akan dijelaskan sebagai berikut.

4.4.1 Pengujian Enkripsi dan Embedding

Pengujian Enkripsi dan *embedding* ini digunakan untuk mengetahui keberhasilan sistem dalam melakukan tujuannya. Berikut hasi pengujian pada penelitian ini, yang ditampilkan pada tabel berikut

Table 4.1 Pengujian Enkripsi dan *Embedding*

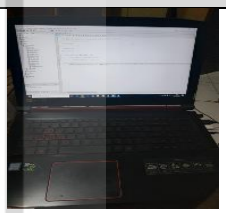
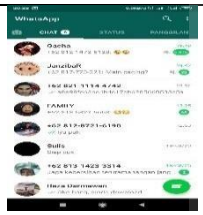
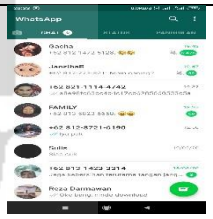
No	Cover Image	Jumlah Karakter	Proses Enkripsi	Proses Embedding	Waktu Proses	Nama Hasil Embedding
1	 Laptop.jpg	680	Sukses	Sukses	00.00.473	 StegoLaptop.png
2	 ScrenShootWA .jpg	1614	Sukses	Sukses	00.00.518	 ScreenShootWA .png
3	 Lena.png	2161	Sukses	Sukses	00.00.569	 LenaStego.png

Table 4.1 Pengujian Enkripsi dan *Embedding* (Lanjutan)

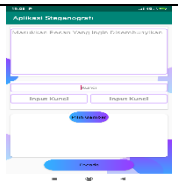
4	 Encode.png	3340	Sukses	Sukses	00.00.910	 EncodeStego.png

Table 4.1 merupakan tabel pengujian pada proses enkripsi dan *embedding* yang hasilnya tersebut dipergunakan untuk mengetahui keberhasilan atau kegegalan dari proses enkripsi dan *embedding* pada pesan dan gambar. Seperti hasil yang terdapat pada Table 4.1, saat ini sistem tidak memiliki kesalahan dalam menjalankan proses enkripsi dan *embedding*. Waktu proses hasil pengujian menunjukan bahwa dengan jumlah panjang karakter yang berbeda menghasilkan waktu proses yang berbeda - beda, semakin besar karakter semakin tinggi waktu proses yang dibutuhkan program untuk menghasilkan *stego image*. Dikarenakan pada proses *embedding* program, terdapat proses menggambungkan kembali pixel – pixel gambar menjadi *stego image*. Aplikasi ini menghasilkan *stego image* berekstensi .png dikarenakan pada tahap penggabungan pixel – pixel gambar, library yang digunakan yaitu .png.

4.4.2 Pengujian Extracting dan Dekripsi

Table 4.2 Pengujian *Extracting* dan Dekripsi

No	Stego Image	Proses Extracting	Proses Dekripsi	Waktu Proses	Jumlah Karakter
1	Laptop.png	Sukses	Sukses	00.00.209	680
2	ScrenShoot WA.png	Sukses	Sukses	00.00.223	1614
3	Lena.png	Sukses	Sukses	00.00.238	2161
4	SCStegano .png	Sukses	Sukses	00.00.484	3340

Table 4.2. dipergunakan untuk mengetahui keberhasilan atau kegegalan dari proses *Extracting* dan dekripsi pada pesan yang ada didalam *stego image*. Seperti hasil yang ada ditabel, hasil dari proses *Extracting* pada program yang menunjukan saat ini, tidak mengalami

kegagalan yang dimana pada saat melakukan *extracting* dan dekripsi pesan yang terdapat didalam gambar dapat dibaca kembali. Waktu proses menujukan lebih cepat ketimbang pada waktu proses Enkripsi dan *Embbeding*. Yang dikarenakan pada saat pengambilan data bit – bit pesan yang terdapat di *stego image*, program hanya melakukan pengecekan karakter tambahan serta program tidak mengembalikan pixel – pixel gambar menjadi gambar kembali.

4.4.3 Perbandingan Image dan Stego-Image

Table 4.3 Perbandingan *Cover Image* dan *Stego Image*

No	Cover Image	Ukuran Cover Image (KB)	Ukuran Pixel Cover Image	Stego Image	Ukuran Stego Image (KB)	Ukuran Pixel Stego Image
1	Laptop.jpg	5.12	187x250	Laptop.png	71.92	187x250
2	ScrenshootWA.jpg	439	1080x2160	ScrenshootWA.png	610	1080x2160
3	Lena.png	474	512x512	LenaStego.png	559	512x512
4	SCStegano.png	89.08	1080x2160	SCSteganoStego.png	102	1080x2160

Table 4.3 merupakan perbandingan ukuran gambar dan pixel gambar pada *cover image* dengan *stego image*. Dari tabel ini, dapat diketahui perubahan yang terjadi pada gambar sebelum dilakukan proses *embedding* pesan. Proses *embedding* membuat ukuran gambar menjadi berubah, sedangkan untuk ukutan pixel tetap tidak berubah. Perubahan tersebut dikarenakan penambahan ukuran data yang masuk sehingga menambahkan jumlah ukuran gambar sebelumnya.

4.4.4 Pengujian Integritas File Stego Image

Pengujian integritas File Stego Image ini, penelitian menggunakan algoritma MD 5, tidak menggunakan algoritma Hashing lain. Dikarenakan ada beberapa keterbatasan dalam melakukan pengujian, yang dimana terdapat spesifikasi ponsel yang tidak memadai, yaitu RAM dibawah 2 GB. Pada proses penggunaan Hashing SHA-1 atau SHA-256 dalam percobaannya, aplikasi mengalami kerusakan diantara lain Aplikasi mengalami *Force close* ataupun membutuhkan proses yang amat sangat lama, sehingga tidak cocok untuk digunakan.

Dalam pengujian ini. Penelitian menggunakan *stego image* dengan nama "ScreenShoot.WA. png" yang terdapat pada Table 4.1. Berikut hasil pengujian File Checksum untuk mengetahui keaslian data yang akan ditampilkan pada tabel berikut

Table 4.4 Pengujian Integritas File *Stego Image*

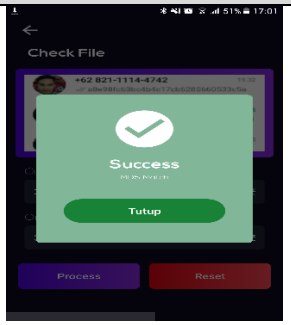
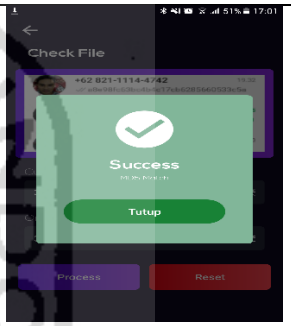
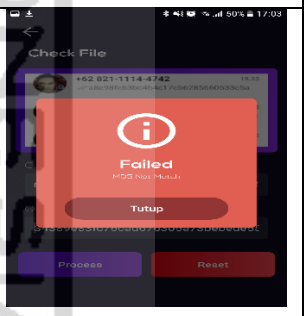
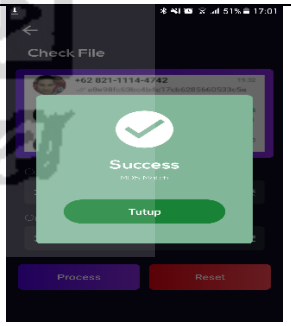
File Md5 Stego Image	Media Pengiriman	File Md5 Setelah terkirim	Hasil File Checksum
34389e83fc76cad676305a73bebede55	Bluetooth	34389e83fc76cad676305a73bebede55	
34389e83fc76cad676305a73bebede55	Gmail	34389e83fc76cad676305a73bebede55	
34389e83fc76cad676305a73bebede55	WhatsApp	6a4da5a0ddfeee316 6975512803d32df	
34389e83fc76cad676305a73bebede55	Telegram	34389e83fc76cad676305a73bebede55	

Table 4.4 merupakan hasil dari proses pengujian integritas *stego image* yang dikirim melalui aplikasi – aplikasi pengiriman data. Dapat dilihat dalam kolom Table 4.4 hasil dari file checksum, file *stego image* yang dikirim melalui aplikasi pengiriman file tanpa kompresi atau perubahan data dari aplikasinya, maka file *stego image* tidak mengalami kerusakan pada proses pengiriman, sehingga dapat melakukan *extracting* data pesan.

Sedangkan untuk file *stego image* yang dikirim melalui aplikasi media sosial terjadi perubahan file. Penyebab perubahannya itu dikarenakan aplikasi media sosial memiliki algoritma kompresi dalam pengiriman suatu file, sehingga terjadi perubahan pada file *stego image* yang dikirim. Dengan kerusakan ini, proses *extracting* tidak dapat dilakukan.

Oleh sebab itu, untuk mengurangi kerusakan – kerusakan yang terjadi dalam proses pengiriman sebuah file *stego image*. Maka berikut proses pengiriman file *stego image* yang dapat merusak file dan tidak merusak file dalam tabel berikut.

Table 4.5 Proses Pengiriman Melalui Aplikasi

Media Pengiriman	Proses Pengiriman	
	Tidak Merusak File Stego Image	Merusak File Stego Image (*)
Bluetooth	• Mengirim melalui menu share • Mengirim melalui file gallery atau file folder berkas/dokumen	
Gmail	• Mengirim melalui menu share aplikasi • Mengirim melalui file gallery atau file folder berkas/dokumen	• Mengirim dengan cara drag and drop
WhatsApp	• Mengirim melalui file folder dokumen/berkas yang terdapat di menu aplikasi WhatsApp	• Mengirim melalui menu share di file folder/gallery • Mengirim melalui file gallery yang terdapat di menu WhatsApp
Telegram	• Mengirim melalui file berkas yang terdapat di menu aplikasi telegram	• Mengirim melalui menu share aplikasi di file folder/gallery • Mengirim file melalui file gallery yang terdapat pada menu telegram

Table 4.5 merupakan hasil pengujian dari proses pengiriman melalui berbagai menu yang dapat digunakan untuk mengirim file *stego image* ke aplikasi lain. Terdapat proses – proses yang harus dihindari, dikarenakan dengan melakukan proses pengiriman yang dapat merusak file *stego image* pada proses pengirimannya, aplikasi tidak bisa mengembalikan pesan yang ada didalam *stego image* tersebut..

4.5 Hasil Pengujian Perbandingan

Pengujian ini dilakukan dengan membandingkan hasil dari metode – metode penelitian yang digunakan pada saat ini dengan penelitian terdahulu ataupun pengujian dengan menggunakan aplikasi lain untuk mengetahui hasil *ciphertext* yang didapat, setelah itu membandingkan kelemahan dan kelebihan dari metode yang digunakan.

4.5.1 Perbandingan Affine Cipher dengan penelitian terdahulu

Table 4.6 Perbandingan *chipertext affine cipher* dengan penelitian terdahulu

No	Penelitian Terdahulu	Pesan	Kunci A	Kunci B	Chipertext	Chipertext penelitian ini
1	(Wibowo et al. 2014)	SELAMAT PAGI	11	24	OQPYAYZH YMI	?2w|;J-DA
2	(Babu 2017)	W52CFA	7	10	UL0YJK	~ VqJh
3	(Juliadi et al. 2013)	SEMANGAT_ ’45	3	7	VTDHGZHY 9ERU	WaZNqKv5e= '

Table 4.6 merupakan tabel perbandingan hasil *chipertext* penelitian terdahulu dengan penelitian yang dilakukan pada saat ini dengan menggunakan pesan dan kunci yang sama.

Seperti yang dapat dilihat, hasil *ciphertext* saat ini karakter huruf pesan yang sama tidak menghasilkan karakter huruf *ciphertext* yang sama, berbeda dengan penelitian terdahulu, karakter huruf yang sama menghasilkan karakter *ciphertext* yang sama.

Dapat dilihat bagian penelitian no 3, terdapat 2 karakter huruf pesan “A” yang menghasilkan 2 karakter huruf “H” pada *ciphertext,* sedangkan hasil dari penelitian ini karakter huruf “A” menghasilkan karakter *ciphertext* yang berbeda. Perbedaan ini disebabkan oleh proses enkripnya dibagi menjadi 5 dengan kunci pergeseran yang berbeda – beda.

Untuk lebih menditail, penelitian akan membandingkan kelemahan dan kelebihan algoritma *Affine cipher* yang dimodifikasi pada penelitian saat ini dengan penelitian terdahulu. berikut tabel hasil perbandingan kelemahan dan kelebih algoritma Affine cipher

Table 4.7 Perbandingan kelemahan dan kelebihan *Affine Cipher*

Nomor Penilitian	Kelebihan	Kelemahan
1		1. Hanya bisa melakukan enkripsi dengan 26 karakter (a – z) 2. Hanya 12 kunci invers yang memenuhi invers modular 3. Tidak memuat karakter angka dan simbol *4.* Tidak menghasilkan *chipertext* huruf kecil, 5. Tidak memasukan karakter spasi. sehingga ketika melakukan dekripsi, maka *plaintext*

		menghasilkan kalimat yang menyambung 6. Karakter enkripsi yang sama menghasilkan karakter yangsama 7. Ukuran kunci percobaan kriptanalisis memecahkan yaitu 12 x 26 = 312
2	1. Dapat mengenkripsi 26 karakter (A – Z) dan angka (0 – 9) dengan jumlah 36 karakter yang dapat digunakan 2. Pesan yang di enkripsi dari kalimat belakang	1. Hanya bisa melakukan enkripsi dengan total 36 karakter 2. Hanya memiliki 12 kunci invers yang memenuhi invers modular *3.* Tidak menghasilkan *chipertext* huruf kecil, 4. Tidak memuat karakter simbol 5. Tidak memasukan karakter spasi. sehingga ketika melakukan dekripsi, maka *plaintext* menghasilkan kalimat yang menyambung 6. Karakter enkripsi yang sama menghasilkan karakter yang sama 7. Ukuran kunci percobaan kriptanalisis memecahkan yaitu 12 x 36 = 432
3	1. Dapat mengenkripsi 26 karakter (A – Z), angka (0 – 9), dan 4 karakter simbol tamabahan “ _ ”, “ . ”, “ . ”, “ ‘ ” dengan 40 karakter yang digunakan 2. Mempunya 16 kunci yang memenuhi invers modular 3. Ukuran kunci percobaan kriptanalisis memecahkan yaitu 16 x 40 = 640	1. Hanya bisa mengenkripsi 40 karakter, tidak membuat banyak simbol - simbol 2. Tidak memasukan simbol karakter spasi, sehingga ketika melakukan dekripsi, maka *plaintext* menghasilkan kalimat yang menyambung 3. Karakter enkripsi yang sama menghasilkan karakter yang sama 4. Karakter yang dihasilkan tidak ada huruf kecil
4	1. Memiliki 127 karakter yang dimana terdapat karakter kapital (A – Z), karakter (a – z), angka (0 – 9) serta simbol – simbol. 2. Spasi dihitung menjadi satu karakter, sehingga *plaintext* tetap menghasilkan kalimat yang sama. 3. Memiliki 5 proses perhitungan yang berbeda. 4. Memiliki 126 kunci yang memenuhi invers modular 5. Ukuran kunci percobaan kriptanalisis memecahkan yaitu 126 x (127 x 5) = 80.010	1. Hanya bisa mengenkripsi 127 karakter. Tidak dapat memuat seluruh karakter ASCII 2. Karakter enkripsi yang sama menghasilkan karakter yang sama tergantung posisi urutan dalam prosesnya

Table 4.7 merupakan informasi dari perbandingan Kelebihan dan kelemahan dari penelitian saat ini dengan penelitian terdahulu. Dapat dilihat, informasi tentang ukuran kunci percobaan kriptanalisis dalam memecahkan sandi, pada penelitian ini yang tedapat pada nomor 4, yang sudah di modifikasi memiliki ukuran yang besar untuk kriptanalisis dalam memecahkan sandi, yang dimana harus melalui 80.010 kali percobaan untuk mendapatkan kunci yang tepat.

Perhitungan ini didapat dari teori yang terdapat pada bab 2, yang dimana perhitungannya mengkalikan antara kunci invers dengan kunci pergeseran. Pada penelitian ini membuat 5 proses untuk mendapatkan sandi, yang dimana setiap proses memiliki kunci pergeseran yang berbeda – beda. Kunci invers pada penelitian ini memiliki total 126 kunci, sedangkan kunci pergeseran memiliki 127 kunci dengan 5 proses kunci yang berbeda, jadi perhitungannya 126 x (127 x 5) = 126 x 635 = 80.010.

Hasil ini membuktikan, semakin banyak pemabagian proses dengan kunci pergeseran yang berbeda – beda, semakin kuat *affine cipher* dalam mengamankan pesan tersebut.

4.5.2 Perbandingan Hill Cipher dengan penelitian terdahulu

Table 4.8 Perbandingan *Ciphertext Hill Cipher* dengan penelitian terdahulu

No	Penelitian Terdahulu	Pesan	Ciphertext Penelitian Terdahulu	Kunci Matriks 4 x 4	Ciphertext Penelitian sekarang
1	Hill Cipher 2 x 2 mod 26 dengan maksimal 66 karakter dapat dienkripsi (Sari et al. 2017)	HALO	HLDD	$\begin{bmatrix} 21 & 37 & 43 & 10 \\ 12 & 23 & 9 & 7 \\ 12 & 31 & 20 & 16 \\ 5 & 7 & 8 & 10 \end{bmatrix}$	*'FFUS¥*
2	Hill Cipher 2 x 2 mod 95 (Barmawi and Hamdani 2016)	MILITER	iAi@e@No	$\begin{bmatrix} 21 & 37 & 43 & 10 \\ 12 & 23 & 9 & 7 \\ 12 & 31 & 20 & 16 \\ 5 & 7 & 8 & 10 \end{bmatrix}$	|Öóºo G“ã
3	Hill Cipher 2 x 2 mod 256 (Puspita and Wayahdi 2015)	MAJU	j{‰ Ó	$\begin{bmatrix} 21 & 37 & 43 & 10 \\ 12 & 23 & 9 & 7 \\ 12 & 31 & 20 & 16 \\ 5 & 7 & 8 & 10 \end{bmatrix}$	v`“ê

Pada Table 4.8 merupakan perbandingan hasil *ciphertext* pada penelitian terdahulu dengan sekarang. Terlihat jelas bahwa hasil yang didapat tidak melihatkan perbedaan, dikarenakan hasil dari pesan menghasilkan jumlah karakter kelipatan 4. Perbedaan penelitian sekarang yaitu jika panjang karakter tidak kelipatan 4 maka akan ada penambahan karakter

sampai karakter kelipatan 4 dikarenakan menggunakan Matriks kunci 4 x 4, sedangkan kunci Matriks 2 x 2 hanya menambahkan 1 karakter jika pesan tidak memiliki kelipatan 2.

Contoh penambahan 1 karakter " L " dari penelitian Table 4.8 nomor 1 dengan pesan yang digunakan yaitu " HALLO " yang memiliki 5 karakter, dengan menggunakan kunci Matriks 4 x 4 maka *ciphertext* yang didapat memiliki 8 karakter. Seperti perhitungan sebagai berikut.

$$\begin{bmatrix} 21 & 37 & 43 & 10 \\ 12 & 23 & 9 & 7 \\ 12 & 31 & 20 & 16 \\ 5 & 7 & 8 & 10 \end{bmatrix} \times \begin{bmatrix} 72 & 79 \\ 65 & 46 \\ 76 & 46 \\ 76 & 46 \end{bmatrix} = \square\square \div \S\P\frac{3}{4} \square\square$$

Setelah mengetahui hasil dari hasil kunci matriks 4 x 4, maka penelitian ini akan menjabarkan perbedaan selain hasil dari *chipertext* yang didapat. Yang dimana akan dijabarkan pada tabel sebagai berikut

Table 4.9 Perbandingan Kelebihan dan Kelemahan *Hill Cipher*

No	Algoritma	Kelebihan	Kelemahan
1	Hill Cipher 2 x 2 mod 26	1. Kunci dapat berubah – ubah (dinamis)	1. Karakter harus Huruf Kapital A – Z 2. Tidak memasukan Karakter huruf kecil dan simbol – simbol lain. 3. Tidak memasukan karakter, spasi, sehingga ketika di dekripsi karakter menyambung 4. Maksimal pesan yang di enkripsi maksimal 66 karakter 5. Kunci yang digunakan 2 x 2 6. Kunci Matrik hanya menampung 26 karakter

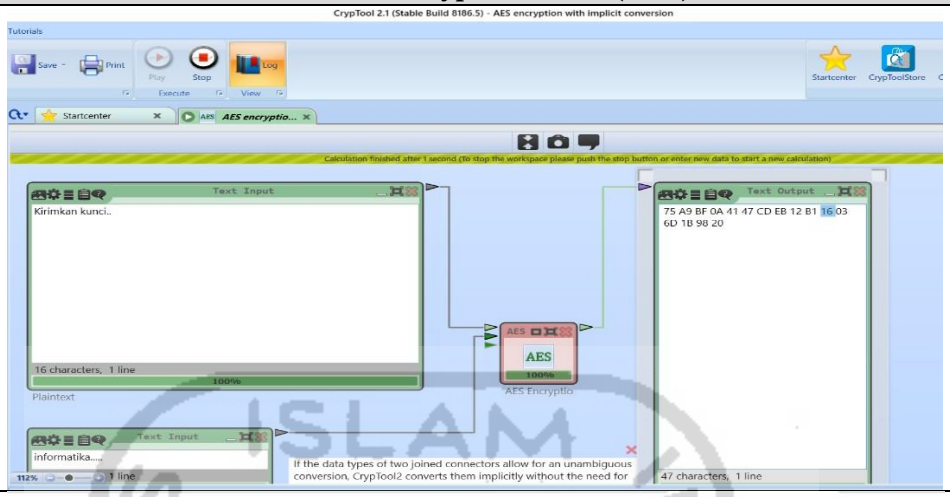
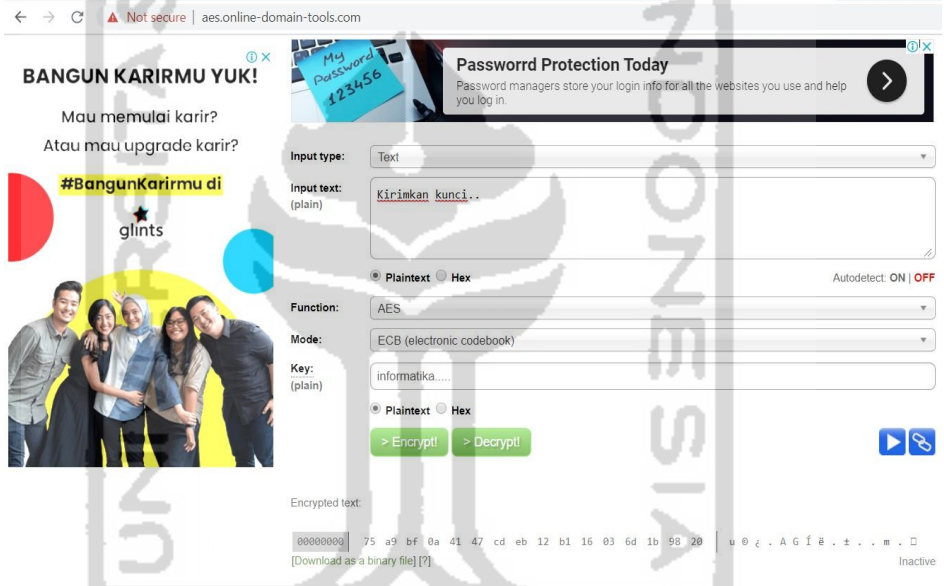
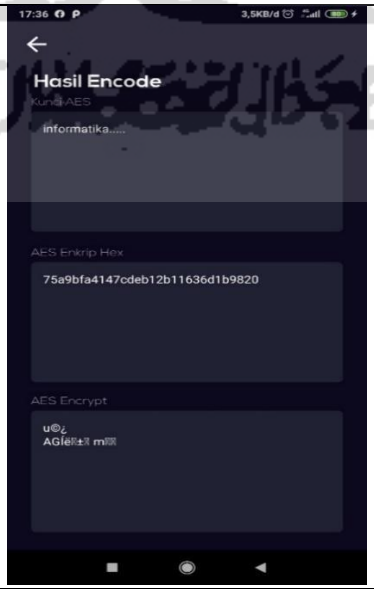
Table 4.9 Perbandingan Kelebihan dan Kelemahan *Hill Cipher* (Lanjutan)

2	Hill Cipher 2 x 2 mod 95	1. Memiliki 95 karakter yang dapat digunakan 2. Karakter spasi terhitung, sehingga proses dekripsi menghasilkan pesan kembali	1. Kunci Statis, tidak dapat dirubah 2. Kunci Matriks hanya menampung 95 karakter 3. Kunci yang digunakan 2 x 2
3	Hill Cipher 2 x 2 mod 256	1. Memiliki 256 karakter yang dapat digunakan 2. Karakter spasi terhitung, sehingga proses dekripsi menghasilkan pesan kembali 3. Kunci bersifat dinamis	1. Kunci yang digunakan 2 x
4	Hill Cipher 4 x 4 mod 256	1. Memiliki 256 karakter yang dapat digunakan 2. Karakter spasi terhitung, sehingga hasil dekripsi menghasilkan pesan kembali 3. Ruang kunci Matriks yang digunakan lebih besar	1. Kunci bersifat statis. dikarena terlalu besar, sehingga mempersulit mencari kunci yang dapat digunakan

4.5.3 Perbandingan kombinasi Caesar Cipher dan AES dengan program lain

Seperti kutipan yang terdapat pada penelitian (Abidin et al. 2016) mengatakan " *Pada Sistem Kriptografi kekuatan dari metode – metode enkripsi adalah pada kunci, sehingga walaupun algoritma metode tersebut tersebar luas orang tidak akan dapat membongkar data tanpa kunci yang tepat* ". Dengan kutipan tersebut maka penelitian ini akan membandingkan program yang dibuat dengan 2 aplikasi enkripsi algoritma AES yang sama, yang dimana aplikasi berbasis desktop dan aplikasi website. Aplikasi desktop yaitu CrypTool sedangkan Website yaitu AES.online domain. Dalam pengujiannya menggunakan pesan dan kunci yang sama. Pesan yang digunakan yaitu " Kirimkan kunci.. " sedangkan kunci yang digunakan yaitu " informatika…..". Berikut tabel hasil pengujiannya.

Table 4.10 Pengujian Algoritma AES

Nama Aplikasi	Hasil Advanced Encrypt Standard (AES) 128 bit
CrypTool2	
AES.online	
ImageMessage	

Dari Table 4.10 terlihat dengan jelas hasil algoritma *Advanced Encrypt Standard (AES)* yang didapat. Dengan kunci yang sama, menghasilkan sebuah *ciphertext* yang sama. Sehingga kekuatan AES terdapat di informasi kunci yang digunakan. Dengan kekurangan itu penelitian mengkombinasikan dengan metode *Caesar Cipher* sehingga ketika informasi pendistribusian kunci bocor, algoritma AES tidak bisa melakukan dekripsi pesan tanpa menggunakan aplikasi yang sama. Maka dari itu, ini merupakan hasil penggabungan enkripsi AES dengan *Caesar Cipher* pada gambar berikut.

Gambar 4.37 Kombinasi Caesar Cipher dan AES-128 bit

Gambar 4.37 merupakan hasil dari kombinasi antar 2 metode enkripsi. Dengan adanya kombinasi ini, kebocoran kunci enkripsi tidak mempengaruhi kekuatan algoritma dari AES, sehingga keamanan AES tetap terjaga dari segi kekuatannya dalam mengamankan data.

4.5.4 Perbandingan waktu proses aplikasi dengan penelitian terdahulu

Dengan berhasilnya proses aplikasi "ImageMessage" melakukan *embedding* dan *extracting*. Selanjutnya yaitu membandingkan waktu proses aplikasi menjalankan program kriptografi dan steganografi dengan penelitian terdahulu, yang dimana penelitian tersebut dilakukan oleh (Anwar 2017).

Dalam melakukan perbandingan waktu proses yang akan dilakukan, maka penelitian ini memasukan pesan, kunci, dan ukuran pixel gambar yang sama dengan penelitian terdahulu, agar mendapatkan informasi perbandingan yang lebih akurat. Berikut data yang digunakan pada penelitian terdahulu yang digambarkan pada tabel.

Table 4.11 Data penelitian terdaulu

No	Pesan	Kunci	Ukuran Pixel
1	gambar ini telah disisipi pesan yang harus di jaga kerahasian.	qwerty! @#$%	300 x 321
2	Abcdefghijklmnopq rstuvwxyz	1234567 890	300 x 321
3	Setelah aplikasi ini melewati proses tahap coding, maka tahap selanjutnya adalah tahap pengujian.	asdfghjk l	420 x 420

Setelah datanya sama, maka membandingkan waktu proses aplikasi melakukan enkripsi dan *embedding* ataupun *extracting* dan dekripsi. Tabel berikut merupakan perbandingan waktu proses aplikasi melakukan *encode*.

Table 4.12 Perbandingan waktu encode

No	AES dan Steganografi	Multiple Kriptografi dan Steganografi
1	21 ms	288 ms
2	17 ms	241 ms
3	24 ms	381 ms

Table 4.12 menujukan perbedaan waktu proses aplikasi melakukan encode, yang dimana terlihat dengan jelas bahwa aplikasi penelitian ini membutuhkan waktu proses lebih dari 200 *millisecond* (ms) untuk mendapatkan hasil akhir *stego image*. Selanjutnya membandingkan waktu proses aplikasi melakukan *decode*.

Table 4.13 Perbandingan waktu decode

No	AES dan Steganografi	Multiple Kriptografi dan Steganografi
1	19 ms	235 ms
2	17 ms	195 ms
3	24 ms	381 ms

Dengan perbandingan yang terdapat di Table 4.12 dan Table 4.13 terlihat dengan jelas, yang dimana aplikasi pada penelitian saat ini. Untuk melakukan proses membutuhkan waktu rata – rata diatas 100 *millisecond* (ms) meskipun isi pesan tidak terlalu banyak.

Dengan ini membuktikan bahwa penambahan algoritma pada salah satu metode menyebabkan penambahan waktu proses untuk menghasilkan keluaran yang semestinya.

4.6 Pemanfaatan Aplikasi

Dari hasil penelitian ini, pemanfaatan metode yang di implementasikan tidak semua orang membutuhkan, dikarenakan tidak semua orang memiliki suatu informasi yang sangat penting. Tetapi beberapa orang individu maupun kelompok membutuhkan keprivasian tersebut untuk menjaga suatu data penting untuk menghindar dari orang yang tidak bertanggung.

Oleh sebab itu, penelitian ini memberikan informasi beberapa bidang mana saja yang cocok dapat menggunakan metode ini, diantara lain sebagai berikut.

a. Militer

   Digunakan untuk menyembunyikan informasi seperti strategi misi penyerbuan, informasi lokasi pengujian senjata baru, latihan dan semestinya, yang dimana notabennya informasi militer membutuhkan sebuah keprivasian dalam penyampaian informasi. Agar musuh tidak mengetahui perkembangan yang dilakukan pihak militer

b. Perusahaan

   Dengan banyak kasus pembobolan sistem pada suatu perusahaan, bisa terjadi yang disebabkan oleh keteledoran dalam penyampaian suatu informasi yang sangat penting , seperti password sistem pusat tanpa mengetahui terjadinya penyadapan alat komunikasi. Sehingga password sistem yang diberikan diketahui oleh orang yang tidak bertanggung jawab. Oleh sebab itu, dengan memanfaatkan metode yang digunakan dapat menghindari dari orang yang tidak bertanggung jawab

c. Kejahatan

   salah satu contohnya digunakan oleh penjual barang illegal dan lain – lain, dengan memanfaatkan metode ini dalam menyampaikan informasi penjualan dari harga, lokasi dan pembayaran dalam berkomunikasi dapat menghindar dari kecurigaan pihak yang tidak berwajib.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pembahasan tugas akhir mengenai Implementasi Multiple Kriptografi dan Steganografi Berbasis android dalam penyembunyian Teks pada citra digital, dapat disimpulkan bahwa

a. Dengan membagi proses menjadi 5 bagian dengan kunci pergeseran yang berbeda – beda dan memperbanyak jumlah karakter yang digunakan, dapat memperkuat hasil *ciphertext* untuk tidak dapat dipecahkan, dikarenakan jumlah ruang percobaan kriptanalisis melakukan perusakan harus melalui 80.010 percobaan.,
b. Dengan menggunakan Kunci Matriks 4 x 4 pada algoritma Hill Cipher, memperkuat hasil *ciphertext* untuk tidak dapat dipecah dikarenakan ruang kunci besar, serta hasil yang didapat memiliki kelipatan 4 karakter, sehingga membuat kriptanalisis harus mengetahui kunci matriks mana yang digunakan antara 2 x 2 atau 4 x 4 dalam memecahkan sandi. Akan tetapi, ruang kunci yang sangat besar tersebut mempersulit dalam mencari kuncinya.
c. Dengan kombinasi antara algoritma *Advanced Encrypt Standard* dan *Caesar Cipher* hasil ciphertext lebih aman, dikarenakan jika informasi kunci keamanan bocor, aplikasi yang menggunakan algoritma yang sama, tidak dapat memecahkan *chipertext* tersebut. Sehingga kombinasi ini memperkuat dalam keamanan data informasi.
d. Dengan menggunakan kombinasi 4 metode untuk mengamankan data pesan, dapat memperkuat dalam mempertahankan keamanan data. Dikarenakan untuk memecah pertahanan tersebut, harus mengetahui algoritma yang digunakan lalu harus mencari kunci yang digunakan pada setiap metodenya, sehingga butuh waktu yang lama untuk kriptanalisis memecahkannya
e. Metode steganografi LSB memang dapat dibuktikan sebagai metode untuk menyembunyikan data ke dalam sebuah objek. Akan tetapi, kekurangan metode steganografi jika objek tersebut rusak atau berubah sedikit saja, dapat merusak data yang ada didalam objek

5.2 Saran

Untuk pengembangan sistem ataupun penelitian lain tentang pengamanan data, maka penelitian ini memberikan sebuah saran yaitu :

a. Algoritma affine cipher yang digunakan saat ini, dapat dimodifikasi lagi dengan cara menambah jumlah proses atau membagi setiap proses dengan kunci invers yang berbeda – beda sehingga menghasilkan raung kunci yang lebih besar lagi
b. Untuk algoritma Hill Cipher, dapat dimodifikasi lagi dengan mencari kunci matriks secara otomatis atau perhitungan matematika yang mempermudah mencari invertible matriks, sehingga kunci matriks dapat digunakan secara dinamis.
c. Untuk penggunaan algoritma AES, dapat dikombinasikan dengan algoritma klasik lain dalam mengamankan kunci, sehingga algoritma AES dapat bertahan dalam mengamankan datanya meskipun algoritmanya tersebar luas.
d. Untuk yang ingin mengembangkan sistem ini, diharapkan menambahkan fiture share file sendiri ataupun menambahkan fiture selain text dalam mengamankan data. Sehingga aplikasi ini dapat digunakan secara maksimal sebagai pengaman data.

DAFTAR PUSTAKA

Ariyus, Dony. 2006. *Kriptografi Kemanan Data Dan Komunikasi*. Yogyakarta: Graha Ilmu.

Sadikin, Rifki. 2012. *Kriptografi Untuk Keamanan Jaringan*. Yogyakarta: ANDI.

Simanjuntak, Rico Afrido. 2019. "Polisi Diminta Responsif Sikapi Kasus Peretasan Medsos Dan WhatsApp." *SINDONEWS*. Retrieved (https://nasional.sindonews.com/read/1393071/13/polisi-diminta-responsif-sikapi-kasus-peretasan-medsos-dan-whatsapp-1554450692).

Kurniawan, Ir Yusuf. 2004. *Kriptografi Keamanan Internet Dan Jaringan Komunikasi*. Bandung: INFORMATIKA.

Totok Sutoyo, Edy Mulyanto, Vincent Suhartono, Oky Dwi Nurhayati, Wijarnato. 2009. *Teori Pengolahan Citra Digital*. Yogyakarta: ANDI.

Munir, Rinaldi. 2004. *Pengolahan Citran Digital Dengan Pendekatan Algoritmik*. Bandung: INFORMATIKA.

Ariyus, Dony. 2009. *Keamanan Multimedia, Konsep Dan Aplikasi*. Yogyakarta: ANDI.

Wibowo, Sasono, Florentina Esti Nilawati, and Suharnawi. 2014. "Implementasi Enkripsi Dekripsi Algoritma Affine Cipher Berbasis Android." *Techno.COM, Vol. 13, No. 4,* 13(4):215–21.

Juliadi, Bayu Prihandono, and Nilamsari Kusumastuti. 2013. "Kriptografi Klasik Dengan Metode Modifikasi Affine Cipher Yang Diperkuatdenganvigenere Cipher." *Buletin Ilmiah Mat. Stat. Dan Terapannya (Bimaster)* 02(2):87–92.

Babu, Sriramoju Ajay. 2017. "Modification Affine Ciphers Algorithm For." *International Journal of Research In Science & Engineering* 3(2):346–51.

Barmawi, Mira Musrini, and Yogi Hamdani. 2016. "Implementasi Hill Chipper Pada Sistem Pengamanan Dokumen." *Seminar Nasional Telekomunikasi Dan Informatika* (Selisik):97–102.

Puspita, and M. Rhifky Wayahdi. 2015. "Analisis Kombinasi Metode Caesar Cipher , Vernam Cipher , Dan Hill Cipher Dalam Proses Kriptografi." *Seminar Nasional Teknologi Informasi Dan Multimedia 2015* (Februari):43–48.

Sari, Jane Irma, Sulindawaty, and Hengki Tamando Sihotang. 2017. "Implementasi Penyembunyian Pesan Pada Citra Digital Dengan Menggabungkan Algoritma Hill Cipher Dan Metode Least Significant Bit (LSB)." *Jurnal Mantik Penusa* 1(2):1–8.

Latifah, Retnani, Sitti Nurbaya Ambo, and Syafitri Indah Kurnia. 2017. "Modifikasi Algoritma Caesar Chiper Dan Rail Fence Untuk Peningkatan Keamanan Teks

Alfanumerik Dan Karakter Khusus." *Seminar Nasional Sains Dan Teknologi* (1-2 November):1–7.

Zuli, Faisal, and Ari Irawan. 2014. "Penerapan Kombinasi Sandi Caesar Dan Vigenere Untuk Pengamanan Data Pesan Pada Surat Elektronik." *Jurnal Sistem Informasi* 7(2):1–11.

Abidin, Ahmad Muazim, Fitria Hardianti, and Indra Nur Setiano. 2016. "Analisa Dan Implementasi Proses Kriptografi Encryption-Decryption Dengan Algoritma Advanced Encryption Standard (Aes-128)." *Jurnal Sarjana Teknik Informatika, Keamanan Komputer* `1-20.

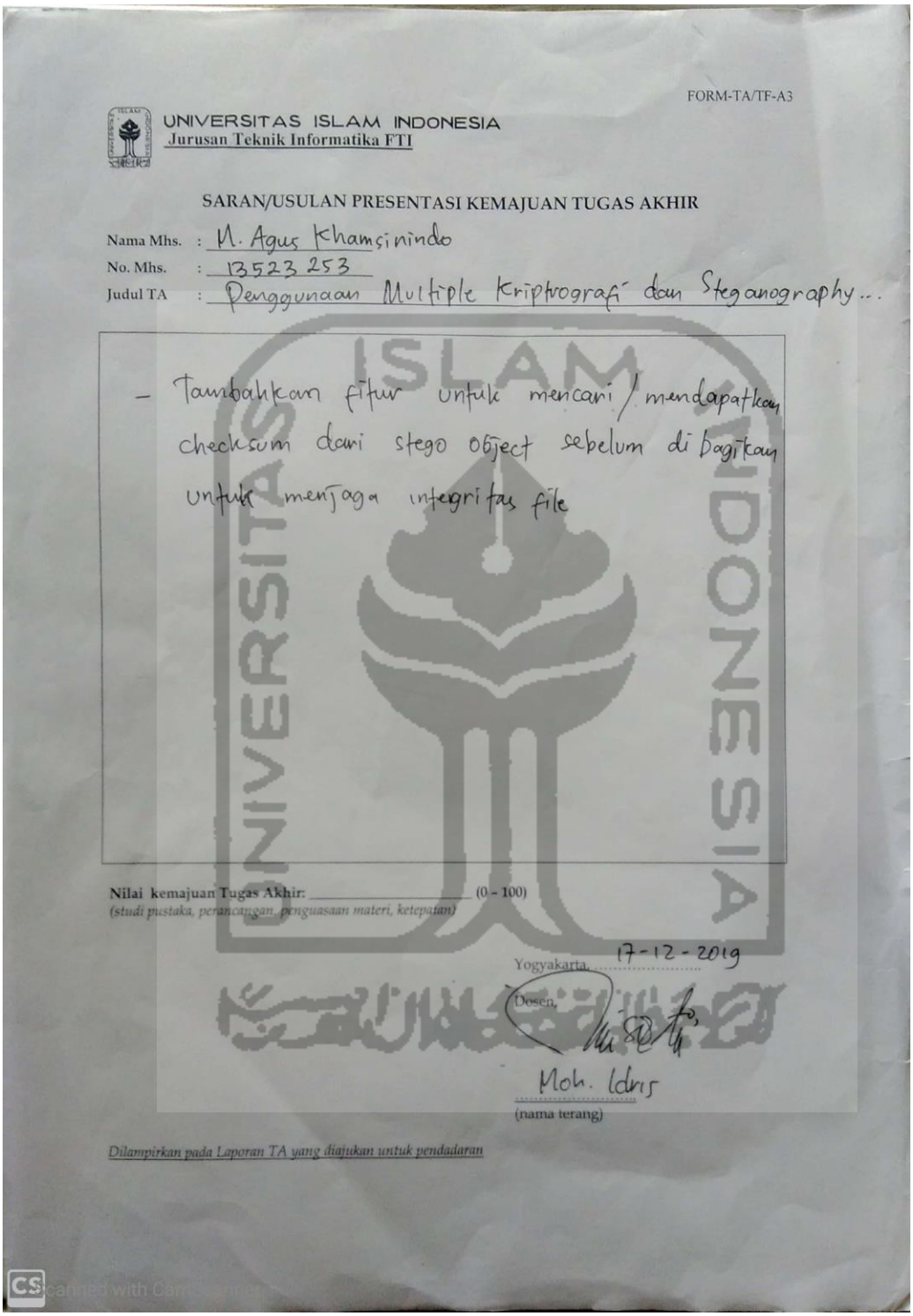
Anwar, Syaiful. 2017. "Implementasi Pengamanan Data Dan Informasi Dengan Metode Steganografi LSB Dan Algoritma Kriptografi AES." *Jurnal* 6:2089–5615.

Maryanto, Budi. 2008. "Penggunaan Fungsi Hash Satu-Arah Untuk Enkripsi Data." *Media Informatika* 7(3):1–10.

Sulianto. 2014. "Perancangan Aplikasi Untuk Memeriksa Keaslian Data Yang Telah Didownload Menggunakan Algoritma Message Digest 5 (Md5)." *Pelita Informatika Budi Darma* 8(3):172–77.

Sukrisno, and Ema Utami. 2007. "Implementasi Steganografi Teknik Eof Dengan Gabungan Enkripsi Rijndael , Shift Cipher Dan Fungsi Hash Md5." *Seminar Nasional Teknologi 2007 (SNT 2007)* 2007(November):1–16.

LAMPIRAN



FORM-TA/TF-A3

UNIVERSITAS ISLAM INDONESIA
Jurusan Teknik Informatika FTI

SARAN/USULAN PRESENTASI KEMAJUAN TUGAS AKHIR

Nama Mhs. : M. Agus Khamsinindo
No. Mhs. : 13523253
Judul TA : Penggunaan Multiple Kriptografi dan Steganography...

- Tambahkan fitur untuk mencari / mendapatkan checksum dari stego object sebelum di bagikan untuk menjaga integritas file

Nilai kemajuan Tugas Akhir: _____ (0 - 100)
(studi pustaka, perancangan, penguasaan materi, ketepatan)

Yogyakarta, 17-12-2019
Dosen,

Moh. Idris
(nama terang)

Dilampirkan pada Laporan TA yang diajukan untuk pendadaran

Rangkuman Kunci Invers Affine Cipher
Yang memiliki Invers Modular 127

Kunci (a)	Kunci Invers (a^{-1})
1	1
2	64
3	85
4	32
5	51
6	106
7	109
0	16
9	113
1 tl	89
11	l@
12	53
13	fi8
l4	118
15	17
I6	8
17	IS
18	120
19	107
20	108
21	121
22	52
23	116
24	90
25	61
26	4d
27	g0
28	59
29	92
30	72
31	41
32	4
33	77
34	71
35	9g
36	60
37	103
38	117
39	114
40	54
41	31
42	124

Kunci (a)	Kunci Invers (a^{-1})
43	65
44	26
45	48
46	5fi
47	100
48	45
49	70
50	94
5 I	5
52	22
53	12
54	40
55	97
56	93
57	78
58	46
59	28
60	**36**
61	25
62	**84**
63	125
64	2
65	4S
66	102
67	91
68	99
69	81
7()	49
71	**34**
72	30
73	87
74	115
75	105
7t	122
77	3
73	57
79	82
80	27
8 I	69
B2	79
83	101
84	62

Kunci (a)	Kunci Invers (a^{-1})
85	**3**
SG	96
S7	73
88	13
89	10
90	**24**
9 I	67
92	29
93	56
94	50
95	123
96	86
07	55
98	35
99	68
100	47
101	53
102	66
103	37
1()4	13
105	75
106	6
107	19
108	20
109	7
110	112
111	**119**
112	110
113	**9**
114	39
115	74
116	23
117	38
11h	14
119	1I I
120	18
121	21
122	76
i 23	95
124	42
123	63
126	126