APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT

SKRIPSI

Disusun Oleh:
MOH. ABDUL HAMID
08.18.152

PROGRAM STUDI TEKNIK INFORMATIKA S-1
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI NASIONAL MALANG
2012

LEMBAR PERSETUJUAN

APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT

SKRIPSI

Disusun dan diajukan untuk melengkapi dan memenuhi persyaratan guna mencapai gelar Sarjana Komputer

Disusun oleh :

Moh. Abdul Hamid
NIM : 08.18.152

Diperiksa dan Disetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Sentot Achmadi, M.Si	**Yosep Agus Pranoto, ST**
NIP.Y. 1039500218	**NIP. P. 1031000432**

Mengetahui,
Ketua Program Studi Teknik Informatika S-1

Joseph Dedy Irawan, ST, MT
NIP. 197404162005011002

PROGRAM STUDI TEKNIK INFORMATIKA S-1
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI NASIONAL MALANG
2012

SURAT PERNYATAAN ORIGINALITAS

Yang bertanda tangan di bawah ini :

Nama : Moh. Abdul Hamid

NIM : 08.18.152

Program Studi : Teknik Informatika S-1

Fakultas : Teknologi Industri

Dengan ini menyatakan bahwa Skripsi yang berjudul "APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT" merupakan hasil karya sendiri, tidak merupakan plagiasi dari karya orang lain. Dalam Skripsi ini tidak memuat karya orang lain, kecuali dicantumkan sumbernya sesuai dengan ketentuan yang berlaku.

Demikian surat pernyataan ini saya buat, dan apabila di kemudian hari ada pelanggaran atas surat pernyataan ini, saya bersedia menerima sangsinya.

Malang, 10 Agustus 2012

Yang membuat Pernyataan,

Moh. Abdul Hamid

NIM. 08.18.152

Teriring Ucapan Terima Kasih kepada

Ayah dan Ibu tercinta

APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT

Moh. Abdul Hamid

Program Studi Teknik Informatika
Fakultas Teknologi Industri, Institut Teknologi Nasional Malang
Jln. Raya Karanglo Km. 2 Malang

Email: moh.abdulhamid@yahoo.co.id

Dosen Pembimbing : **1. Ir. Sentot Achmadi, MSi**
2. Yosep Agus Pranoto, ST

Abstrak

Kerahasiaan dan keamanan data merupakan suatu hal yang menjadi kebutuhan pada saat ini. Seiring dengan hal itu, kriptografi dibutuhkan untuk menangani segala masalah yang menyangkut keamanan informasi. Untuk mengatasi masalah tersebut maka perlu dibuat suatu aplikasi kriptografi yang bisa melakukan proses enkripsi dan dekripsi data dengan menggunakan algoritma kriptografi RSA dan AES. Kriptografi merupakan ilmu dan seni untuk mengamankan pesan dengan menggunakan proses enkripsi dan dekripsi. Enkripsi sendiri merupakan suatu proses merubah data pesan asli menjadi bentuk karakter yang tidak bisa dibaca dengan menggunakan algoritma tertentu, sedangkan dekripsi merupakan suatu proses merubah karakter yang tidak bisa dibaca menjadi data pesan asli.

Saat ini algoritma kriptografi yang sering digunakan adalah algoritma RSA dan algoritma AES. Algoritma RSA merupakan algoritma asimetris yang menggunakan kunci publik untuk melakukan proses enkripsi dan kunci privat untuk melakukan proses dekripsi. Sedangkan algoritma AES merupakan algoritma simetris yang hanya menggunakan satu buah kunci yang sifatnya rahasia.

Aplikasi kriptografi yang dikembangkan ini dapat melakukan proses enkripsi dan dekripsi pada file text, sehingga dokumen ataupun teks yang rahasia dapat diamankan dengan melakukan proses enkripsi, dan mengembalikan data kembali ke asalnya dengan melakukan proses dekirpsi. File output yang dihasilkan dari proses enkripsi algoritma RSA serta algoritma gabungan RSA dan AES berupa text dengan ekstensi (.txt), sedangkan proses enkripsi menggunakan algoritma AES output yang dihasilkan memiliki ekstensi (*txt, *.doc, *.odt). Diharapkan kedepannya aplikasi ini dapat melakukan enkripsi pada file dokumen menggunakan algoritma RSA dan algoritma gabungan RSA dan AES. Serta dapat melakukan enkripsi pada gambar, audio, video, folder serta drive.*

Kata Kunci: *Kriptografi, RSA, AES, File Text*

KATA PENGANTAR

Alhamdulillah, puji syukur kehadirat Allah SWT. karena dengan anugrah yang telah diberikan sehingga penulis berhasil menyelesaikan skripsi ini dengan semaksimal mungkin, walaupun masih jauh dari nilai sempurna. Karena semua yang sempurna hanyalah milik Allah SWT.

Selama penulisan skripsi ini tidak sedikit bantuan yang penulis dapatkan dari berbagai pihak. Oleh karena itu, dalam kesempatan ini dengan segala kerendahan hati, penulis ingin menyampaikan terima kasih yang sebesar-besarnya kepada :

1. Bapak. Ir. Soeparno Djiwo, MT, selaku rektor Institut Teknologi Nasional Malang.
2. Bapak. Ir. Sidik Noertjahjono, MT, selaku dekan Fakultas Teknologi Industri Institut Teknologi Nasional Malang.
3. Bapak. Joseph Dedy Irawan, ST, MT, selaku ketua program studi Teknik Informatika S-1.
4. Bapak. Ir. Sentot Achmadi, MSi, selaku dosen pembimbing I dalam pengerjaan skripsi ini yang telah memberikan bimbingan, kritik, saran, dan kesabaran sehingga skripsi ini terselesaikan.
5. Bapak. Yosep Agus Pranoto, ST, selaku dosen pembimbing II dalam pengerjaan skripsi ini yang telah memberikan bimbingan serta saran dalam proses penyelesaian skripsi ini.
6. Seluruh dosen Teknik Informatika atas semua kesabaran dan keteguhan hati untuk mendidik penulis, sehingga semua ilmu yang diberikan dapat menjadi hal yang berguna bagi penulis untuk menghadapi masa depan.
7. Kedua Orang Tua, yang selalu memberikan doa, materi dan dukungan yang sangat membantu penulis dalam menyelesaikan skripsi ini.

8. Seluruh teman-teman Teknik Informatika ITN Malang pada umumnya yang selalu memberikan dukungan dan bantuan dalam menyelesaikan skripsi ini.
9. Dan semua pihak yang tidak bisa saya sebutkan satu persatu.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna, oleh karena itu kritik dan saran yang sifatnya membangun sangat diperlukan untuk memperbaiki mutu penulisan selanjutnya.

Malang, Agustus 2012

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Masalah keamanan merupakan salah satu aspek terpenting dari suatu sistem informasi. Masalah keamanan sering kali kurang mendapat perhatian dari para perancang suatu program ataupun pengelola sistem informasi. Para perancang program lebih mementingkan bentuk tampilan dari pada faktor kemanan data, sehingga apabila mengganggu performa dari tampilan tersebut maka masalah keamanan tidak begitu dipedulikan bahkan ditiadakan.

Kemajuan sistem informasi memiliki banyak keuntungan tetapi juga memiliki kelemahan seperti pencurian informasi. Misalnya pada sebuah perusahaan, beberapa informasi yang sifatnya rahasia dan hanya bisa diketahui oleh orang-orang tertentu dalam perusahaan tersebut jatuh kepada pihak lawan bisnis sehingga perusahaan mengalami kerugian yang sangat besar. Salah satu hal terpenting dalam pengamanan suatu informasi adalah terjaminnya keamanan dari pesan, data, ataupun informasi dalam proses pertukaran data, sehingga menjadi salah satu pendorong munculnya teknologi kriptografi yang mendukung kebutuhan dari dua aspek keamanan informasi yaitu *secrecy* (perlindungan terhadap kerahasiaan data informasi) dan *authenticity* (perlindungan terhadap pemalsuan dan pengubahan informasi yang tidak diinginkan).

Kriptografi merupakan studi matematika yang mempunyai hubungan dengan aspek keamanan informasi seperti integritas data, keaslian entitas dan keaslian data. Kriptografi menggunakan berbagai macam teknik dalam upaya untuk mengamankan data. Pengiriman data dan penyimpanan data melalui media elektronik memerlukan suatu proses yang dapat menjamin keamanan dan keutuhan dari data yang dikirimkan. Data tersebut harus tetap rahasia selama pengiriman dan harus tetap utuh pada saat data diterima. Untuk memenuhi hal tersebut, perlu dilakukan

proses penyandian (enkripsi dan dekripsi) terhadap data yang akan dikirimkan. Algoritma kriptografi yang baik akan memerlukan waktu yang lama untuk memecahkan data yang telah disandikan.

Sejak perang dunia kedua berakhir, kriptografi dirasa masih dibutuhkan pada kehidupan sehari-hari untuk mengamankan data-data yang bersifat penting dan rahasia. Dengan begitu memicu banyaknya algoritma kriptografi yang bermunculan sesuai dengan perkembangan zaman. Sampai saat ini algoritma kriptografi modern berkembang dengan pesat. Banyak algoritma kriptografi modern yang digunakan untuk melakukan pengamanan suatu data, di antaranya adalah algoritma RSA (dari tiga nama penemunya: Rivest, Shamir, dan Adelman) dan algoritma AES (*Advanced Encryption Standard*). Karena penggunaan satu algoritma pada kriptografi kemungkianan besar bisa dipecahkan kode enkripsinya, maka akan dilakukan penggabungan antara algoritma RSA dan algoritma AES supaya tingkat keamanan yang dihasilkan semakin tinggi.

Algoritma RSA sendiri merupakan algoritma yang termasuk dalam kategori algoritma asimetri atau algoritma kunci publik dimana kunci yang digunakan untuk proses enkripsi dan dekripsi berbeda. Kunci yang digunakan untuk proses enkripsi bersifat tidak rahasia (diketahui oleh publik), sedangkan kunci untuk dekripsi bersifat rahasia (kunci privat). Dengan demikian maka isi dari data yang disandikan hanya dapat dibuka oleh orang yang mempunyai kunci privat. Sedangkan algoritma AES merupakan blok kode simetris untuk menggantikan algoritma DES (*Data Encryption Standard*) yang dianggap memiliki panjang kunci yang terlalu pendek. Agoritma AES sendiri memiliki kunci 128 bit, 192 bit, dan 256 bit sehingga tingkat kemanannya lebih baik. Penyandian AES menggunakan proses berulang yang disebut dengan ronde. Jumlah ronde yang digunakan oleh AES tergantung panjang dari kunci yang digunakan.

Pada proses penyandian, data yang digunakan merupakan data berupa file teks atau data tulisan. File teks sendiri merupakan data yang berisi informasi-informasi dalam bentuk teks atau tulisan. Data yang

berasal dari dokumen pengolah kata, angka yang digunakan dalam perhitungan, nama dan alamat dalam basis data merupakan contoh masukan data teks yang terdiri dari karakter, angka dan tanda baca.

Berdasarkan semua hal yang telah dijabarkan di atas, maka pada skripsi ini algoritma RSA dan algoritma AES digunakan untuk merancang dan membangun aplikasi enkripsi dan dekripsi untuk mengamankan data berupa file teks.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka dapat di ambil rumusan masalah sebagai berikut:

a. Bagaimana melakukan proses enkripsi dan dekripsi data menggunakan algoritma RSA, algoritma AES, serta gabungan antara algoritma RSA dan AES.
b. Bagaimana Menggabungkan algoritma RSA dan algoritma AES dalam kriptografi.

1.3 Batasan Masalah

a. Algoritma kriptografi yang digunakan yaitu algoritma RSA dan algoritma AES.
b. Program aplikasi ini hanya dapat mengenkripsi dan mendekripsi data yang berupa teks atau tulisan, bukan suara maupun gambar.
c. Aplikasi ini dibuat dengan menggunakan bahasa pemrograman Borland Delphi 7.
d. Jenis file yang dapat dienkripsi menggunakan algoritma AES berupa (*.txt, *.docx, *.doc, *odt).
e. Jenis file yang dapat dienkripsi menggunakan algoritma RSA hanya (*.txt).
f. Jenis file yang dapat di enkripsi menggunakan algoritma gabungan RSA dan AES hanya berupa (*.txt).

1.4 Tujuan

Aplikasi kriptografi ini memiliki tujuan untuk mengimplementasikan algoritma RSA dan algoritma AES untuk proses enkripsi dan dekripsi pada file teks.

1.5 Manfaat Penelitian

Adapun manfaat dari penelitian ini adalah:

1. Penelitian ini bermanfaat untuk memberikan keamanan data pada setiap pengguna dan memudahkan para pengguna dalam mengirimkan suatu data atau pesan rahasia ke tempat tujuan tanpa dapat diketahui isi dari data atau pesan oleh pihak yang tidak berkepentingan.
2. Sebagai sarana pengembangan keilmuan teknik informatika dalam bidang kriptografi (Penyandian) dan kemampuan dalam membangun suatu aplikasi kriptografi enkripsi dan dekripsi dengan mengimplementasikan algoritma RSA dan algoritma AES.

1.6 Metodologi Penelitian

Metode yang digunakan pada penelitain ini adalah:

a. Studi Literatur

   Melalui studi ini penulis memperolch data atau informasi dengan cara mengumpulkan, mempelajari, dan membaca berbagai referensi baik itu dari buku-buku, jurnal, makalah, internet dan berbagai sumber lainnya yang menunjang dalam penulisan skripsi ini.

b. Analisis Data

   - Menganalisa algoritma kriptografi RSA, serta teknik-teknik yang digunakan untuk proses enkripsi dan dekripsi.
   - Menganalisa algoritma kriptografi AES, serta teknik-teknik yang digunakan untuk proses enkripsi dan dekripsi.

c. Perancangan Sistem

Merancang sistem aplikasi yang mengimplementasikan algoritma RSA dan algoritma AES dengan menggunakan bahasa pemrograman Borland Delphi 7.

d. Pengujian Sistem

Melakukan pengujian terhadap aplikasi yang dirancang.

1.7 Sistematika Penulisan

Sistematika penyusunan laporan praktek kerja nyata ini dilakukan secara berurutan pada tiap-tiap bab dari awal hingga akhir. Model penyusunan laporan skripsi sebagai berikut :

1. BAB I PENDAHULUAN

Bab pendahuluan terdiri dari : Latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penyusunan skripsi.

2. BAB II LANDASAN TEORI

Berisi pembahasan tentang pengertian dari teori-teori yang dipakai sebagai referensi. Bahan pustaka yang digunakan diperoleh dari berbagai sumber seperti : Jurnal penelitian laporan penelitian, buku, maupun temuan-temuan hasil *browsing* di internet.

3. BAB III ANALISA DAN PERANCANGAN

Berisi analisa dan desain aplikasi secara terstruktur, yang dilengkapi dengan flowchart program.

4. BAB IV IMPLEMENTASI DAN PENGUJIAN

Membahas tentang implementasi yang dibuat secara keseluruhan, serta melakukan pengujian terhadap aplikasi yang dibuat untuk mengetahuai apakah aplikasi tersebut telah berjalan sesuai dengan yang diharapkan.

5. BAB V PENUTUP

Penutup berisi kesimpulan dan saran dari hasil penelitian. Kesimpulan merupakan pernyataan singkat yang dijabarkan dari hasil penelitian dan pembahasan, untuk membuktikan kebenaran dari temuan pustaka yang diperoleh sekaligus menjawab tujuan penelitian. Sedangkan saran adalah rekomendasi untuk penelitian selanjutnya, yang didasarkan atas pengalaman dan pertimbangan dari hasil penelitian yang telah dilakukan.

BAB II
LANDASAN TEORI

2.1 Kriptografi

Kriptografi merupakan seni dan ilmu menyembunyikan informasi dari penerima yang tidak berhak. Kata kriptografi berasal dari bahasa yunani *kryptos* (tersembunyi) dan *graphein* (menulis). Jadi kriptografi adalah ilmu yang mempelajari tentang bagaimana menjaga kerahasiaan suatu pesan, agar pesan yang disampaikan tersebut aman sampai ke penerima pesan.

Konsep kriptografi sendiri telah lama digunakan oleh manusia misalnya pada peradaban mesir dan romawi meskipun masih sangat sederhana. Berbeda dengan kriptografi klasik yang menitikberatkan kekuatan pada kerahasiaan algoritma yang digunakan, kriptografi modern lebih menitikberatkan pada kerahasiaan kunci yang digunakan pada algoritma tersebut sehingga algoritma tersebut dapat saja disebarkan kepada kalangan masyarakat tanpa takut kehilangan kerahasiaan bagi para pemakainya.

Pada abad ke-20, kriptografi lebih banyak digunakan oleh kalangan militer. Pada perang dunia ke II, pemerintah Nazi jerman membuat mesin enkripsi yang diberi nama *enigma*. Mesin ini menggunakan beberapa buah rotor (roda berputar), dan melakukan proses enkripsi yang sangat rumit. Jerman percaya pesan yang dikirim melalui *enigma* tidak akan terpecahkan kode enkripsinya. Tetapi anggapan jerman tersebut salah, setelah mempelajari mesin *enigma* bertahun-tahun, sekutu berhasil memecahkan kode-kode tersebut. Setelah jerman mengetahui kode-kode mereka telah terpecahkan, kemudian *enigma* mengalami beberapa kali perubahan.

Mesin *enigma* yang digunakan di Jerman bisa mengenkripsi satu pesan dengan 15 milyar kemungkinan. *Enigma* termasuk kriptografi berbasis rotor. Mesin berbasis rotor ini dibangun dan dipatenkan oleh beberapa orang penemu dari negara-negara yang berbeda dari tahun 1917

sampai 1921, di antaranya oleh warga Amerika yang bernama Edward Hug hebern, warga Jerman Arthur Scherbius, warga Belanda Alexander Koch dan warga Swedia Arvid Gerhard Damm. Milik Koch dikembangkan oleh Arthur Scherbius yang dipatenkan diberi nama *enigma*. Pada tahun 1930, *enigma* untuk versi militer dibangun. Diperkirakan mesin *enigma* yang digunakan pada tahun 1935 sampai 1945 mencapai 100.000 mesin.

Perkembangan peralatan computer digitalmemicu terbentuknya kriptografi modern. Dengan menggunakan computer digital akan lebih memungkinkan menghasilkan cipher lebih banyak dan rumit. Kriptografi klasik pada umumnya dienkripsi menggunakan alphabet tradisional (karakter per karakter), sedangkan kriptografi modern beroperasi pada string biner. Cipher yang lebih banyak seperti halnya AES (*Advanced Encryption Standard*) dan RSA adalah algoritma modern yang sangat terkenal di dunia kriptografi. Kriptografi modern tidak hanya berkaitan dengan teknik menjaga kerahasiaan pesan, tetapi juga menghasilkan tanda tangan digital dan sertifikat digital. Dengan kata lain kriptografi modern tidak hanya memberikan aspek keamanan, tapi juga aspek-aspek yang lain.

Kriptografi mempunyai komponen-komponen unutk mencapai tujuan kriptografi. Komponen-komponen tersebut antara lain:

a. *Plainteks*: merupakan suatu pesan bermakna yang ditulis atau diketik dalam format biasa yang kemudian diproses menggunakan algoritma kriptografi untuk menjadi *cipherteks*.
b. *Cipherteks*: merupakan suatu pesan dalam bentuk karakter-karakter yang tidak terbaca karena sudah melalui proses enkripsi menggunakan algoritma kriptografi.
c. Kunci: kunci yang dimaksu disini adalah kunci yang digunakan untuk melakukan proses enkripsi dan dekripsi, sehingga kunci yang digunakan harus sama apabila ingin mengetahui isi dari pesan asli. Kunci dibagi menjadi dua bagian, yaitu kunci publik (*public key*) dan kunci rahasia (*private key*).

d. Enkripsi: merupakan suatu proses merubah data pesan asli (*plainteks*) menjadi bentuk pesan karakter yang tidak bisa dibaca dengan menggunakan algoritma kriptografi tertentu.
e. Dekripsi: merupakan kebalikan dari enkripsi, yaitu pesan yang berbentuk karakter-karakter yang tidak bisa dibaca (*cipherteks*) dikembalikan ke bentuk asalnya (*plainteks*) sehingga isi pesan dapat dibaca. Algoritma yang digunakan untuk dekripsi tentu berbeda dengan yang digunakan untuk enkripsi.
f. Pesan: pesan bisa berupa data atau informasi yangdikirim melalui kurir,saluran komunikasi data, atau yang disimpan di dalam media perekaman (kertas, *storage*, dan sebagainya).
g. *Cryptanalys*: bisa juga diartikan sebagai analisis sandi atau suatu ilmu untuk dapat mengetahui *plainteks* tanpa harus mengetahui kunci secara wajar. Jika suatu *cipherteks* berhasil menjadi *plainteks* tanpa menggunakan kunci yang sah, maka proses tersebut dinamakan *breaking code* yang dilakukan oleh para *cryptanalyst*. Analisis sandi juga dapat menemukan kelemahan dari suatu algoritma kriptografi dan akhirnya bisa menemukan kunci dari *cipherteks* yang dienkripsi menggunakan algoritma tertentu.

Secara umum ada dua tipe algoritma kriptografi berdasarkan kuncinya yaitu algoritma simetris dan algoritma asimetris. Algoritma simetris adalah algoritma yang memiliki kunci enkripsi dan dekripsi yang sama, sedangkan algoritma asimetris terdiri atas dua buah kunci yaitu kunci publik untuk melakukan enkripsi dan kunci privat untuk melakukan dekripsi.

Algoritma kriptografi yang baik tidak ditentukan oleh kerumitan dalam mengolah data atau pesan yang akan disampaikan tetapi harus memenuhi 4 persyaratan berikut:

a. Kerahasiaan. Pesan hanya bisa dibaca oleh pihak yang memiliki kewenangan.
b. Autentikasi. Pengirim pesan harus dapat diidentifikasi dengan pasti.

c. Integritas. Penerima pesan harus dapat memastikan bahwa pesan yang diaterima tidak dimodifikasi ketika sedang dalam proses transmisi data.
d. *Non-Repudiation*. Pengirim pesan harus tidak bisa menyangkal pesan yang dia kirimkan.

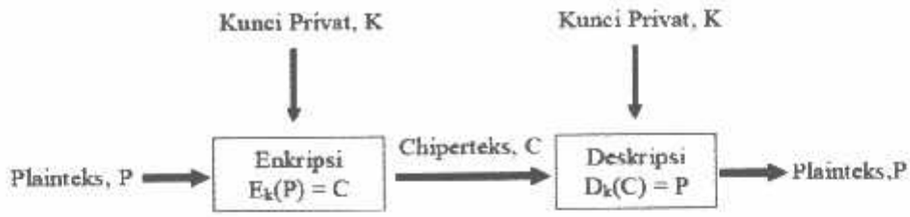
Gambar 2.1 Urutan proses kriptografi

2.2 Algoritma Simetris

Algoritma simetris merupakan algoritma kriptografi dimana kunci yang digunakan untuk proses enkripsi dan dekripsi sama. Dalam kriptografi algoritma simetris dapat diasumsikan bahwa penerima dan pengirim pesan telah terlebih dahulu berbagi kunci sebelum pesan dikirimkan.

Semua kriptografi klasik menggunakan algoritma simetris. Sebelum tahun 1976 hanya algoritma simetris inilah yang dikenal. Kriptografi modern juga ada yang termasuk ke dalam lingkup algoritma simetris, yaitu AES (*Advanced Encryption Standard*).

Algoritma simetris dapat dibagi menjadi dua, yaitu penyandian blok (*Block Cipher*) dan penyandian alir (*Stream Cipher*). Penyandian blok bekerja pada suatu data yang terkelompok menjadi blok-blok atau kelompok data dengan panjang yang sudah ditentukan. Pada penyandian blok, data yang masuk akan dipecah menjadi beberapa blok data yang sudah ditentukan ukurannya. Sedangkan penyandian alir bekerja pada suatu data bit tunggal atau kadang dalam satu *byte*, sehingga format data yang mengalami proses enkripsi dan dekripsi berupa aliran bit data.



Gambar 2.2 Skema algoritma simetris

Pada gambar 2.2 menjelaskan sebuah *plainteks* dienkripsi menggunakan seuatu kunci enkripsi sehingga menjadi *cipherteks*, dan setelah itu dari bentuk *cipherteks* didekripsi menggunakan kunci yang sama pada saat dilakukan enkripsi sehingga menjadi *plainteks* kembali.

Tingkat keamanan dari algoritma simetris sangat ditentukan oleh kerahasiaan kunci yang digunakan. Jika seseorang hendak mengirimkan suatu pesan kepada orang lain, orang tersebut harus terlebih dahulu memberitahu kepada pihak yang dituju. Untuk itu jalur komunikasi harus benar-benar aman dari penyadapan agar kunci tidak diketahui oleh orng lain yang tidak berhak.

Kelebihan algoritma simetris adalah:

1. Proses enkripsi ataupun dekripsi membutuhkan waktu yang sangat singkat.
2. Ukuran kunci simetris pendek.
3. Otentikasi pengiriman pesan langsung diketahui dari chipherteks yang diterima, karena kunci hanya diketahuai oleh pengirim dan penerima pesan saja.

Kekurangan algoritma simetris adalah:

1. Kunci simetris harus dikirim melalui saluran komunikasi yang aman, dan kedua orang yang berkomunikasi harus menjaga kerahasiaan kunci.
2. Kunci harus sering diubah.

2.2.1 Stream Cipher

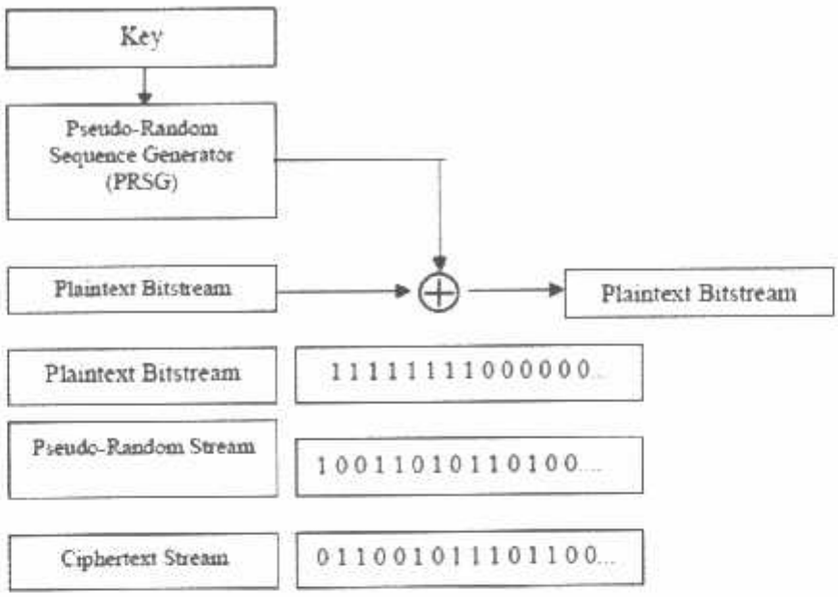
Stream Cipher atau aliran cipher merupakan suatu cipher yang berasal dari hasil XOR setiap bit *plainteks* dengan setiap bit *key*. *Key* merupakan kunci utama (kunci induk) yang digunakan untuk membankitkan kunci acak semu yang dibangkitkan dengan *Pseudo Random Sequence Generator* yang merupakan suatu nilai yang nampak seperti diacak, tetapi sesungguhnya nilai tersebut merupakan suatu urutan. Secara khusus urutan dari nilai yang dihasilkan oleh RNG (*Random Number Generator*), *computational* mekanisme *deterministic* atau FSM (*Finite State Machine*) merupakan kebalikan dari *Really Random*.

Salah satu pembangkit bilangan acak yang cocok untuk kriptografi dinamakan *Cryptographically Secure Pseudorandom Generator* (CSPRNG).

Persyaratan dari CSPRNG adalah:

1. Secara statistik harus mempunyai sifat-sifat yang bagus, yaitu lolos uji keacakan statistik.
2. Tahan terhadap serangan (*attack*) yang serius. Serangan ini bertujuan untuk memprediksi bilangan acak yang dihasilkan.

Random Number Generator secara umum adalah *Pseudo Random*. Yang memberikan *initial state* atau *seed* (nilai yang di-*input* ke dalam *state*), seluruh urutan tersebut ditentukan secara keseluruhan, tetapi meskipun demikian banyaknya karakteristik yang ditampilkan dari suatu urutan yang diacak tersebut. *Pseudorandomness* menghasilkan urutan yang sama secara berulang-ulang pada penempatan yang berbeda. Kemudian kunci acak tersebut diberikan operasi XOR dengan *plainteks* untuk mendapatkan *cipherteks*.



Gambar 2.3 Proses Stream Cipher

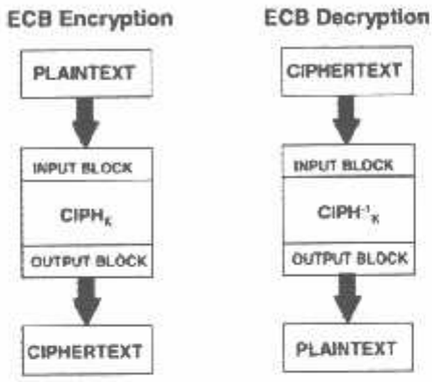
2.2.2 Block Cipher

Block cipher merupakan suatu algoritma dimana input dan outputnya berupa satu blok dan setiap blok terdiri dari 64 bit atau 128 bit. Masukan dari *plainteks* biasanya dibagi menjadi beberapa blok, misalnya 64 bit setiap blok, apabila masukan kurang dari jumlah tersebut maka akan dilakukan penambahan bit (*padding*) sehingga menjadi 64 bit. Block cipher terbagi menjadi empat mode operasi yaitu, mode *electronic Code Book* (ECB), mode *Cipher Block Chaining* (CBC), mode *Cipher Feed Back* (CFB), dan mode *Output Feed Back* (OFB).

2.2.2.1 *Electronic Code Book* (ECB)

Ariyus (2008:97). Pada mode *Electronic Code Book* ini suatu blok kode yang panjang dibagi dalam bentuk urutan binary menjadi satu blok tanpa mempengaruhi blok-blok lain. Satu blok terdiri dari 64 bit atau 128 bit. Setiap blok merupakan bgian dari pesan yang dienkripsi. Kata *code book* di dalam ECB muncul dari fakta bahwa blok teks asli yang sama selalu dienkripsi menjadi blok teks kode yang sama, maka secara teoritis

dimungkinkan untuk membuat buku teks asli dan teks kode yang berkorespondensi. Namun semakin besar ukuran blok, semakin besar pula ukuran buku kodenya. Misalkan jika blok berukuran 64 bit, buku kode terdiri dari 2^{64}-1 buah kode (*entry*), yang berarti terlalu besar untuk disimpan. Lagipula setiap kunci mempunyai buku kode yang berbeda.



Gambar 2.4 Mode operasi ECB

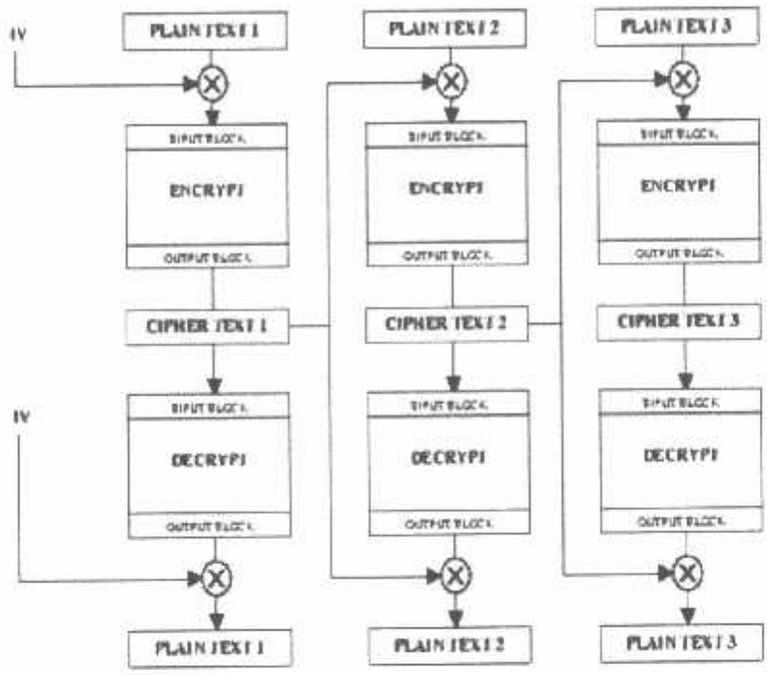
Keuntungan dari mode ECB adalah kemudahan dalam implementasi dan pengurangan risiko salahnya semua *plainteks* akibat kesalahan dari satu *plainteks*. Namun mode ini memiliki kelemahan pada aspek keamanannya. Dengan mengetahui pasangan *plainteks* dan *cipherteks*, seorang kriptanalis dapat menyusun suatu *code book* tanpa perlu mengetahui kuncinya.

2.2.2.2 *Cipher Block Chaining* (CBC)

Ariyus (2008:101). Sistem dari mode *cipher block chaining* adalah *plainteks* yang sama akan dienkripsi ke dalam bentuk kode yang berbeda, hal ini disebabkan karena blo kode yang satu tidak berhubungan dengan blok kode yang lain, melainkan tergantung pada kode sebelumnya.

Pada mode CBC, masukan untuk enkripsi merupakan hasil dari XOR antara *plainteks* dengan teks kode sebelumnya. Kunci digunakan pada setiap blok yang ada. Pola dari *plainteks* pada mode ini selalu berulang dan ditutupi oleh operasi XOR sehingga penyerang lebih sulit

menganalisis kemungkinan teks aslinya. Namun mode operasi CBC memiliki kekurangan yaitu apabila terjadi kesalahan satu bit saja pada salah satu blok, maka akan terjadi kesalahan pada blok-blok berikutnya.



Gambar 2.5 Mode operasi CBC

Blok kode pertama dari enkripsi merupakan IV (*Initial Value*) yang digunakan untuk mengganti kode sebelumnya. Begitu juga dengan dekripsinya, IV dilakukan dengan operasi XOR dengan keluaran algoritma dekripsi untuk memperoleh blok *plainteks* yang pertama. *Initial value* diketahui oleh pengirim dan penerima pesan. Untuk mendapatkan kemanan yang maksimum, IV hrus dilindungi sebagaimana melindungi kunci.

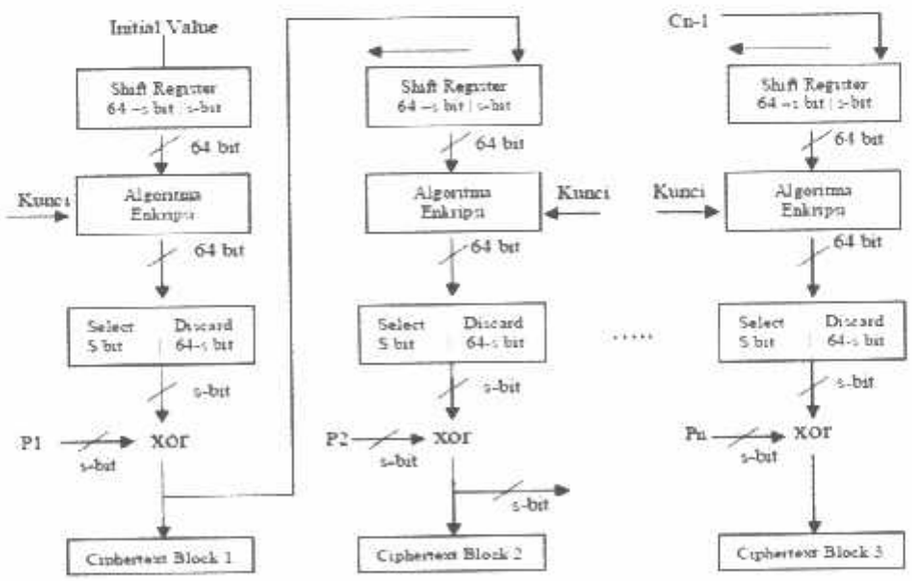
2.2.2.3 *Cipher Feed Back* (CFB)

Mode operasi ini digunakan untuk melakukan enkripsi pada *stream chipper* (aliran kode), mode operasi ini tidak memerlukan tambahan bit karena jumlah panjang blok sama dengan jumlah panjang *plainteks*. Mode operasi ini bekerja pada sistem real time.

Pada mode CBC, proses enkripsi atau dekripsi tidak dapat dilakukan sebelum blok data yang diterima lengkap terlebih dahulu. Masalah tersebut dapat di atasi menggunakan mode CFB. Pada mode CFB, data dapat dienkripsi pada unit-unit yang lebih kecil atau sama dengan ukuran blok.

Mode operasi CFB ini memeliki masukan 8 bit yang diproses setiap enkripsi dan teks kode sebelumnya digunakan sebagai masukan dari algoritma enkripsi untuk menghasilkan keluaran yang acak. Output diambil dari 8 bit paling kiri untuk kemudian dilakukan operasi XOR pada *plainteks* dengan panjang 8 bit sehingga menghasilkan teks kode berikutnya. Input dari enkripsi terdiri dari 8 bit yang digeser ke kiri sebanyak 8 bit. Karena terjadi penggeseran maka kekosongan yang ada akan diisi oleh kode sebelumnya. Input dari enkripsi pada awalnya adalah IV (*initial value*). Jika panjang IV pada register geser 64 bit maka keluaran enkripsinya akan memiliki panjang yang sama yaitu 64 bit.

Satu hal yang tidak menguntungkan dari mode ini adalah jika satu blok kode mengalami kesalahan maka kesalahan itu juga untuk semua blok yang lain, karena antara blok yang satu dengan blok yang lain saling berhubungan.

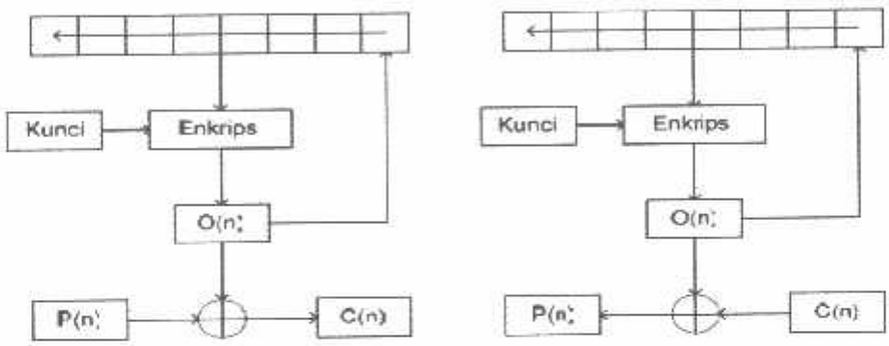


Gambar 2.6 Mode operasi CFB

Pada gambar di atas, s-bit merupakan pergeseran 1 bit ke kiri dari nilai yang sebenarnya. Hal ini untuk mendapatkan urutan bit yang tidak mudah dilacak oleh penyerang.

2.2.2.4 *Output Feed Back* (OFB)

Mode *Output Feed Back* (OFB) tidak mempengaruhi blok yang lai jika terjadi error. Satu bit yang error pada teks kode hanya akan mempengaruhi satu bit *plainteks* pada proses dekripsi. Hal ini sangat berguna untuk sistem analog seperti voice atau video. Jika satu bit yang error tidak merusak semua blok yang ada, tetapi jika rusak semua maka akan mempengaruhi arti dari *plainteks* yang ada.

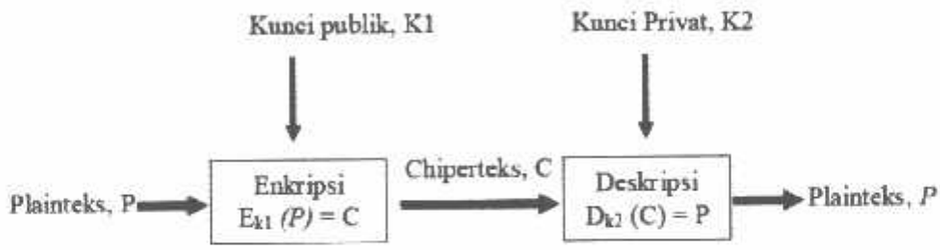


Gambar 2.7 Mode operasi OFB

Sama seperti pada mode CFB, mode OFB juga memerlukan suatu register geser dalam pengoperasiannya. Pertama kali, IV akan masuk ke dalam register geser dan dilakukan enkripsi terhadap IV tersebut. Dari hasil proses enkripsi tersebut akan di ambil 8 bit paling kiri untuk dilakukan XOR dengan *plainteks* yang nantinya akan menjadi teks kode. Teks kode tidak akan di umpan balik ke dalam register geser, tetapi yang akan di umpan balik adalah hasil dari enkripsi IV.

2.3 Algoritma Asimetris

Algoritma asimetris merupakan suatu algoritma kriptografi dimana kunci yang digunakan untuk proses enkripsi berbeda dengan kunci yang digunakan untuk proses dekripsi. Kunci enkripsi dinamakan sebagi kunci publik yaitu kunci yang disebarluaskan atau diketahuai oleh banyak orang, sedangkan kunci dekripsi dinamakan kunci privat atau kunci rahasia yaitu kunci yang hanya diketahui oleh penerima pesan. Contoh dari algoritma asimetris adalah RSA.



Gambar 2.8 Skema standar kunci asimetri

Pada gambar 2.3 menjelaskan sebuah palinteks dienkripsi menggunakan kunci publik (K1) sehingga menjadi *cipherteks*, dan setelah itu dari bentuk *cipherteks* didekripsi menggunakan kunci privat (K2) artinya kunci yang digunakan untuk proses dekripsi berbeda dengan kunci yang digunakan pada waktu proses enkripsi sehingga menghasilkan *plainteks*.

Algoritma asimetris ini dapat di analogikan seperti kotak surat yang terkunci dan memiliki lubang untuk memasukkan surat. Setiap orang dapat memasukkan surat tersebut ke dalamnya tetapi hanya tetapi hanya pemilik surat yang mempunyai kunci dan yang dapat membuka kotak surat tersebut. Kunci publik dapat dikirimkan malalui jalur mana saja tanpa harus takut, karena meskipun kunci tersebut diketahui oleh pihak yang tidak berkepentingan, surat tersebut tidak dapat didekripsikan. Untuk mendekripsikan surat harus menggunakan kunci privat.

Kelebihan algoritma asimetris adalah:

1. Hanya kunci privat yang perlu dijaga kerahasiaannya
2. Pasangan kunci publik dan kunci privat diubah dalam jangka waktu yang lama.
3. Dapat digunakan dalam pengaman kunci simetris.
4. Beberapa algoritma kunci publik dapat digunakan untuk member tanda tangan digital pada pesan.

Kekurangan algoritma asimetris adalah:

1. Proses enkripsi dan dekripsi umumnya lebih lambat dari algoritma simetris karena menggunakan bilangan yang besar dan operasi bilangan yang besar pula.
2. ukuran kunci lebih besar dari kunci simetris.

2.4 Algoritma Kriptografi RSA

Algoritma kriptografi RSA dijabarkan pada tahun 1977 oleh tiga orang yaitu Ron Rivest, Adi Shamir, dan Lan Adleman dari Massachusetts Institute of Technology. Algoritma tersebut dipatenkan oleh Massachusetts Institute of Technology pada tahun1983 di Amerika Serikat sebagai *U.S. Patent 4405829*. Paten tersebut berlaku hingga 21 September 2000. Semenjak algoritma RSA dipublikasikan sebagai aplikasi paten, regulasi di negara-negara lain tidak memungkinkan penggunaan paten.

RSA merupakan salah satu algoritma penyandian yang paling banyak mengundang kontroversi. Sejauh ini belum ada yang menemukan lubang *security* pada RSA, tetapi tak seorang pun juga yang berhasil memberikan pembuktian ilmiah yang memuaskan dari keamanan teknik sandi ini.

Algoritma kriptografi RSA merupakan algoritma yang termasuk dalam kategori algoritma asimetri (algoritma kunci publik), di desain sedemikian rupa sehingga kunci yang digunakan untuk enkripsi berbeda dari kunci yang digunakan untuk dekripsi. Algoritma ini melakukan pemfaktoran bilangan yang cukup besar. Oleh karena alasan tersebut algoritma RSA dianggap aman. Algoritma RSA mengekspresikan teks asli

yang dienkripsi menjadi blok-blok yang mana setiap blok memiliki nilai bilangan biner yang diberi simbol "n", blok teks asli "M" dan blok teks kode "C". untuk melakukan enkripsi pesan "M", pesan dibagi ke dalam blok-blok numerik yang lebih kecil daripada "n" (data biner dengan pangkat terbesar). Jika bilangan prima yang panjangnya 200 digit, dapat ditambah beberapa bit 0 dikiri bilangan untuk menjaga agar pesan tetap kurang dari nilai "n".

Keamanan algoritma RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan non-prima menjadi faktor primanya, yang dalam hal ini $r = p \times q$. penemu algoritma RSA menyarankan agar panjang nilai p dan q lebih dari 100 digit. Dengan demikian hasil kali $r = p \times q$ akan lebih dari 200 digit. Selain itu kemanan algoritma RSA tergantung pada ukuran kunci sandi tersebut (dalam ukuran bit), karena makin besar ukuran kunci, maka makin besar pula kemungkinan kombinasi kunci yang harus dijebol dengan mencoba satu persatu semua kemungkinan kunci atau dengan istilah *brute-force attack*. Jika dibuat suatu sandi RSA dengan panjang 256-bit, maka metode *brute-force attack* akan menjadi sia-sia dan tidak akan sanggup untuk dijebol.

Fakta inilah yang membuat algoritma RSA tetap dipakai hingga saat ini. Selagi belum ditemukan algoritma yang bagus untuk memfaktorkan bilangan bulat menjadi faktor primanya, maka algoritma RSA tetap direkomendasikan untuk mengenkripsi pesan.

2.4.1 Konsep Dasar RSA

Fungsi penggunaan kriptografi kunci publik terbagi atas dua bagian yang berjalan paralel, yaitu untuk konfidensialitas dan untuk otentifikasi identitas. Untuk fungsi konfidensialitas, pengirim melakukan enkripsi pesan menggunakan kunci publik (kunci yang diketahui oleh umum) milik penerima, kemudian penerima mendekripsi pesan menggunakan kunci privat (kunci yang hanya diketahui oleh penerima pesan). Sedangkan untuk fungsi otentifikasi identitas, pengirim mengenkripsi menggunakan

kunci privat, kemudian penerima pesan mendekripsi pesan menggunakan kunci publik. Jika kunci publik dan kunci privat yang dimiliki oleh pengirim dan penerima cocok kombinasinya, maka pesan-pesan yang telah dienkripsi dapat dikembalikan ke pesan asli.

2.4.2 Pembangkit Kunci RSA

Untuk menggunakan algoritma RSA terlebih dahulu pendekripsi membangkitkan sepasang kunci, yaitu kunci publik dan kunci privat. Hal pertama yang dilakukan untuk membangkitkan kunci adalah menentukan 2 bilangan prima besar. Pembangkitan bilangan prima besar menggunakan algoritma pengujian bilangan prima, misalnya algoritma Miller-Rabin. Agar sistem kriptografi RSA aman, maka diperlukan bilangan prima yang besar sehingga sulit untuk difaktorisasi.

Fungsi dari pembangkit kunci adalah menerima masukan panjang bit n dan mengembalikan sebuah struktur *linked list* yang berisi 2 objek yaitu kunci publik dan kunci privat RSA.

2.4.3 Konsep Dasar Enkripsi RSA

Ketika sebuah pesan pribadi (*plainteks*) akan dikirimkan melalui media transmisi yang tidak aman, maka perlu melakukan proses enkripsi untuk menjaga kerahasiaan dari pesan tersebut. Untuk melakukan proses enkripsi RSA, pesan disusun menjadi blok $x_1, x_2, \ldots$, sedemikian rupa sehingga setiap blok merepresentasikan nilai di dalam rentang 0 sampai $r-1$. Fungsi enkripsi memiliki masukan pesan sebagai sebuah larik *byte* p dan sebuah kunci publik. Kunci publik terdiri dari 2 elemen yaitu n dan e.

2.4.4 Konsep Dasar Dekripsi RSA

Setelah pesan tersandi (*cipherteks*) diterima oleh penerima pesan, maka dilakukan proses dekripsi untuk mengembalikan pesan tersandi (*cipherteks*) menjadi pesan semula (*plainteks*). Proses dekripsi hanya bisa dilakukan dengan menggunakan kunci privat. Fungsi dekripsi memiliki

masukan teks sandi sebagai sebuah larik *byte* c dan sebuah kunci privat. Kunci privat terdiri dari 3 elemen yaitu d, p dan q.

2.4.5 Konsep Dasar Perhitungan Matematis

- **Fungsi Phi-Euler**

  Fungsi Phi-Euler $\varphi(n)$ merupakan fungsi terhadap bilangan bulat positif n yang menyatakan banyaknya bilangan Z_n yang mempunyai invers terhadap operasi pergandaan. Ingat bahwa Z_n belum tentu merupakan group terhadap operasi pergandaan. Dengan kata lain $\varphi(n)$ adalah banyaknya elemen $\{x, 0 \pounds x < n \mid gcd(x,n) = 1\}$. Jika $n=p.q$ dengan p dan q adalah bilangan prima, maka $\varphi(n) = (p-1)(q-1)$. Jika n adalah bilangan prima, maka $\varphi(n)=n-1$.

- **Algoritma Euclide**

  Algoritma ini digunakan untuk mencari nilai pembagi persekutuan terbesarndari dua bilangan bulat. Algoritma ini didasarkan pada pernyataan berikut ini. Diberikan bilangan bulat r_0 dan r_1 dengan $r_0 \geq r_1$, kemudian dihitung menggunakan algoritma pembagian:

  $r_0 = q_1 r_1 + r_2 \quad 0 < r_2 < r_1$

  $r_1 = q_2 r_2 + r_3 \quad 0 < r_3 < r_2$

  .....

  $r_{n-2} = q_{n-1} r_{n-1} + r_n \quad 0 < r_n < r_{n-1}$

  $r_{n-1} = q_n r_n$

  Dari pernyataan di atas, dapat ditunjukkan bahwa:

  $\gcd(r_0, r_1) = \gcd(r_1, r_2) = \ldots = \gcd(r_{n-1}, r_n) = \gcd(r_n, 0) = r_n$

2.4.6 RSA Sebagai Standar Resmi dan De Facto

Sistem kriptografi dengan menggunakan algoritma RSA merupakan bagian dari banyak standar resmi di seluruh dunia. Standar ISO (*International Standard Organization*) mencantumkan RSA sebagai algoritma kriptografi yang cocok, begitu juga dengan standar keamanan ITU-TC.509.

Sistem RSA adalah bagian dari standar *Society for World Wide Interbank Financial Telecommunications* (SWIFT), standar *rench financial industry's* ETEBAC 5, standar ANSI X9.31 rDSA dan naskah standar X9.44 untuk industri perbankan Amerika Serikat. RSA juga dicantumkan dalam standar pengelolaan kunci Australia, AS2805.6.5.3.

Algoritma RSA dapat ditemukan dalam standar-standar internet dan protokol-protokol yang dianjurkan, termasuk S/MIMF, IPSec, dan TLS (penerus dari standar SSL), demikian juga dengan standar PKCS untuk penggunaan dalam perangkat lunak. OSI *Implementers Workshop* (OIW) mengeluarkan perjanjian pemakai berkenaan dengan PKCS, yang mencangkup penggunaan RSA.

RSA sebagai standar de facto, sistem RSA sudah digunakan begitu luas dalam sistem kriptografi kunci publik belakangan ini dan sering disebut sebagai standar yang de facto. Tanpa memperhatikan standar-standar resmi, keberadaan standar de facto sangat penting untuk pengembangan ekonomi digital. Jika satu sistem kunci publik digunakan dimana saja untuk pembuktian keaslian, kemudian dokumen digital yang telah ditandai dapat ditukarkan antara *user-user* yang berada pada negara-negara yang berbeda menggunakan perangkat lunak yang berbeda pada platform-platform yang berbeda.

2.4.7 Penggunaan RSA pada Kehidupan Sehari-hari

Mengamankan sebuah data dengan kunci dan hasil enkripsi yang cukup panjang merupakan keunggulan tersendiri dari enkripsi menggunakan algoritma RSA. RSA merupakan algoritma pertama yang cocok untuk *digital signature* seperti halnya enkripsi, dan salah satu yang paling maju dalam bidang kriptografi kunci publik. RSA masih digunakan secara luas dalam protokol *E-Commerce*, dan dipercaya dalam mengamankan dengan menggunakan kunci yang cukup panjang.

Banyak orang yang ragu dalam keamanan dalam penggunaan *e-banking*. RSA adalah salah satu bagian dari solusi masalah tersebut selain

itu juga ada SSL (*Secure Socket Layer*), *Digital Signature*, dan *Certificate of Authority*. Dengan adanya kunci publik dan kunci privat, data-data yang dikirmkan saat melakukan transaksi *e-banking* lebih terjamin kemanannya, hal ini karena untuk tahu data yang dikirim, harus tahu kombinasi kunci privat dan kunci publiknya.

2.5 Algoritma Kriptografi AES

Algoritma kriptografi AES merupakan standar enkripsi menggunakan kunci simetris yang diadobsi oleh pemerintah Amerika Serikat. Standar algoritma AES terdiri dari 3 blok *cipher*, yaitu AES-128, AES 192, dan AES 256. Algoritma AES telah dianalisis secara luas dan sekarang digunakan di seluruh dunia seperti pendahulunya yaitu *Data Encryption Standard* (DES).

2.5.1 Sejarah AES

Pada awal Januari 1997, setelah beberapa tahun standar penyandian simetris DES dianggap tidak lagi aman, lembaga standar Amerika Serikat NIST (*National Institute of Standard and Tecnology*) mengumumkan diadakannya kontes pemilihan algoritma standar terbaru menggantikan DES dengan harapan algoritma standar yang baru dapat bertahan paling tidak 30 tahun serta mampu memproteksi informasi rahasia selama 100 tahun. Pada bulan April di tahun yang sama, NIST mengumumkan persyaratan yang harus dipenuhi pada algoritma yang baru, antara lain:

1. *Block Cipher*.
2. Panjang blok 128 bit.
3. Panjang kunci yang bervariasi (128 bit, 192 bit, 256 bit).
4. Memiliki kekuatan yang sama atau lebih baik dari Triple DES.
5. Dapat diimplementasikan dalam dua bahasa pemrograman yaitu bahasa C dan java.
6. Jika algoritma tersebut terpilih maka akan menjadi standar publik yang tidak menerima bayaran atas hak cipta algoritma.

Setelah melalui tahapan beberapa seleksi akhirnya terdapat 15 algoritma yang menjadi finalis AES, kemudian dilakukan proses analisis antar para kontestan maupun publik yang mengerti terhadap pemilihan AES. Hampir seluruh kandidat algoritma saling mengungkapkan kelemahan satu sama lain namun hal yang mengejutkan adalah *Rijndael*, tidak terdapat satupun komentar negatif yang ditujukan pada algoritma ini. Kemudian komentar datang dari Eli Biham mengenai jumlah *round* minimal yang harus dipenuhi supaya algoritma tersebut dinyatakan aman.

Akhirnya NIST memilih algoritma *Rijndael* yang dikembangkan oleh Joan Daemen dan Vincent Rijment sebagai sistem penyandian AES pada tahun 2000. Pemilihan *Rijndael* berdasarkan pada kriteria:

1. **Keamanan**. Sistem penyandian harus tahan terhadap serangan analisis sandi selain serangan secara *brute force*.
2. **Biaya**. Sistem penyandian harus memiliki biaya komputasi dan memori yang terjangkau sehingga dapat diimplementasikan secara perangkat keras maupun perangkat lunak.
3. **Karakteristik algoritma dan implementasi**. Sistem penyandian harus bersifat terbuka, mudah digunakan, dan sedarhana.

Pada tahun 2001algoritma kriptografi AES (*Advanced Encryption Standard*) dipublikasikan oleh NIST. AES merupakan simetris blok *cipher* untuk menggantikan DES *(Data Encryption Standard)*.

2.5.2 Deskripsi AES

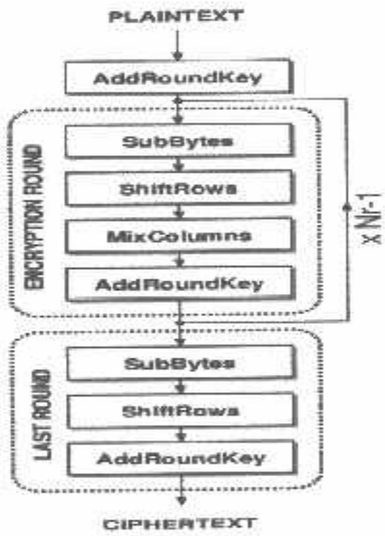
Pada algoritma AES, jumlah blok input, blok output, dan state adalah 128 bit. Dengan besar 128 bit, *key* yang digunakan pada algoritma AES tidak harus memiliki besar dengan blok input. *Cipher key* (K) pada algoritma *AES* bisa menggunakan kunci dengan panjang 128 bit, 192 bit, atau 256 bit. Perbedaan panjang kunci akan mempengaruhi jumlah *round* (ronde) yang akan diimplementasikan pada algoritma AES ini. Berdasarkan ukuran blok yang tetap, algoritma AES bekerja pada matriks berukuran 4×4 dimana tiap-tiap sel matriks terdiri atas 1 *byte* (8 bit).

Tabel 2.1 Perbandingan jumlah ronde dan kunci

	h Kunci (Nk)	Blok (Nb)	h Ronde (Nr)
128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

2.5.3 Struktur Enkripsi AES

Proses di dalam AES merupakan transformasi terhadap *state*. Sebuah *plainteks* dalam blok 128 bit terlebih dahulu diorganisir sebagai *state*. Enkripsi AES adalah transformasi terhadap *state* secara berulang dalam beberapa ronde. *State* yang menjadi keluaran ronde k menjadi masukan untuk ronde ke-$k + 1$.



Gambar 2.9 Struktur enkripsi AES

Proses enkripsi pada algoritma AES terdiri dari 4 jenis transformasi *bytes*, yaitu *SubBytes, ShiftRows, Mixcolumns, dan AddRoundKey*. Pada awal proses enkripsi , input yang telah dikopikan ke dalam akan mengalami transformasi *byte AddRoundKey*. Setelah itu, *state* akan mengalami transformasi *SubBytes, ShiftRows, MixColumns*, dan

AddRoundKey secara berulang-ulang sebanyak *Nr*. Proses ini dalam algoritma AES disebut sebagai *round function*.

2.5.3.1 *SubBytes*

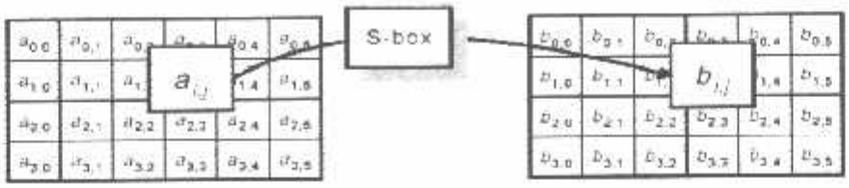
Algoritma AES menggunakan substitusi non linear pada ukuran *byte* yang disebut dengan *SubBytes*. Setiap elemen pada *state* dari elemen $a_{0,0}$ sampai dengan $a_{3,3}$ dikenakan transformasi *SubBytes*.

Pada *SubBytes* menggunakan tabel substitusi, yaitu dengan cara menginterpretasikan *byte* masukan $a_{i,j}$ sebagai 2 bilangan heksadesimal, kemudian digit kiri menunjukkan indeks baris dan digit kanan menunjukkan indeks kolom di tabel substitusi. Nilai *byte* pada tabel substitusi yang dirujuk oleh indeks baris dan kolom menjadi nilai yang mensubstitusi $a_{i,j}$.

Tabel 2.2 Contoh S-Box

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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

Pada tabel di atas merupakan *S-Box* yang digunakan sebagai operasi substitusi tak linear yang beroperasi secara mandiri pada setiap *byte*.



Gambar 2.10 Proses *SubBytes*

Hasil yang didapat dari pemetaan dengan menggunakan tabel *S-Box* ini sebenarnya adalah hasil dari dua proses transformasi *bytes*, yaitu:

1. *Invers* perkalian dalam GF (2^8) adalah fungsi yang memetakan 8 bit ke 8 bit yang merupakan *invers* dari elemen *finite field* tersebut. Satu *byte* a merupakan *invers* perkalian dari *byte* b bila a•b = 1, kecuali {0,0} dipetakan ke dirinya sendiri. Setiap elemen pada *state* akan dipetakan pada tabel *invers*.
2. Transformasi *affine* pada *state* yang telah dipetakan. Berikut transformasi *affine* dipetakan ke dalam bentuk matriks:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

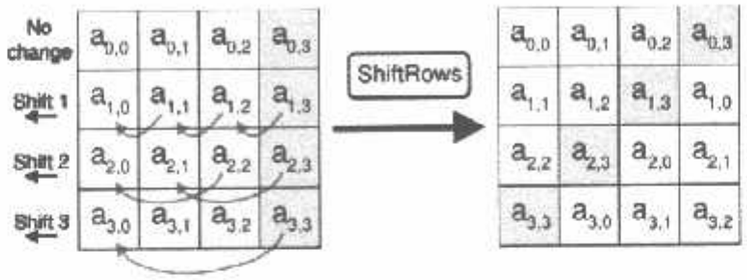
Gambar 2.11 Matriks *affine*

$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ merupakan urutan bit dalam elemen *state* atau *array byte* dimana b_7 adalah *most significant* bit atau bit dengan posisi paling kiri.

2.5.3.2 *ShiftRows*

Selain menggunakan substitusi untuk mengganti nilai pada elemen *state*, AES menggunakan permutasi pada *state*. Transformasi permutasi *state* disebut dengan transformasi *ShiftRows*. Transformasi *ShiftRows* dilakukan dengan menjalankan operasi *circular shift left* sebanyak *i* pada baris ke-*i* pada *state*. Transformasi *ShiftRows* pada dasarnya adalah poses

pergeseran bit dimana bit paling kiri akan dipindahkan menjadi bit paling kanan. Transformasi ini diterapkan pada bris 2, baris 3, dan baris 4. Baris 2 akan mengalami pergeseran bit sebanyak satu kali, sedangkan baris 3 dan baris 4 masing-masing mengalami pergeseran bit sebanyak dua kali dan tiga kali.



Gambar 2.12 Transformasi *ShiftRows*

2.5.3.3 *MixColumns*

MixColumns mengoperasikan setiap elemen yang berada dalam satu kolom pada *state*. Elemen pada kolom dikalikan dengan suatu *polynomials* tetap $a(x) = \{03\}x^3 + \{01\}x + \{02\}$. Berikut transformasi *MixColumns* dalam bentuk perkalian metriks:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Gambar 2.13 Matriks transformasi *MixColumns*

Melakukan proses penambahan pada operasi ini berarti melakukan operasi *bitwise* XOR, maka hasil perkalian dari metrics di atas adalah:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

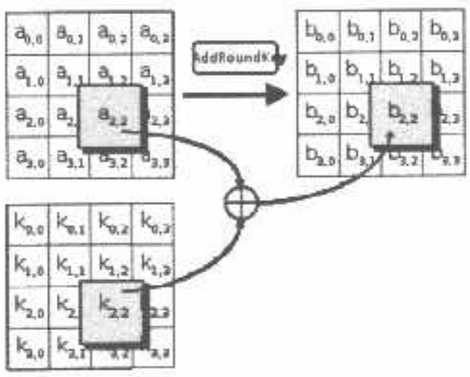
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

2.5.3.4 *AddRoundKey*

Transformasi *AddRoundKey* mencampur sebuah *state* masukan dengan kunci ronde yang dilakukan menggunakan operasi OR ($\oplus$). Setiap elemen pada *state* masukan yang merupakan sebuah *byte* dikenakan operasi eksklusif OR dengan *byte* pada posisi yang sama di kunci ronde (kunci ronde direpresentasikan sebagai *state*).

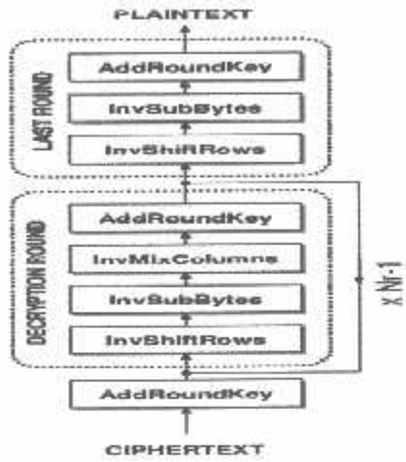
Transformasi *AddRoundKey* diimplementasikan pertama kali pada pada *round* = 0, dimana kunci yang digunakan adalah *initial key* (kunci yang dimasukkan oleh kriptografer dan belum mengalami proses *key expansion*).



Gambar 2.14 Transformasi *AddRoundKey*

2.5.4 Struktur Dekripsi AES

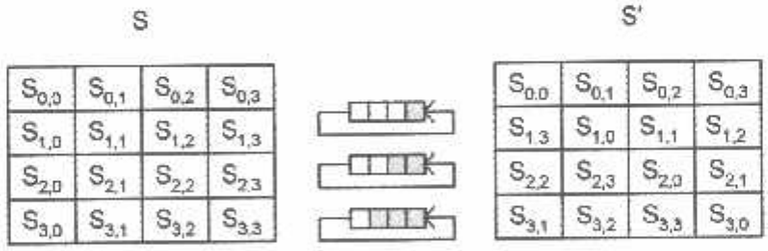
Pada proses dekripsi, transformasi *cipher* dapat dibalikkan dan diimplementasikan dalam arah yang berlawanan untuk menghasilkan *inverse cipher* yang mudah dipahami untuk algoritma AES. Transformasi *bytes* yang digunakan pada *invers cipher* adalah *InvShiftRows*, *InvSubBytes*, *InvMixColumns*, dan *AddRoundKey*. Untuk *AddRoundKey* merupakan transformasi yang bersifat *self-invers* dengan syarat menggunakan kunci yang sama.



Gambar 2.15 Struktur dekripsi AES

2.5.4.1 *InvShiftRows*

InvShiftRows merupakan kebalikan transformasi *byte ShiftRows*. Pada transformasi *InvShiftRows*, dilakukan pergeseran bit ke arah kanan sedangkan pada transformasi *ShiftRows* dilakukan pergeseran bit kea rah kiri. Pada baris kedua, pergeseran bit dilakukan sebanyak 3 kali, sedangkan pada baris ketiga dan baris keempat, dilakukan pergeseran sebanyak 2 kali dan 1 kali.



Gambar 2.16 Transformasi *InvShiftRows*

2.5.4.2 *InvSubBytes*

InvSubBytes merupakan transformasi *bytes* kebalikan dari *SubBytes*. Pada *InvSubBytes*, tiap elemen pada *state* dipetakan dengan menggunakan tabel *invers S-Box*. Tabel ini berbeda dengan tabel *S-Box* dimana hasil yang didapat dari tabel ini adalah hasil dari dua proses yang

urutannya berbeda, yaitu transformasi *affine* terlebih dahulu baru kemudian perkalian *invers* dalam GF (2^8).

Tabel 2.3 *Invers* S-Box

x \ y	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	dd	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

Tabel di atas merupakan tabel *invers S-Box* yang berfungsi sebagai tempat pemetakan setiap elemen pada *state*. Setelah *state* selesai dipetakan maka dilanjutkan dengan transformasi *affine* yang dipetakan kedalam bentuk matriks.

$$\begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Gambar 2.17 Matriks *invers affine*

Cara perkalian *invers* yang dilakukan pada transformasi *InvSubBytes* sama dengan cara perkalian *invers* pada *SubBytes*.

2.5.4.3 *InvMixColumns*

Pada transformasi *InvMixColumns* menggunakan perkalian matriks antara sebuah konstan dengan *state*. Konstan yang dipakai *InvMixColumns*

merupakan *invers* matriks konstan pada transformasi *MixColumns*. Kolom-kolom pada tiap *state* akan dipandang sebagai polinom atas GF (2^8) dan mengalikan modulo $x^4 + 1$ dengan polinom tetap $a^{-1}(x)$ yang diperoleh dari:

$$a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

atau dalam matriks:

$$s'(x) = a(x) \otimes s(x)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Gambar 2.18 Matriks transformasi *InvMixColumns*

Hasil perkalian di atas:

$$s'_{0,c} = (\{0E\} \bullet s_{o,c}) \oplus (\{0B\} \bullet s_{1,c}) \oplus (\{0D\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$
$$s'_{1,c} = (\{09\} \bullet s_{o,c}) \oplus (\{0E\} \bullet s_{1,c}) \oplus (\{0B\} \bullet s_{2,c}) \oplus (\{0D\} \bullet s_{3,c})$$
$$s'_{2,c} = (\{0D\} \bullet s_{o,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0E\} \bullet s_{2,c}) \oplus (\{0B\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{0B\} \bullet s_{o,c}) \oplus (\{0D\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0E\} \bullet s_{3,c})$$

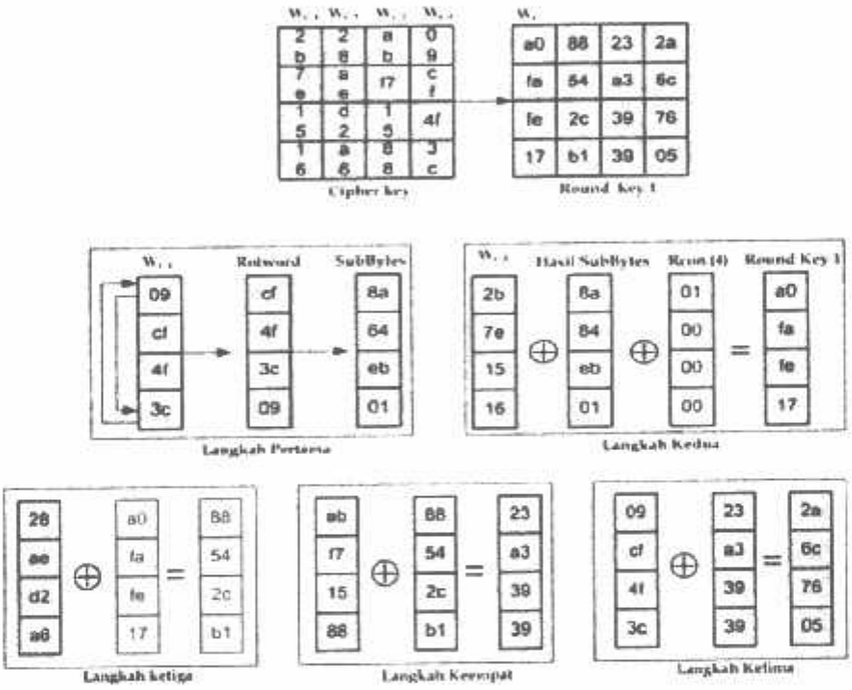
2.5.4.4 *Invers AddRoundKey*

Transformasi *Invers AddRoundKey* tidak mempunyai perbedaan dengan transformasi *AddRoundKey* karena hanya perlu dilakukan operasi sederhana dengan menggunakan operasi *bitwise* XOR.

2.5.5 Ekspansi Kunci AES

Penyandian AES membutuhkan kunci ronde untuk setiap ronde transformasi. Kunci ronde ini dibangkitkan (diekspansi) dari kunci AES. Ekspansi kunci ronde dengan panjang 192 bit dan 256 bit mirip dengan ekspansi kunci AES 128 bit dengan sedikit modifikasi.

Masukan prosedur ekspansi kunci adalah Nb(Nr + 1) kata sehingga bisa digunakan AES 128 bit atau 4 kata sehingga 4(10+1) dan menghasilkan sebuah larik sepanjang 44 kata yang menjadi kunci ronde. Jika dihitung dalam satuan bit menjadi 44 x 32 bit = 1408 bit, jadi ekspansi dari 128 menjadi 1408 bit upa-kunci. Proses ini disebut dengan kunci skedul. Upa-kunci ini diperlukan karena setiap putaran merupakan suatu inisial dari Nb kata untuk Nr = 0 dan 2 Nb untuk Nr = 1, 3 Nb untuk Nr = 2,……, 11 Nb untuk Nr = 10. Dari operasi ini didapatkan kunci skedul yang berisi larik linear 4 *byte* kata (w_i), $0 \le i \le Nb(Nr + 1)$.



Gambar 2.19 Proses putaran kunci

Kunci kode terdiri dari W_{i-1}, W_{i-2}, W_{i-3}, dan W_{i-4}, tetapi dalam kasus putaran kunci hal ini dibalik menjadi W_{i-4} pada baris pertama, W_{i-3} pada baris kedua, W_{i-2} pada baris ketiga, W_{i-1} pada baris terkahir. Proses perputaran kunci dimulai dari W_{i-1}. Pada W_{i-1}, dilakukan *rotword* dengan cara menukar bit paling atas ke bit paling bawah. Setelah didapatkan *rotword* maka hasilnya diproses *SubBytes* seperti pada Gambar 2.19. hasil

dari *SubBytes*, kemudian di XOR-kan dengan W_{i-4} dan Rcon (i) sehingga didapat upa-kunci yang baru.

2.6 Borland Delphi 7

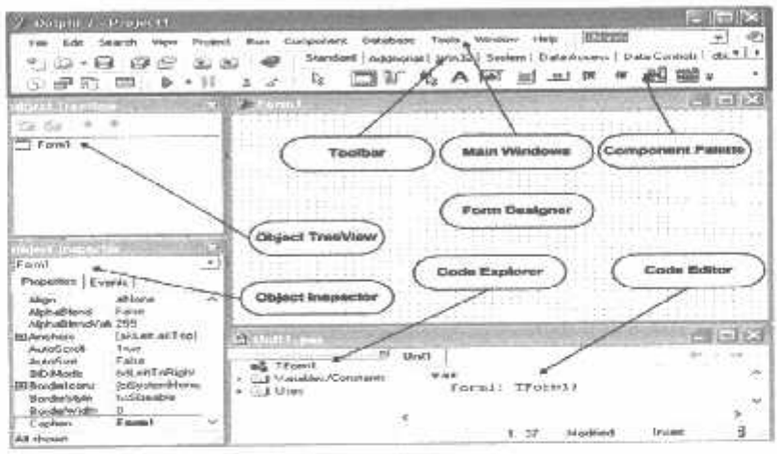
Delphi 7 merupakan suatu bahasa pemrograman (*development language*) yang digunakan untuk merancang suatu aplikasi program. Delphi termasuk dalam pemrograman bahasa tingkat tinggi (*high level language*). Maksud dari bahasa tingkat tinggi yaitu perintah-perintah programnya menggunakan bahasa yang mudah dipahami oleh manusia.

Bahasa pemrograman Delphi disebut bahasa procedural artinya mengikuti urutan tertentu. Dalam membuat aplikasi perintah-perintah, Delphi menggunakan lingkungan pemrograman visual.

Delphi merupakan generasi penerus dari turbo pascal. Pemrograman Delphi dirancang untuk beroperasi dibawah sistem operasi Windows. Program ini mempunyai beberapa keunggulan, yaitu produktivitas, kualitas, pengembangan perangkat lunak, kecepatan compiler, pola desain yang menarik serta diperkuat dengan bahasa pemrograman yang tersruktur dalam struktur bahasa pemrograman *Object Pascal*. Sebagian besar pengembang Delphi menuliskan dan mengkompilasi kode program di dalam lingkungan pengembang aplikasi atau *Integrated Development Environment* (IDE). Lingkungan kerja IDE ini menyediakan sarana yang diperlukan untuk merancang, membangun, mencoba, mencari atau melacak kesalahan, serta mendistribusikan aplikasi. Sarana-sarana inilah yang memungkinkan pembuatan pembuatan *prototype* aplikasi menjadi lebih mudah dan waktu yang diperlukan untuk pengembangan aplikasi menjadi lebih singkat.

2.6.1 IDE (*Integrated Development Environment*)

IDE (*Integrated Development Environment*) merupakan lingkungan/wilayah dimana seluruh *tools* atau komponen-komponen yang dibutuhkan untuk merancang atau membangun aplikasi program.



Gambar 2.20 Lembar kerja Borland Delphi

Secara umum IDE Delphi dikelompokkan menjadi 9 bagian, yaitu :

1. *Main Window* (Jendela Utama)

   Di dalam jendela utama Delphi terdapat menu-menu sebagaimana menu aplikasi Windows umumnya, *toolbar* yang merupakan langkah cepat dari beberapa menu, dan *component palette* yaitu gudang komponen yang akan dibuat untuk membuat aplikasi.

2. *Main Menu* (Menu Utama)

   Menu utama pada Delphi memiliki kegunaan yang sama seperti program aplikasi *windows* lainnya. Pada fasilitas menu terdapat tombol yang digunakan untuk memanggil atau menyimpan program.

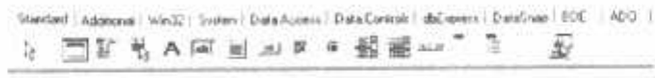
3. *Toolbar*

   Delphi memiliki beberapa *toolbar* yang masing-masing memiliki perbedaan fungsi dan setiap tombolnya berfungsi sebagai pengganti suatu menu perintah yang sering digunakan. *Toolbar* sering disebut juga dengan *Speedbar*. *Toolbar* terletak pada bagian bawah baris menu.

Tombol-tombol yang terletak pada *toolbar* dapat ditambah atau dikurangi sesuai kebutuhan.

4. *Component Palette*

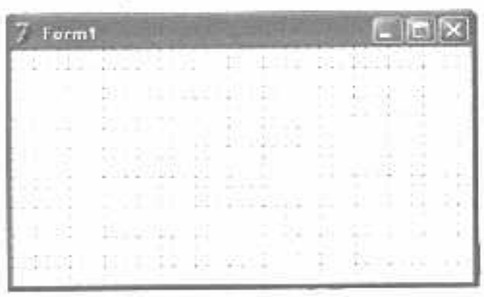
Component Palette berisi kumpulan ikon yang melambangkan komponen-komponen yang terdapat pada VCL (*Visual Component Library*).



Gambar 2.21 *Component Palette*

5. *Form Designer*

Form Designer merupakan suatu objek yang dapat dipakai sebagai tempat untuk merancang program aplikasi. *Form* berbentuk sebuah meja kerja yang dapat diisi dengan komponen-komponen yang di ambil dari *component palette*. Pada saat program Delphi dijalankan, Delphi akan memberikan sebuah *form* kosong yang disebut dengan Form1.



Gambar 2.22 Lembar kerja *form*

Dalam sebuah *form* terdapat titik-titik yang disebut grid yang fungsinya untuk membantu pengaturan tata letak objek yang dimasukkan ke dalam lembar kerja.

6. *Object Inspector*

Object ini digunakan untuk mengatur *properties* dan *event* suatu komponen, akan tetapi tidak dapat mengubah langsung properti-properti yang tidak ditampilkan kecuali melalui penulisan kode program. *Properties* digunakan untuk menentukan seting suatu objek. Satu objek

memiliki beberapa properti yang dapat diatur langsung dari *object inspector* maupun melalui kode program. Seting ini mempengaruhi cara kerja objek tersebut saat aplikasi dijalankan. Event merupakan bagian yang dapat diisi dengan kode program tertentu yang berfungsi untuk menangani *event-event* (berupa sebuah prosedur) yang dapat direspon oleh sebuah komponen. Event adalah peristiwa atau kejadian yang diterima oleh suatu objek, misal : klik, drag, dan lain-lain. Event yang diterima objek akan memicu Delphi menjalankan kode program.

Gambar 2.23 Lember kerja *Object Inspector*

Tab Properties digunakan untuk mengubah properti komponen. Properti dengan tanda (+) menunjukkan bahwa properti tersebut mempunyai subproperti.

Tab Event, bagian yang dapat diisi dengan kode program tertentu yang berfungsi menangani kejadian-kejadian yang berupa sebuah prosedur yang dapat direspon oleh sebuah komponen.

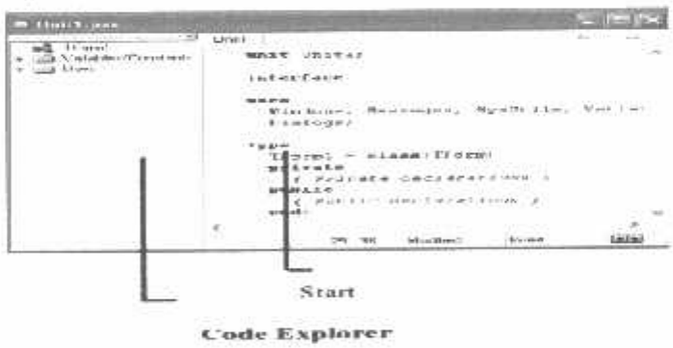
7. Code Editor

Code editor merupakan tempat untuk menuliskan kode program menggunakan bahasa *object pascal*. Kode program tidak perlu ditulis secara keseluruhan karena Delphi sudah menyediakan blok atau kerangka untuk menulis kode program.

Gambar 2.24 Lembar kerja code editor

8. Code Explorer

Code explorer digunakan untuk memudahkan perpindahan antar file unit di dalam jendela code editor. Code explorer berisi diagram pohon yang menampilkan semua tipe, *class*, *method*, *variabel global*, rutin global yang telah didefinisikan di dalam unit. Saat memilih sebuah item dalam code explorer, kursor akan berpindah menuju implementasi dari item yang dipilih di dalam code editor.

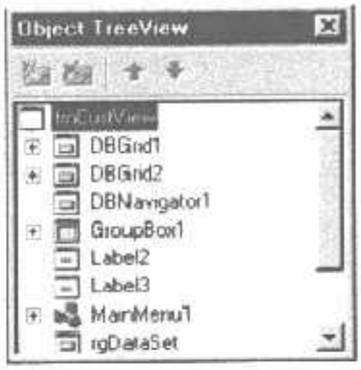


Gambar 2.25 Lembar kerja code explorer

9. *Object TreeView*

Object TreeView menampilkan diagram pohon dari komponen-komponen yang bersifat visual maupun nonvisual yang telah terdapat di dalam *form*, *data module*, atau *frame*. *Object TreeView* juga menampilkan hubungan logika antar komponan. Apabila mengklik

kanan salah satu item yang terdapat di dalam diagram pohon, maka akan terlihat konteks menu komponen sebelumnya.



Gambar 2.26 Jendela *Object TreeView*

2.6.2 File-File Penyusun *Project* Delphi

Sepintas sebuah program aplikasi yang dapat dibuat dengan menggunakan Delphi hanya terdiri dari file *project* dan sebuah unit. Namun kenyataannya terdapat beberapa file yang dibentuk pada saat membangun sebuah program aplikasi. Berikut ini merupakan file-file penyusun project yang terdapat pada program Delphi, yaitu :

1. File Project (.Dpr) dan file Unit (.Pas)

   Sebuah program Delphi terbangun dari modul-modul *source code* yang disebut unit. Delphi menggunakan sebuah file *project* (.Dpr) untuk menyimpan program utama. File sumber untuk unit biasanya berisi sebagian besar kode di dalam aplikasi, file ini ditandai dengan ekstensi (.Pas). Setiap aplikasi atau terdiri atas file *project* tunggal atau lebih dalam file unit.

2. File *Form* (.Dfm)

   File *form* adalah file biner yang dibuat Delphi untuk menyimpan informasi yang berkaitan dengan *form*, dan setiap dan setiap *form* mempunyai sebuah file unit (.Pas).

3. File *Resource* (.Res)

   File *resource* merupakan file biner yang berisi sebuah ikon yang digunakan oleh *project*. File ini secara terus-menerus di-update atau diubah oleh Delphi sehingga file ini tidak bisa diubah oleh pemakai.

Pemakai dapat menambahkan file *resource* pada aplikasi dan menghubungkan dengan file *project*, dimana pemakai dapat menggunakan sebuah editor *resource*, misalnya *image* editor untuk membuat file *resource*.

4. File *Project Options* (.Dof) dan File *Desktop Setting* (.Dsk)

   File *project options* merupakan file yang berisi *option-option* dari suatu *project* yang dinyatakan melalui perintah *Options* dari menu *project*. Sedangkan file *desktop setting* berisi *option-option* yang dinyatakan melalui perintah *Environmet Options* dari menu *Tools*. Perbedaan di antara jenis file tersebut adalah bahwa file *project options* dimiliki oleh setiap *project*, sedangkan file *desktop setting* dipakai untuk lingkungan Delphi. Apabila ada kerusakan pada kedua jenis file tersebut dapat mengganggu proses kompilasi. Prosedur yang dapat ditempuh untuk menangani gangguan tersebut adalah dengan menghapus kedua jenis file tersebut yang memiliki ekstensi (.Dof) dan (.Dsk), karena kedua file tersebut akan terbentuk secara otomatis pada saat menyimpan *project*.

5. File *Backup* (.~dp, .~df, .~pa)

   File-file dengan ekstensi di atas merupakan file *backup* dari suatu *project, form*, dan unit. Ketiga jenis file tersebut akan terbentuk pada saat proses penyimpanan untuk yang kedua kalinya. Karena ketiga file tersebut berjenis *backup* (cadangan), maka ketiga jenis file tersebut berisi salinan terakhir dari file-file utama sebelum disimpan lebih lanjut.

6. File Jenis Lain

   File-file dengan ekstensi lain yang dapat ditemukan dalam folder tempat penyimpanan program aplikasi selain yang memiliki ekstensi yang telah disebutkan pada umumnya adalah file-file yang dibentuk oleh *compiler* dan beberapa file *Windows* yang digunakan Delphi. File-file tersebut adalah :

   a) File *executable* (.Exe). File ini dibentuk oleh *compiler* dan merupakan file eksekusi dari program aplikasi. File ini berdiri

sendiri dan hanya memerlukan file *library* di DLL, VBX dan lain-lain.

b) File unit *object* (.Dcu). File ini merupakan file unit (.Pas) yang telah dikompilasi oleh *compiler* yang akan dihubungkan dengan file eksekusi.

c) File *Dinamic Link Library* (.Dll). File ini dibentuk *compiler* apabila kita merancang (.Dll) sendiri.

d) File *Help* (.Hlp). file ini merupakan file *windows* dan merupakan file *help* standar yang dapat dipakai di program aplikasi Delphi.

e) File *Image* (.Wmf, .Bmp, .Ico). File-file ini merupakan file *windows* dari program aplikasi selain Delphi yang dapat digunakan untuk mendukung program aplikasi yang telah dirancang agar tampak lebih menarik.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Sistem yang akan dibuat pada penelitian ini adalah sebuah aplikasi yang digunakan untuk melakukan suatu proses enkripsi dan dekripsi data berupa teks atau tulisan. Sistem ini memiliki dua proses enkripsi dan dekripsi yang menggunakan algoritma kriptografi RSA, algoritma kriptografi AES, serta gabungan antara algoritma RSA dan algoritma AES. Tujuan dari analisis ini adalah agar sistem yang dirancang menjadi tepat guna dan ketahanan dari sistem tersebut akan lebih terjaga. Di samping itu dengan dialakukannya analisis terhadap sistem yang akan dibuat maka akan memudahkan dalam proses pengerjaannya dan jika terdapat perbaikan atau penambahan dalam sistem tersebut, maka akan mudah untuk diselesaikan.

Tahapan-tahapan penelitian yang dilakukan dalam pembuatan aplikasi kriptografi antara lain:

1. Mempelajari ilmu tentang kriptografi dan semua hal-hal tentang algoritma RSA dan algoritma AES beserta metode-metode yang ada di dalamnya.
2. Menganalisa dan merancang sistem enkripsi dan dekripsi dengan menggunakan algoritma RSA dan algoritma AES.
3. Uji coba sistem yang dilakukan adalah file teks yang memiliki ekstensi seperti; *.txt, *.doc, *.docx, *.odt.
4. Membandingkan hasil enkripsi dan dekripsi, baik dari segi kecepatan proses enkripsi maupun proses dekripsi, dan perbandingan ukuran file sebelum dan sesudah proses enkripsi.

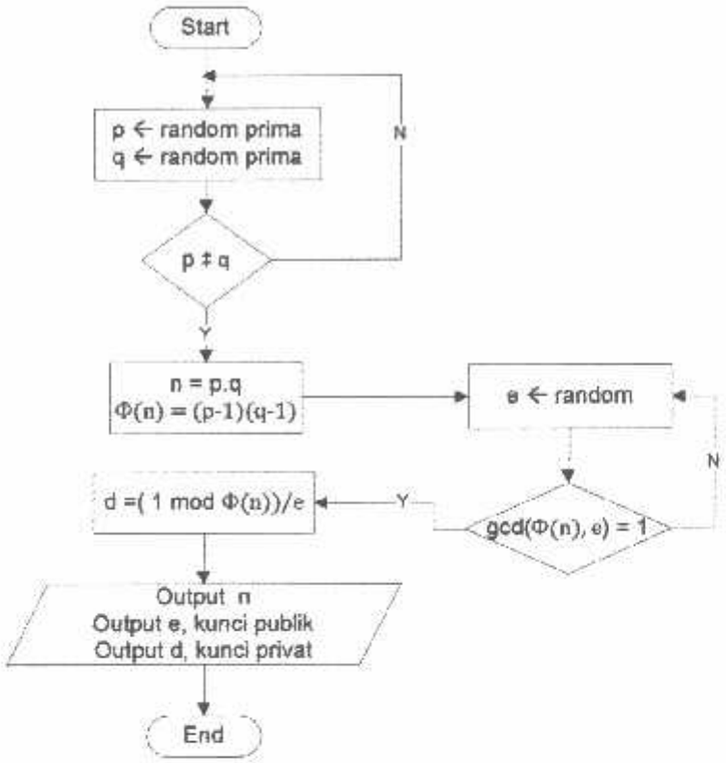
3.1.1 Anilisis Algoritma RSA

Dalam sistem kriptografi algoritma RSA ini terdapat tiga proses utama, yaitu proses pembangkitan kunci, proses enkripsi *plainteks* menjadi *cipherteks*, dan proses dekripsi *cipherteks* menjadi *plainteks* kembali.

3.1.1.1 Pembangkit Kunci RSA

Prinsip kerja algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi faktor-faktor prima. Pemfaktoran dilakukan untuk mendapatkan kunci publik dan kunci privat. Berikut ini adalah tahapan-tahapan dalam membangkitkan kunci RSA:

1. Mengambil nilai P dan q yang merupakan bilangan prima, proses ini bisa dilakukan secara acak atau ditentukan sendiri nilai primanya oleh penerima pesan. Sifat dari kedua bilangan ini adalah rahasia, dimana penerima pesan saja yang mengetahui.
2. Hitung nilai $n = p.q$, sifat dari n ini adalah bisa diketahui oleh publik.
3. Hitung nilai $\phi(n) = (p-1)(q-1)$. Sifat dari bilangan ini adalah rahasia.
4. Pilih kunci publik yang disimbolkan dengan e. syarat dari pemilihan kunci ini adalah e harus relative prima terhadap $\phi(n)$.
5. Membangkitkan kunci privat dengan persamaan $e \times d = 1 \bmod \phi(n)$ atau $d = e^{-1} \bmod \phi(n)$.
6. Hasil dari algoritma di atas adalah :
   a. Kunci publik (e,n).
   b. Kunci privat (d,n).



Gambar 3.1 Flowchart pembangkit kunci RSA

Pada gambar flowchart di atas dapat dijelaskan bahwa proses pengambilan nilai bilangan prima *p* dan *q* dilakukan secara acak (*random*), kemudian ditentukan nilai prima *p* dan *q* yang telah diambil tersebut sama atau tidak, jika sama maka dilakukan pengambilan kembali nilai prima, jika tidak maka dilanjutkan pada proses untuk menghitung nilai modulus *n* dan *Φ(n)*. Nilai modulus *n* digunakan untuk melakukan proses enkripsi dan dekripsi, sedangkan modulus *Φ(n)* digunakan untuk mencari nilai kunci privat. Setelah nilai n dan *Φ(n)* didapat, maka dilanjutkan dengan proses pengambilan bilangan bulat untuk nilai *e* sebagai kunci publik dengan syarat *e* harus relatif prima terhadap *Φ(n)*. Setelah didapat nilai *e* maka dilanjutkan dengan proses pengambilan nilai *d* sebagai kunci privat. Rumus yang digunakan untuk mencari nilai kunci privat adalah:

$$d = \frac{1}{e} \bmod \Phi(n)$$

dimana:

d = Kunci privat

e = kunci publik

Φ(n) = nilai modular

Keluaran (*output*) yang ditampilkan adalah modulus (n), kunci publik (e), dan kunci privat (d).

Contoh:

Jika ditentukan bilangan $p = 17$ dan $q = 31$, maka

$n = p \times q$

$n = 17 \times 31 = 527$

$\phi(n) = (p-1)(q-1)$

$\phi(n) = (17-1)(31-1) = 480$

Misalnya, e dipilih 77, nilai 77 memenuhi syarat karena gcd(480,77) = 1. Jadi kunci publik yang digunakan adalah $K_{publik} = (77,527)$.

Sedangkan parameter d untuk kunci privat adalah $d = \frac{1}{77} \bmod 480$, dengan syarat $d < 480$ sehingga didapat nilai $d = 293$. Jadi kunci privat yang digunakan adalah $K_{privat} = (293,527)$.

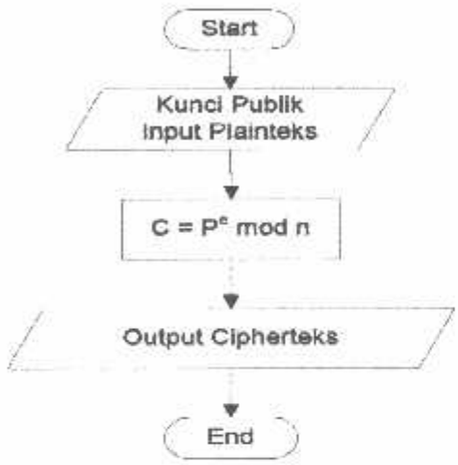
3.1.1.2 Enkripsi RSA

Untuk melakukan proses enkripsi menggunakan algoritma RSA memiliki tiga buah masukan, yaitu masukan untuk nilai kunci publik, masukan untuk nilai modular, dan masukan untuk file yang akan dienkripsi. Fungsi enkripsi memiliki masukan *plainteks* sebagai sebuah larik *byte P* dan sebuah kunci publik. Kunci publik terdiri dari dua elemen yaitu n dan e.

Proses enkripsi pada algoritma RSA ini merupakan proses perpangkatan atau fungsi pangkat.Tahapan-tahapan yang dilakukan untuk melakukan proses enkripsi:

1. Masukkan *plainteks* (P) yang akan dienkripsi.

2. Masukkan kunci publik (e,n) yang didapat dari proses pembangkit kunci RSA.
3. Lakukan proses enkripsi dengan menggunakan rumus:
   $C = P^e \bmod n$.
4. Hasil dari proses enkripsi adalah *cipherteks* (C).



Gambar 3.2 Flowchart proses enkripsi

Untuk melakukan proses enkripsi RSA, terlebih dahulu harus membangkitkan kunci publik. Pada contoh sebelumnya telah dijelaskan tentang proses untuk membangkitkan kunci, pada gambar flowchart di atas setelah melakukan pembangkitan kunci maka langkah selanjutnya melakukan *input* data kemudian dilakukan proses enkripsi dengan menggunakan rumus:

$$C = P^e \bmod n$$

dimana:

P = *Plainteks*

C = *Cipherteks*

e = Kunci publik

n = Nilai modular

Keluaran (*output*) yang dihasilkan berupa *file cipher*.

Contoh:

Jika ditentukan $P = 51$, $K_{publik} = (77,527)$,

$K_{publik} = (e, n)$, sehingga $e = 77$, dan $n = 527$.

Untuk melakukan proses enkripsi rumus yang digunakan adalah:

$C = P^e \bmod n$

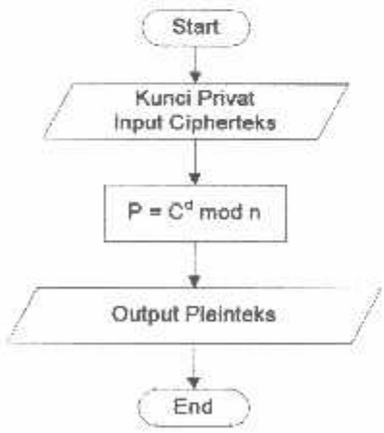
$C = 51^{77} \bmod 527 = 493$

Jadi hasil dari proses enkripsi adalah C = 493.

3.1.1.3 Dekripsi RSA

Selain melakukan proses enkripsi yang merubah *plainteks* menjadi *cipherteks*, sistem ini juga dibuat melakukan dekripsi yang merubah *cipherteks* kembali menjadi *plainteks* untuk mendapatkan kembali informasi awal. Pada proses dekripsi terdapat tiga buah masukan, yaitu masukan untuk kunci privat, nilai modular, dan file yang akan di dekripsi. Tahapan-tahapan yang perlu dilakukan untuk melakukan proses dekripsi adalah:

1. Masukkan *cipherteks* (C) yang akan dilakukan proses dekripsi.
2. Masukkan kunci privat dan nilai modular.
3. Lakukan proses dekripsi dengan menggunakan rumus:
   $P = C^d \bmod n$



Gambar 3.3 Flowchart global proses dekripsi

Pada gambar flowchart di atas, proses dekripsi di mulai dengan memasukkan kunci privat yang telah dibuat serta *cipherteks* yang akan didekripsi, selanjutnya proses dekripsi dilakukan dengan menggunakan rumus:

$$P = C^d \bmod n$$

dimana:

P = *Plainteks*

C = *Cipherteks*

d = Kunci privat

n = Nilai modular

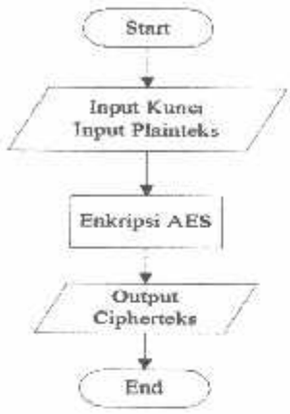
Hasil dari proses dekripsi merupakan plainteks atau file asli, apabila kunci yang dimasukkan adalah kunci yang tepat maka isi dari plainteks dapat dibaca, sedangkan bila kunci privat yang dimasukkan salah, maka plainteks tidak akan terbaca.

3.1.2 Analisis Algoritma AES

Pada sistem kriptografi algoritma AES ini terdapat dua proses utama, yaitu proses enkripsi dan proses dekripsi. Pada algoritma AES tidak perlu dilakukan pembangkitan kunci karena tipe dari algoritma ini adalah algoritma simetris, yaitu algoritma yang hanya menggunakan satu buah kunci dan bersifat rahasia.

3.1.2.1 Enkripsi AES

Pada proses enkripsi menggunakan algoritma AES memiliki dua buah masukan dan satu pilihan, pada masukan atau *inputam* terdiri dari masukan untuk kunci privat dan *plainteks* yang akan dienkripsi sedangkan pada pilihan yaitu melakukan pemilihan untuk menggunakan panjang kunci antara 128, 192, dan 256 bit. Berikut ini merupakan *flowchart* untuk proses enkripsi menggunakan algoritma AES:

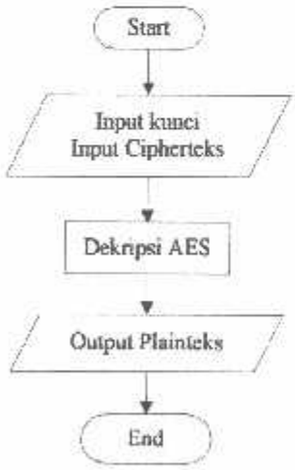


Gambar 3. 4 Flowchart enkripsi AES

Pada gambar *flowchart* diatas dapat dijelaskan bahwa pertama-tama dilakukan *input* kunci dan *plainteks* kemudian dilakukan proses enkripsi. Setelah proses enkripsi selesai *file ouput* berupa *cipherteks*.

3.1.2.2 Dekripsi AES

Pada dekripsi AES memiliki dua buah proses masukan (input), yaitu memasukkan *cipherteks* yang akan didekripsi dan memasukkan kunci yang sudah ditentukan pada saat proses enkripsi, apabila kunci yang digunakan tidak sama proses dekripsi akan tetap berjalan, tapi *plainteks* yang dihasilkan tidak sesuai seperti file asli.

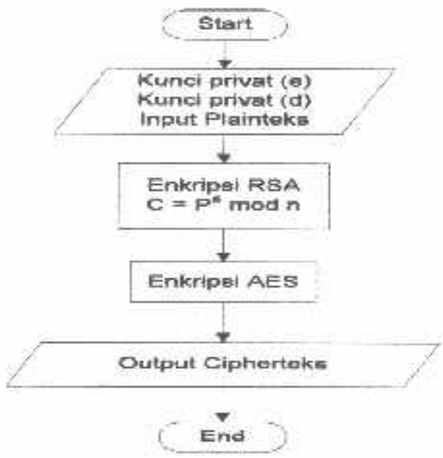


Gambar 3.5 Flowchart dekripsi AES

Pada *flowchart* dekripsi AES diatas dapat dijelaskan bahwa pertama memasukkan data atau cihperteks yang akan didekripsi dan memasukkan kunci yang sudah ditentukan, kemudian dilenjutkan dengan proses dekripsi sehingga menghasilkan *plainteks* atau teks asli.

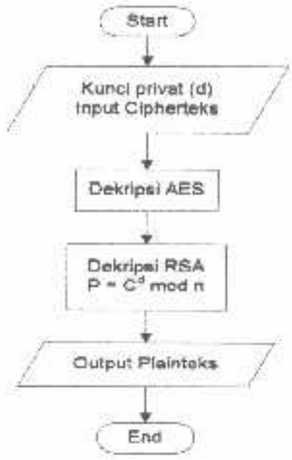
3.1.3 Analisis Algoritma Gabungan RSA Dan AES

Algoritma gabungan antara RSA dan AES disini adalah menggabungkan algoritma RSA dan algoritma AES dalam satu kali proses enkripsi dan dekripsi. Pada algoritma gabungan ini kunci yang digunakan untuk proses enkripsi ada dua dan bersifat rahasia, sedangkan untuk proses dekripsi menggunakan satu kunci dan bersifat rahasia.



Gambar 3.6 Flowchart Enkripsi RSA dan AES

Pada gambar di atas dapat dijelaskan bahwa pada proses enkripsi terdapat tiga buah masukan (*input*), yaitu kunci privat (e), kunci privat (d), dan *plainteks*. Setelah itu, dilakukan proses enkripsi dengan menggunakan algoritma RSA dengan menggunakankunci privat (e), hasil dari proses enkripsi atatu *cipher* tersebut kemudian di enkripsi lagi menggunakan algoritma AES dengan menggunakan kunci privat (d). hasil dari proses tersebut berupa *output cipher* hasil gabungan dua algoritma.



Gambar 3.7 Flowchart dekripsi RSA dan AES

Dari gambar *flowchart* di atas dapat dijelaskan bahwa pada proses awal dekripsi terdapat dua masukan (*input*), yaitu kunci privat (d) dan file *cipher* yang akan didekripsi. Setelah file *cipher* dan kunci privat dimasukkan langkah selanjutnya melakukan proses dekripsi menggunakan algoritma AES dengan kunci privat (d). Hasil dari proses dekripsi ini masih berupa file *cipher* yang kemudian didekripsi lagi menggunakan algoritma RSA dan tetap menggunakan kunci privat (d). hasil dari proses tersebut sudah berupa *plainteks* atau file asli.

3.2 Perancangan antarmuka Sistem

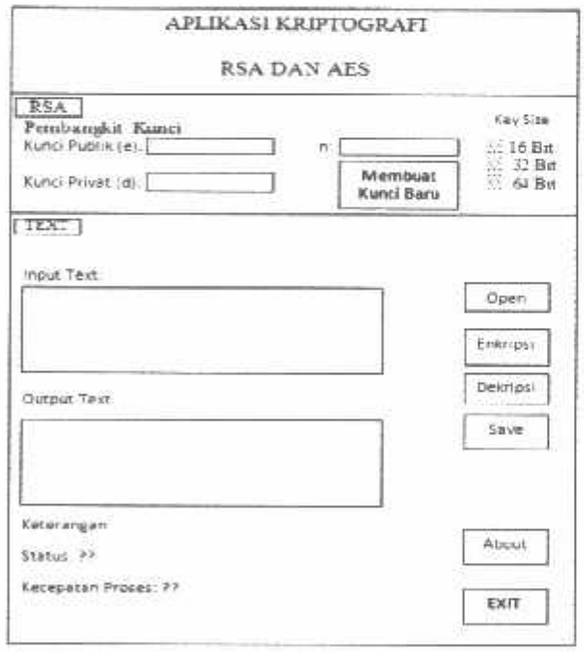
Perancangan antarmuka sistem ini digunakan untuk mendukung proses pembuatan program aplikasi kriptografi. Pada perancangan antarmuka ini terdapat 4 tampilan, yaitu:

1. Tampilan enkripsi dan dekripsi menggunakan algoritma RSA pada *text*.
2. Tampilan enkripsi dan dekripsi pada *text* menggunakan algoritma AES.
3. Tampilan enkripsi dan dekripsi file dokumen menggunakan algoritma AES

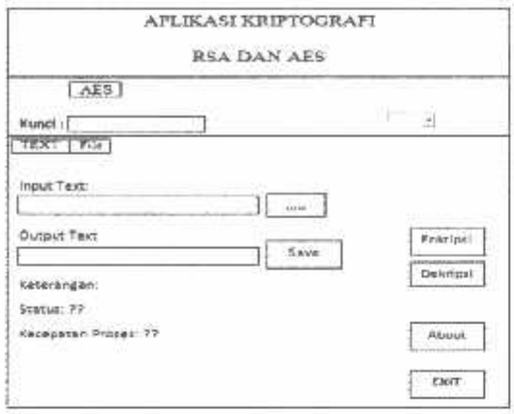
4. Tampilan enkripsi dan dekripsi *text* menggunakan algoritma gabungan RSA dan AES.

3.2.1 Desain Enkripsi Dan Dekripsi Menggunakan Algoritma RSA Pada *Text*

Pada tampilan ini terdapat pembangkit kunci yang didalamnya ada tiga *TEdit* untuk menampilkan hasil pembuatan kunci prima secara acak. Dan satu tombol (*button*) untuk melakukan proses pengambilan kunci. Selain itu ada *Radio Button* yang berfungsi sebagai tempat memilih panjang kunci yang digunakan. Panjang kunci yang bisa dipilih adalah 16 bit, 32 bit, dan 64 bit. Selain itu terdapat dua *TMemo* yang berfungsi untuk menampilkan masukan (*input*) dari teks yang akan dienkripsi atau didekripsi dan menampilkan hasil enkripsi berupa *cipher* dan hasil dekripsi berupa *plainteks*.



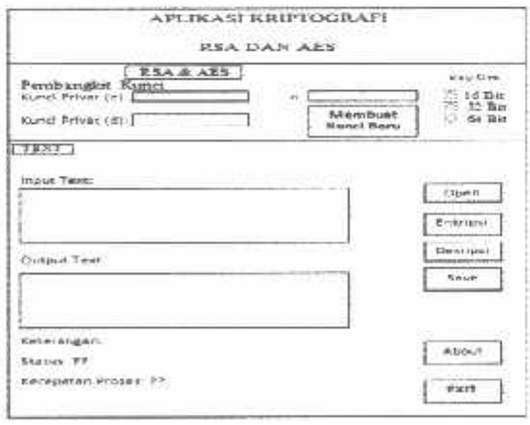
Gambar 3.8 Desain enkripsi dan dekripsi algoritma RSA pada Text



Gambar 3.10 Desain enkripsi dan dekripsi file dokumen algoritma AES

3.2.4 Desain Enkripsi Dan Dekripsi pada *Text* Menggunakan Algoritma Gabungan RSA dan AES

Pada desain ini terdapat pembangkit kunci yang akan digunakan pada proses enkripsi dan dekripsi, kunci yang digunakan ada dua, yaitu kunci privat (e) dan kunci privat (d). selain itu pada desain ini terdapat tombol *open* untuk melakukan panggilan pada data yang akan dienkripsi atau didekripsi dan menampilkan isi data *text* pada kolom *input text*, serta terdapat tombol *save* untuk menyimpan hasil enkripsi atau dekripsi pada *directory*. Pada keterangan bagian bawah terdapat status yang menyatakan proses apa yang dilakukan dan kecepatan proses yang menampilkan waktu pada saat proses enkripsi atau dekripsi berjalan dalam *millisecond* (ms).



Gambar 3.11 Desain enkripsi dan dekripsi pada text dengan algoritma RSA dan AES

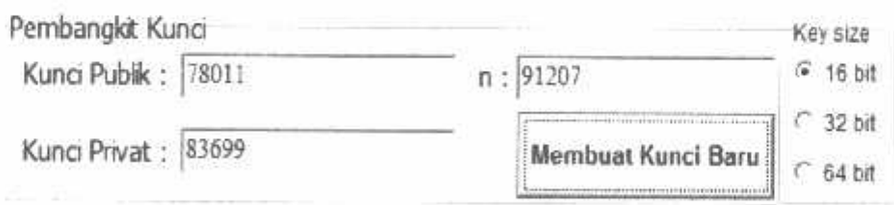
BAB IV
IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi Sistem

Implementasi merupakan proses transformasi rancangan sistem yang diterjemahkan ke dalam bahasa pemrograman yang sesuai dengan sistem yang dipakai. Proses pengembangan aplikasi ini menggunakan Delphi 7 untuk melakukan proses pembangkit kunci bilangan prima, memasukkan teks kedalam program, melakukan proses enkripsi dan dekripsi, serta menyimpan hasil dari enkripsi ataupun dekripsi.

4.1.1 Pembangkit Kunci

Pembangkit kunci ini berfungsi untuk mencari kunci public (e) dan kunci privat (d), serta nilai modulus (n). proses pembangkitan kunci dilakukan secara acak oleh program dengan cara mencari dua nilai prima p dan q yang nilainya tidak sama. Setelah itu dilakukan perhitungan sehingga menghasilkannilai untuk kunci publik, kunci privat dan nilai modulus (n). pada pembangkit kunci terdapat pemilihan panjang kunci yang ingin digunakan mulai dari 16 bit, 32 bit, dan 64 bit. Masing-masing dari panjang bit yang digunakan akan menghasilkan nilai kunci publik, kunci privat dan modulus (n) semakin besar.



Gambar 4.1 Pembangkit kunci RSA

4.1.2 Halaman RSA

Pada halaman RSA ini terdapat pembangkit kunci dan *Tab Sheet* yang didalamnya terdapat halaman enkripsi dan dekripsi pada *text* serta halaman enkripsi dan dekripsi pada file dokumen.

Gambar 4.2 Halaman RSA untuk enkripsi dan dekripsi pada teks

Pada gambar di atas dapat dijelaskan bahwa proses pertama yang dilakukan untuk melakukan proses enkripsi adalah dengan memasukkan kunci publik serta nilai modulus (n). setelah itu memasukkan teks yang akan dilakukan proses enkripsi dengan cara klik tombol *open* dan pilih teks yang akan dienkripsi, teks yang dipilih akan dimunculkan pada kolom *input text*. Tekan tombol enkripsi untuk melakukan proses enkripsi dan hasilnya akan ditampilkan pada kolom *output text*. Pada kolom keterangan akan menampilkan status proses yang dilakukan dan kecepatan proses. Sedangkan pada proses dekripsi hal pertama yang dilakukan adalah memasukkan kunci privat (d) dan modulus (n), setelah itu memasukkan teks dekripsi dengan cara klik *open* dan pilih teks yang akan didekripsi yang isi teksnya akan ditampilkan pada kolom *input text*. Langkah selanjutnya dengan mengklik tombol dekripsi dan hasil dari dekripsi akan ditampilkan pada *output text*. Jika teks hasil dekripsi ingin disimpan, klik tombol *save* dan simpan teks pada *directory* yang di inginkan.

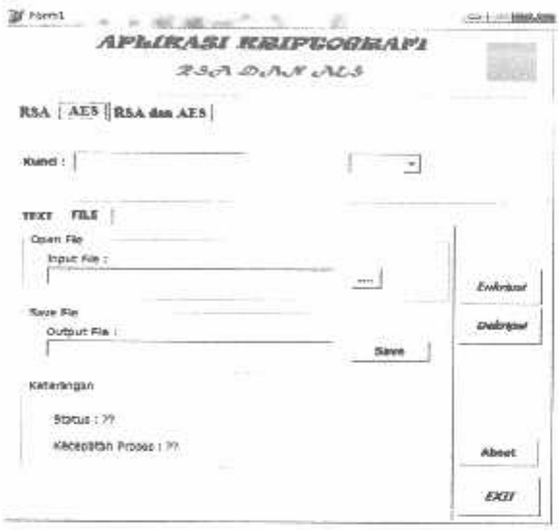
4.1.3 Halaman AES

Pada halaman AES terdapat terdapat *Tab Sheet* yang didalamnya berisi proses enkripsi dan dekripsi untuk teks dan file dokumen. Pada bagian atas terdapat masukan kunci AES serta *combo box* untuk memilih panjang kunci.

Gambar 4.3 Halaman AES untuk enkripsi dan dekripsi teks

pada gambar 4.5 dapat dijelsaikan bahwa untuk proses enkripsi teks pertama masukkn kunci yang akan digunakan serta memilih panjang kunci yang akan digunakan mulai dari 128, 192,dan 256. Setelah itu klik tombol *open* untuk memasukkan teks yang akan di enkripsi dan isi dari teks tersebut akan ditampilkan pada kolom *input text*. Tekan tombol enkripsi untuk memulai proses, hasil dari proses enkripsi berupa teks *cipher* yang akan ditampilkan pada kolom *output text*. Apabila ingin menyimpan teks hasil enkripsi, tekan tombol *save* kemudian simpan teks pada *directory* yang di inginkan. Sedangkan untuk proses dekripsi pertama-tama masukkan kunci yang telah dibuat kemudian pilih panjang kunci yang telah digunakan pada proses enkripsi, bila kunci dan panjang kunci yang digunakan pada saat proses dekripsi tidak sama dengan pada saat proses enkripsi maka teks yang telah dienkripsi tidak bisa didekripsi

lagi karena hasil dari proses dekripsi tidak tidak kembali pada teks aslinya. Untuk itu hasrus dipasitakan bahwa kunci dan panjang kunci yang digunakan harus sama. Selanjutnya masukkan teks yang akan dilakukan proses dekripsi, kemudian klik tombol dekripsi untuk memulai proses dan hasil dari dekripsi akan ditampilkan pada kolom *output text*. Saimpan teks yang telah didekripsi apbila dibutuhkan.

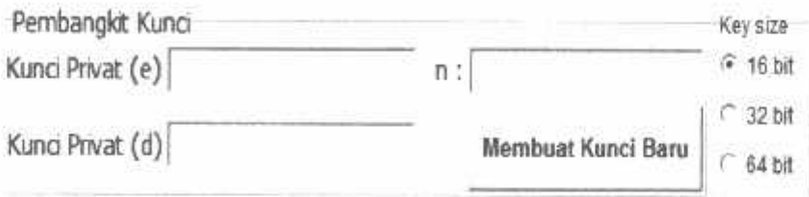


Gambar 4.4 Halaman AES untuk enkripsi dan dekripsi file dokumen

Pada gambar 4.6 dapat dijelaskan bahwa untuk proses enkripsi pada file dokumen langkah pertama yaitu masukkan kunci dan panjang kunci yang akan digunakan, kemudian masukkan file dokumen yang akan dienkripsi dengan cara mengklik tombol (...). Isi dari file dokumen tidak ditampilkan pada program tetapi hanya alamat *directory* saja. Kemudian simpan hasil enkripsi dengan cara mengklik tombol *save*. Langkah selanjutnya dengan mengklik tombol enkripsi untuk memulai proses enkripsi. Pada saat proses sudah selesai maka akan muncul status program dan kecepatan proses yang dibutuhkan pada kolom keterangan bagian bawah.

4.1.4 Halaman Gabungan RSA Dan AES

Pada halaman ini proses enkripsi dan dekripsi algoritma RSA dan AES digabung menjadi satu, sehingga dalam satu kali proses enkripsi dan dekripsi langsung menggunakan dua buah algoritma sekaligus. Kunci yang digunakan untuk proses ini adalah kunci asimetris.



Gambar 4.5 Pembangkit kunci gabungan RSA dan AES

Pada gambar 4.7 dapat dijelaskan bahwa proses pembangkitan kunci untuk algoritma gabungan antara RSA dan AES mirip dengan dengan pembangkitan kunci pada algoritma RSA, yang membedakan adalah kunci yang dihasilkan ada dua kunci privat, yaitu kunci privat (e) dan kunci privat (d), serta modulus (n).

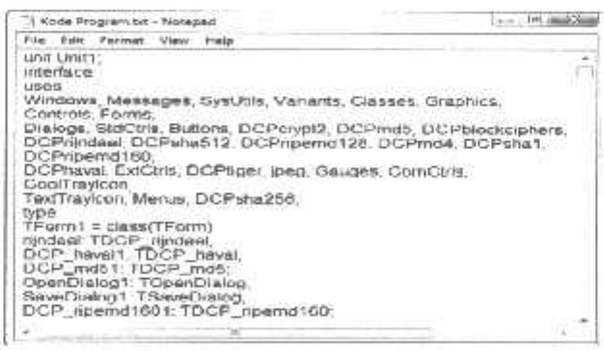
Gambar 4.6 Halaman RSA dan AES untuk enkripsi dan dekripsi teks

Pada gambar 4.8 dapat dijelaskan bahwa pada proses enkripsi hal pertama yang dilakukan adalah memasukkan dua buah kunci privat, yaitu kunci privat (e) dan kunci privat (d), ini karena dalam satu kali proses

enkripsi terdapat dua algoritma yang digunakan, yaitu algoritma RSA dan algoritma AES. Setelah memasukkan kunci langkah selanjutnya adalah memasukkan teks yang akan dienkripsi dengan cara klik tombol *open* kemudian isi dari teks tersebut akan ditampilkan pada kolom *input text*. Langkah berikutnya klik tombol enkripsi untuk memulai proses enkripsi, hasil dari proses enkripsi akan ditampilkan pada kolom *output text*. Apabila ingin menyimpan teks dalam *directory* klik tombol *save*. Sedangkan pada proses dekripsi kunci yang digunakan hanya satu yaitu kunci privat (d). kemudian masukkan teks yang akan dienkripsi dengan cara klik tombol *open* kemudian isi dari tersebut akan ditampilkan pada kolom *input text*. Langkah berikutnya adalah klik timbol dekripsi untuk memulai proses dan hasilnya akan ditampilkan pada kolom *output text*. Apabila ingin menyimpan hasil dekripsi klik tombol *save* kemudian pilih *directory* tempat penyimpanan.

4.2 Pengujian Sistem

Pengujian dilakukan untuk menguji kebenaran dari algoritma RSA dan algoritma AES yang digunakan pada proses enkripsi dan dekripsi. Pengujian dilakukan dengan cara melakukan enkripsi pada teks dan file dokumen sehingga menghasilkan sebuah *cipher*, dari *cipher* tersebut kemudian dilakukan proses dekripsi. Apabila hasil dari dekripsi sama dengan teks atau file dokumen sebelum mengalami proses enkripsi, maka pengujian dikatakan berhasil.


Kode Program.txt - Notepad

File Edit Format View Help

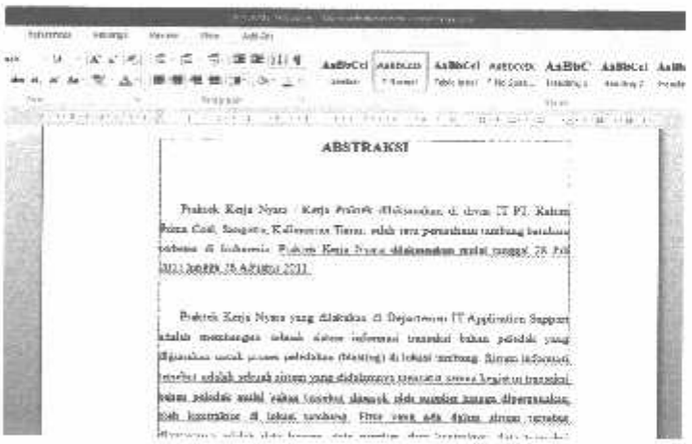
```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
Dialogs, StdCtrls, Buttons, DCPcrypt2, DCPmd5, DCPblockciphers,
DCPrijndael, DCPsha512, DCPripemd128, DCPmd4, DCPsha1,
DCPripemd160,
DCPhaval, ExtCtrls, DCPtiger, Ipeg, Gauges, ComCtrls,
CoolTrayIcon
TextTrayIcon, Menus, DCPsha256,
type
TForm1 = class(TForm)
rijndael: TDCP_rijndael;
DCP_haval1: TDCP_haval;
DCP_md51: TDCP_md5;
OpenDialog1: TOpenDialog;
SaveDialog1: TSaveDialog;
DCP_ripemd1601: TDCP_ripemd160;
```

Gambar 4.7 Teks yang belum dienkripsi

Pada gambar 4.7 merupakan teks asli sebelum dilakukan proses enkripsi sehingga isi dari teks tersebut masih dapat dibaca.

Gambar 4.8 Hasil dari proses enkripsi

Pada gambar di atas merupakan teks yang telah dienkripsi sehingga isi dari teks tersebut tidak bisa dibaca karena isinya berupa kata-kata acak yang tidak bisa dipahami.



Gambar 4.9 File dokumen sebelum dienkripsi

Pada gambar si atas, file dokumen yang belum dilakukan proses enkripsi isi datanya dapat dibaca dan dipahami.

Gambar 4.13 File dokumen setelah dienkripsi

Pada gambar 4.13 merupakan isi dari file dokumen setelah dilakukan proses enkripsi, hasilnya isi dari file dokumen tersebut tidak bisa diba karena merupakan kata-kata acak dan simbol.

Tabel 4.1 Uji coba enkripsi RSA 16 bit dan AES 128 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 16 bit	Text	Kode Program.txt	26 KB	2496 ms	65 KB	150%	60%
		File.txt	9 KB	1607 ms	25 KB	177%	63%
AES 128 bit	Text	Kode Program.txt	26 KB	6022 ms	52 KB	100%	67%
		File.txt	9 KB	1014 ms	17 KB	88%	61%
	File	Laporan PKN.docx	27.911 KB	16006 ms	27.911 KB	0%	85%
		Proposal PKN.doc	80 KB	124 ms	81 KB	1,25%	24%
		Flowchart RSA.odt	16 KB	78 ms	16 KB	0%	23%
RSA & AES	Text	Kode Program.txt	26 KB	11684 ms	155 KB	496,15%	70%
		File.txt	9 KB	6209 ms	49 KB	444,44%	64%

Pada tabel di atas dapat dijelaskan bahwa enkripsi pada teks dengan menggunakan algoritma RSA 16 bit, untuk ukuran file menjadi lebih besar dari file aslinya,perubahan ukuran filenya sendiri mencapai 150% serta meningkatkan pemakaian CPU sebesar 60% - 63%. Untuk proses enkripsi pada teks menggunakan algoritma AES 128 bit, ukuran file

hasil enkripsi lebih besar dari ukuran file asli,perubahan ukuran file mencapai 88% dan 100% sedangkan pemakaian CPU mencapai 61% - 67%. Selain itu pada percobaan enkripsi file dokumen menggunakan algoritma AES, untuk ukuran file setelah proses enkripsi pada file dengan ekstensi *.docx dan *.odt tidak mengalami perubahan, sedangkan untuk file yang berekstensi *.doc mengalami perubahan dari dari 80 KB menjadi 81 KB, perubahan filenya hanya 1,25% saja. Pada proses enkripsi gabungan yaitu RSA dan AES ukuran file teks jauh lebih besar dari file aslinya, perubahan filenya mencapai 444,44% dan 496,15% serta meningkatkan pemakaian CPU antara 64% - 70%.

Tabel 4.2 Uji coba dekripsi RSA 16 bit dan AES 128 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 16 bit	Text	Kode Program1.txt	65 KB	874 ms	26 KB	36%
		File1.txt	25 KB	374 ms	9 KB	20%
AES 128 bit	Text	Kode Program2.txt	52 KB	140 ms	26 KB	15%
		File2.txt	17 KB	62 ms	9 KB	20%
	File	Lap PKN2.doc	27.911 KB	16941 ms	27.911 KB	69%
		Proposal PKN2.doc	81 KB	78 ms	80 KB	9%
		Flowchart RSA2.odt	16 KB	31 ms	16 KB	5%
RSA & AES	Text	Kode Program3.txt	155 KB	3510 ms	26 KB	51%
		File3.txt	49 KB	1045	9 KB	51%

Pada tabel uji coba di atas dapat dijelaskan bahwa dekripsi pada teks menggunakan algoritma RSA untuk ukuran file lebih kecil serta pemakaian CPU hanya sebesar 20% - 36% dan memerlukan waktu yang singkat untuk proses dekripsinya. Untuk proses dekripsi pada teks menggunakan algoritma AES, ukuran file lebih kecil serta pada saat proses dekripsi pemakaian CPU hanya 15% - 20%. Pada percobaan dekripsi menggunakan algoritma AES pada file dokumen, ukuran file untuk enstensi *.docx dan *.odt tidak mengalami perubahan, sedangkan untuk file *.doc hasil dekripsinya memiliki ukuran file lebih lebih kecil. Untuk pemakaian CPU sangat sedikit yaitu 5% - 9%, kecuali dokumen yang

memiliki ukuran file besar, yaitu 69%. Pada percobaan mengunakan algoritma gabungan antara RSA dan AES pada teks, ukuran file yang dihasilkan lebih kecil dengan pemakaian CPU sebesar 51%, dan memerlukan waktu proses sedikit lebih lama.

Tabel 4.3 Uji coba enkripsi RSA 16 bit dan AES 192 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 16 bit	Text	Kode Program.txt	26 KB	2496 ms	65 KB	150%	60%
		File.txt	9 KB	1607 ms	25 KB	177%	63%
AES 192 bit	Text	Kode Program.txt	26 KB	6271 ms	52 KB	100%	74%
		File.txt	9 KB	717 ms	17 KB	88%	25%
	File	Laporan PKN.docx	27.911 KB	18580 ms	27.911 KB	0%	77%
		Proposal PKN.doc	80 KB	78 ms	81 KB	1,25%	24%
		Flowchart RSA.odt	16 KB	47 ms	16 KB	0%	16%
RSA & AES	Text	Kode Program.txt	26 KB	12754 ms	155 KB	496,15%	72%
		File.txt	9 KB	6312 ms	49 KB	444,44%	60%

Pada tabel di atas dapat dijelaskan bahwa enkripsi pada teks dengan menggunakan algoritma RSA 16 bit, untuk ukuran file menjadi lebih besar dari file aslinya,perubahan ukuran filenya sendiri mencapai 150% serta meningkatkan pemakaian CPU sebesar 60% - 63%. Sedangkan untuk proses enkripsi menggunakan algoritma AES 192 bit pada teks ukuran file yang dihasilkan lebih besar dari file asli, perubahan file itu sendiri mencapai 88% dan 100%, selain itu untuk pemakain CPU pada waktu proses berjalan mencapai 25% - 74% dengan waktu yang diperlukan untuk proses mencapai 717 ms – 6271 ms. Sedangkan untuk proses enkripsi pada file dokumen dengan ekstensi file *.docx dan *.odt ukuran filenya tidak mengalami perubahan dengan pemakaian CPU sebesar 16% - 77%, sedangkan waktu yang diperlukan dalam satu kali proses mencapai 18580 ms. Untuk proses enkripsi menggunakan algoritma gabungan RSA dan AES pada teks ukuran file yang dihasilkan jauh lebih besar dari file aslinya, perubahan ukuran filenya mencapai 444,44% dan 496,15% dengan pemakaian CPU sebesar 60% - 72%. Waktu yang diperlukan untuk proses enkripsi ini antara 6312 ms – 12754 ms.

Tabel 4.4 Uji coba dekripsi RSA 16 bit dan AES 192 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 16 bit	Text	Kode Program1.txt	65 KB	874 ms	26 KB	36%
		File1.txt	25 KB	374 ms	9 KB	20%
AES 192 bit	Text	Kode Program2.txt	52 KB	109 ms	26 KB	19%
		File2.txt	17 KB	62 ms	9 KB	15%
	File	Lap PKN2.doc	27.911 KB	19126 ms	27.911 KB	84%
		Proposal PKN2.doc	81 KB	62 ms	80 KB	21%
		Flowchart RSA2.odt	16 KB	16 ms	16 KB	16%
RSA & AES	Text	Kode Program3.txt	155 KB	3728 ms	26 KB	58%
		File3.txt	49 KB	1219 ms	9 KB	52%

Pada tabel uji coba di atas dapat dijelaskan bahwa dekripsi pada teks menggunakan algoritma RSA untuk ukuran file lebih kecil serta pemakaian CPU hanya sebesar 20% - 36% dan memerlukan waktu yang singkat untuk proses dekripsinya. Sedangkan pada algoritmaAES 192 bit pada teks waktu yang diperlukan hanya 62 ms – 109 ms jauh lebih cepat dari pada menggunakan algoritma RSA 16 bit dengan pemakaian CPU hanya 15% - 19%. Sedangkan untuk proses dekripsi pada file dokumen dengan ekstensi *.docx, waktu yang diperlukan cukaup lama yaitu 19126 ms, ini karena file yang akan didekripsi ukurannya sangat besar mencapai 27 MB dengan pemakaian CPU mencapai 84%. Hal ini berbeda dengan file yang berekstensi *.doc dan *odt yang hanya memerlukan waktu sekitar 16 ms – 62 ms dengan pemakaian CPU sebesar 16% - 21%. Untuk proses dekripsi menggunakan algoritma gabungan pada teks, waktu yang diperlukan untuk satu kali proses antara 1219 ms – 3728 ms dengan pemakaian CPU sebesar 52% - 58%.

Tabel 4.5 Uji coba enkripsi RSA 16 bit dan AES 256 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 16 bit	Text	Kode Program.txt	26 KB	2496 ms	65 KB	150%	60%
		File.txt	9 KB	1607 ms	25 KB	177%	63%
AES 256 bit	Text	Kode Program.txt	26 KB	6084 ms	52 KB	100%	60%
		File.txt	9 KB	733 ms	17 KB	88%	31%
	File	Laporan PKN.docx	27.911 KB	18798 ms	27.911 KB	0%	76%
		Proposal PKN.doc	80 KB	109 ms	81 KB	1,25%	30%
		Flowchart RSA.odt	16 KB	62 ms	16 KB	0%	20%
RSA & AES	Text	Kode Program.txt	26 KB	12932 ms	155 KB	496,15%	81%
		File.txt	9 KB	6523 ms	49 KB	444,44%	65%

Pada tabel di atas dapat dijelaskan bahwa enkripsi pada teks dengan menggunakan algoritma RSA 16 bit, untuk ukuran file menjadi lebih besar dari file aslinya,perubahan ukuran filenya sendiri mencapai 150% serta meningkatkan pemakaian CPU sebesar 60% - 63%. Untuk proses enkripsi menggunakan algoritma AES 256 bit pada teks ukuran file yang dihasilkan lebih besar yaitu mencapai 88% dan 100% dengan waktu yang dibutuhkan sekitar 733 ms – 6084 ms, untuk pemakaian CPU sendiri mencapai 60%. Pada proses enkripsi file dokumen ukuran file yang dihasilkan tidak mengalami perubahan kecuali file dengan ekstensi *.doc yang mengalami perubahan file sebesar 1,25%. Waktu yang diperlukan untuk satu kali proses antara 62 ms – 109 ms dengan pemakaian CPU hanya 20% - 30%, kecuali untuk proses enkripsi dengan ukuran file yang besar waktu yang diperlukan mencapai 18798 ms dengan pemakaian CPU sebesar 76%. Pada proses enkripsi gabungan yaitu RSA dan AES ukuran file teks jauh lebih besar dari file aslinya, perubahan filenya mencapai 444,44% dan 496,15% serta meningkatkan pemakaian CPU antara 65% - 81%. Selain itu waktu yang dibutuhkan antara 6523 ms – 12932 ms.

Tabel 4.6 Uji coba dekripsi RSA 16 bit dan AES 256 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 16 bit	Text	Kode Program1.txt	65 KB	874 ms	26 KB	36%
		File1.txt	25 KB	374 ms	9 KB	20%
AES 256 bit	Text	Kode Program2.txt	52 KB	110 ms	26 KB	14%
		File2.txt	17 KB	46 ms	9 KB	24%
	File	Lap PKN2.doc	27.911 KB	17176 ms	27.911 KB	80%
		Proposal PKN2.doc	81 KB	78 ms	80 KB	17%
		Flowchart RSA2.odt	16 KB	140 ms	16 KB	24%
RSA & AES	Text	Kode Program3.txt	155 KB	3790 ms	26 KB	62%
		File3.txt	49 KB	1045 ms	9 KB	50%

Pada tabel uji coba di atas dapat dijelaskan bahwa dekripsi pada teks menggunakan algoritma RSA 16 bit untuk ukuran file lebih kecil serta pemakaian CPU hanya sebesar 20% - 36% dan memerlukan waktu yang singkat untuk proses dekripsinya. Untuk proses dekirpsi menggunakan algoritma AES 256 bit pada teks waktu yang diperlukan sangat singkat hanya 46 ms – 110 ms dengan pemakaian CPU sebesar 14% - 24%, selain itu pada file dokumen waktu yang diperlukan untuk proses dekripsi pada file berukuran kecil hanya 78 ms – 140 ms dengan pemakaian CPU sebesar 17% - 24%, sedangkan pada file yang berukuran besar waktu yang diperlukan mencapai 17176 ms dengan pemakaian CPU sebesar 80%. Untuk proses dekripsi menggunakan algoritma gabungan RSA dan AES pada teks, waktu yang dibutuhkan untuk satu kali proses mencapai 1045 ms – 3790 ms dengan pemakaian CPU sebesar 50% - 62%.

Tabel 4.7 Uji coba enkripsi RSA 32 bit dan AES 128 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 32 bit	Text	Kode Program.txt	26 KB	4524 ms	65 KB	150%	63%
		File.txt	9 KB	1435 ms	21 KB	133,33%	67%
AES 128 bit	Text	Kode Program.txt	26 KB	6022 ms	52 KB	100%	67%
		File.txt	9 KB	1014 ms	17 KB	88%	61%
	File	Laporan PKN.docx	27.911 KB	16006 ms	27.911 KB	0%	85%
		Proposal PKN.doc	80 KB	124 ms	81 KB	1,25%	24%
		Flowchart RSA.odt	16 KB	78 ms	16 KB	0%	23%
RSA & AES	Text	Kode Program.txt	26 KB	9253 ms	129 KB	396,15%	68%
		File.txt	9 KB	3525 ms	41 KB	355,55%	60%

Pada tabel 4.7 dapat dijelaskan bahwa untuk proses enkripsi menggunakan algoritma RSA dengan panjang kunci 32 bit pada teks ukuran file yang dihasilkan lebih besar dari file asli, perubahan filenya mencapai 150%, sedangkan pemakaian CPU pada saat proses berjalan mencapai 63%, serta memerlukan waktu sekitar 1435 ms – 4524 ms. Untuk proses enkripsi pada teks menggunakan algoritma AES, ukuran file hasil enkripsi lebih besar dari ukuran file asli, perubahan ukuran file mencapai 88% dan 100% sedangkan pemakaian CPU mencapai 61% - 67%. Selain itu pada percobaan enkripsi file dokumen menggunakan algoritma AES, untuk ukuran file setelah proses enkripsi pada file dengan ekstensi *.docx dan *.odt tidak mengalami perubahan, sedangkan untuk file yang berekstensi *.doc mengalami perubahan dari dari 80 KB menjadi 81 KB, perubahan filenya hanya 1,25% saja. Pada proses enkripsi gabungan yaitu RSA dan AES ukuran file teks jauh lebih besar dari file aslinya, perubahan filenya mencapai 355,55% dan 396,15% serta meningkatkan pemakaian CPU antara 60% - 68%.

Tabel 4.8 Uji coba dekripsi RSA 32 bit dan AES 128 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 32 bit	Text	Kode Program1.txt	65 KB	3510 ms	26 KB	64%
		File1.txt	21 KB	1061 ms	9 KB	61%
AES 128 bit	Text	Kode Program2.txt	52 KB	140 ms	26 KB	15%
		File2.txt	17 KB	62 ms	9 KB	20%
	File	Lap PKN2.doc	27.911 KB	16941 ms	27.911 KB	69%
		Proposal PKN2.doc	81 KB	78 ms	80 KB	9%
		Flowchart RSA2.odt	16 KB	31 ms	16 KB	5%
RSA & AES	Text	Kode Program3.txt	129 KB	2425 ms	26 KB	64%
		File3.txt	41 KB	1039 ms	9 KB	46%

Pada tabel dekripsi di atas dapat dijelaskan bahwa untuk proses dekripsi menggunakan algoritma RSA 32 bit waktu yang diperlukan mencapai 1061 ms – 3510 ms dengan pemakaian CPU sebesar 61% - 64%. Ukuran file yang dihasilkan berupa ukuran file asli pada saat sebelum

proses enkripsi. Untuk proses dekripsi pada teks menggunakan algoritma AES, ukuran file lebih kecil serta pada saat proses dekripsi pemakaian CPU hanya 15% - 20%. Pada percobaan dekripsi menggunakan algoritma AES 128 bit pada file dokumen, ukuran file untuk enstensi *.docx dan *.odt tidak mengalami perubahan, sedangkan untuk file *.doc hasil dekripsinya memiliki ukuran file lebih lebih kecil. Untuk pemakaian CPU sangat sedikit yaitu 5% - 9%, kecuali dokumen yang memiliki ukuran file besar, yaitu 69%. Sedangkan pada algoritma gabungan RSA dan AES pada teks, ukuran file yang dihasilkan lebih kecil dengan pemakaian CPU sebesar 51% dan memerlukan waktu proses sedikit lebih lama, tetapi membutuhkan waktu antara 1039 ms – 2425 ms.

Tabel 4.9 Uji coba enkripsi RSA 32 bit dan AES 192 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 32 bit	Text	Kode Program.txt	26 KB	4524 ms	65 KB	150%	63%
		File.txt	9 KB	1435 ms	21 KB	133,33%	67%
AES 192 bit	Text	Kode Program.txt	26 KB	6271 ms	52 KB	100%	74%
		File.txt	9 KB	717 ms	17 KB	88%	25%
	File	Laporan PKN.docx	27.911 KB	18580 ms	27.911 KB	0%	77%
		Proposal PKN.doc	80 KB	78 ms	81 KB	1,25%	24%
		Flowchart RSA.odt	16 KB	47 ms	16 KB	0%	16%
RSA & AES	Text	Kode Program.txt	26 KB	9719 ms	129 KB	396,15%	70%
		File.txt	9 KB	4555 ms	41 KB	355,55%	67%

Pada tabel di atas dapat dijelaskan bahwa untuk proses enkripsi menggunakan algoritma RSA dengan panjang kunci 32 bit pada teks ukuran file yang dihasilkan lebih besar dari file asli, perubahan filenya mencapai 150%, sedangkan pemakaian CPU pada saat proses berjalan mencapai 63%, serta memerlukan waktu sekitar 1435 ms – 4524 ms. Untuk enkripsi menggunakan algoritma AES dengan panjang kunci 192 bit pada teks ukuran file yang dihasilkan lebih besar dari file asli, perubahan file itu sendiri mencapai 88% dan 100%, selain itu untuk

pemakain CPU pada waktu proses berjalan mencapai 25% - 74% dengan waktu yang diperlukan untuk proses mencapai 717 ms – 6271 ms. Sedangkan untuk proses enkripsi pada file dokumen dengan ekstensi file *.docx dan *.odt ukuran filenya tidak mengalami perubahan dengan pemakaian CPU sebesar 16% - 77%, sedangkan waktu yang diperlukan dalam satu kali proses mencapai 18580 ms. Untuk proses enkripsi menggunakan algoritma gabungan RSA dan AES pada teks ukuran file yang dihasilkan jauh lebih besar dari file aslinya, perubahan ukuran filenya mencapai 355,55% dan 396,15% dengan pemakaian CPU sebesar 67% - 70%. Waktu yang diperlukan untuk proses enkripsi ini antara 4555 ms – 9719 ms.

Tabel 4.10 Uji coba dekripsi RSA 32 bit dan AES 192 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 32 bit	Text	Kode Program1.txt	65 KB	3510 ms	26 KB	64%
		File1.txt	21 KB	1061 ms	9 KB	61%
AES 192 bit	Text	Kode Program2.txt	52 KB	109 ms	26 KB	19%
		File2.txt	17 KB	62 ms	9 KB	15%
	File	Lap PKN2.doc	27.911 KB	19126 ms	27.911 KB	84%
		Proposal PKN2.doc	81 KB	62 ms	80 KB	21%
		Flowchart RSA2.odt	16 KB	16 ms	16 KB	16%
RSA & AES	Text	Kode Program3.txt	129 KB	3229 ms	26 KB	71%
		File3.txt	41 KB	1077 ms	9 KB	52%

Pada tabel dekripsi di atas dapat dijelaskan bahwa untuk proses dekripsi menggunakan algoritma RSA 32 bit waktu yang diperlukan mencapai 1062 ms – 3510 ms dengan pemakaian CPU sebesar 61% - 64%. Ukuran file yang dihasilkan berupa ukuran file asli pada saat sebelum proses enkripsi. Untuk proses dekripsi menggunakan algoritma AES 192 bit pada teks waktu yang diperlukan hanya 62 ms – 109 ms jauh lebih cepat dari pada menggunakan algoritma RSA 32 bit dengan pemakaian CPU hanya 15% - 19%. Sedangkan untuk proses dekripsi pada file

dokumen dengan ekstensi *.docx, waktu yang diperlukan cukaup lama yaitu 19126 ms, ini karena file yang akan didekripsi ukurannya sangat besar mencapai 27 MB dengan pemakaian CPU mencapai 84%. Hal ini berbeda dengan file yang berekstensi *.doc dan *odt yang hanya memerlukan waktu sekitar 16 ms – 62 ms dengan pemakaian CPU sebesar 16% - 21%. Untuk proses dekripsi menggunakan algoritma gabungan pada teks, waktu yang diperlukan untuk satu kali proses antara 1077 ms – 3229 ms dengan pemakaian CPU sebesar 52% - 71%.

Tabel 4.11 Uji coba enkripsi RSA 32 bit dan AES 256 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 32 bit	Text	Kode Program.txt	26 KB	2496 ms	65 KB	150%	60%
		File.txt	9 KB	1607 ms	25 KB	177%	63%
AES 256 bit	Text	Kode Program.txt	26 KB	6084 ms	52 KB	100%	60%
		File.txt	9 KB	733 ms	17 KB	88%	31%
	File	Laporan PKN.docx	27.911 KB	18798 ms	27 911 KB	0%	76%
		Proposal PKN.doc	80 KB	109 ms	81 KB	1,25%	30%
		Flowchart RSA.odt	16 KB	62 ms	16 KB	0%	20%
RSA & AES	Text	Kode Program.txt	26 KB	9845 ms	129 KB	396,15%	72%
		File.txt	9 KB	5795 ms	41 KB	355,55%	67%

Pada tabel 4.11 dapat dijelaskan bahwa untuk proses enkripsi menggunakan algoritma RSA dengan panjang kunci 32 bit pada teks ukuran file yang dihasilkan lebih besar dari file asli, perubahan filenya mencapai 150%, sedangkan pemakaian CPU pada saat proses berjalan mencapai 63%, serta memerlukan waktu sekitar 1435 ms – 4524 ms. Untuk proses enkripsi menggunakan algoritma AES 256 bit pada teks ukuran file yang dihasilkan lebih besar yaitu mencapai 88% dan 100% dengan waktu yang dibutuhkan sekitar 733 ms – 6084 ms, untuk pemakaian CPU sendiri mencapai 60%. Pada proses enkripsi file dokumen ukuran file yang dihasilkan tidak mengalami perubahan kecuali file dengan ekstensi *.doc yang mengalami perubahan file sebesar 1,25%. Waktu yang diperlukan untuk satu kali proses antara 62 ms – 109 ms dengan pemakaian CPU hanya 20% - 30%, kecuali untuk proses enkripsi

dengan ukuran file yang besar waktu yang diperlukan mencapai 18798 ms dengan pemakaian CPU sebesar 76%. Sedangkan untuk enkripsi gabungan RSA dan AES file teks yang dihasilkan sangat besar mencapai 396,15% dari file asli dengan waktu enkripsi 5795 ms – 9845 ms, serta pemakaian CPU yang diperlukan mencapai 67% - 72%.

Tabel 4.12 Uji coba dekripsi RSA 32 bit dan AES 256 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 32 bit	Text	Kode Program1.txt	65 KB	3510 ms	26 KB	64%
		File1.txt	21 KB	1061 ms	9 KB	61%
AES 256 bit	Text	Kode Program2.txt	52 KB	110 ms	26 KB	14%
		File2.txt	17 KB	46 ms	9 KB	24%
	File	Lap PKN2.doc	27.911 KB	17176 ms	27.911 KB	80%
		Proposal PKN2.doc	81 KB	78 ms	80 KB	17%
		Flowchart RSA2.odt	16 KB	140 ms	16 KB	24%
RSA & AES	Text	Kode Program3.txt	129 KB	3954 ms	26 KB	72%
		File3.txt	41 KB	1282 ms	9 KB	54%

Pada tabel dekripsi di atas dapat dijelaskan bahwa untuk proses dekripsi menggunakan algoritma RSA 32 bit waktu yang diperlukan mencapai 1061 ms – 3510 ms dengan pemakaian CPU sebesar 61% - 64%. Ukuran file yang dihasilkan berupa ukuran file asli pada saat sebelum proses enkripsi. Untuk proses dekirpsi menggunakan algoritma AES 256 bit pada teks waktu yang diperlukan sangat singkat hanya 46 ms – 110 ms dengan pemakaian CPU sebesar 14% - 24%, selain itu pada file dokumen waktu yang diperlukan untuk proses dekripsi pada file berukuran kecil hanya 78 ms – 140 ms dengan pemakaian CPU sebesar 17% - 24%, sedangkan pada file yang berukuran besar waktu yang diperlukan mencapai 17176 ms dengan pemakaian CPU sebesar 80%. Sedangkan untuk proses dekripsi menggunakan algoritma gabungan RSA dan AES pada teks, waktu yang dibutuhkan untuk satu kali proses mencapai 1282 ms – 3954 ms dengan pemakaian CPU sebesar 54% - 72%.

Tabel 4. 13 Uji coba enkripsi RSA 64 bit dan AES 128 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 64 bit	Text	Kode Program.txt	26 KB	7551 ms	70 KB	169,23%	66%
		File.txt	9 KB	2434 ms	22 KB	144,44%	70%
AES 128 bit	Text	Kode Program.txt	26 KB	6022 ms	52 KB	100%	67%
		File.txt	9 KB	1014 ms	17 KB	88%	61%
	File	Laporan PKN.docx	27.911 KB	16006 ms	27.911 KB	0%	85%
		Proposal PKN.doc	80 KB	124 ms	81 KB	1,25%	24%
		Flowchart RSA.odt	16 KB	78 ms	16 KB	0%	23%
RSA & AES	Text	Kode Program.txt	26 KB	9875 ms	140 KB	438,46%	69%
		File.txt	9 KB	4742 ms	44 KB	388,88%	68%

Pada tabel uji coba di atas dapat dijelaskan bahwa untuk proses enkripsi menggunakan algoritma RSA 64 bit ukuran file yang dihasilkan jauh lebih besar dari file aslinya, perubahan file itu sendiri mencapai 144,44% dan 169,23% sedangkan waktu yang diperlukan untuk satu kali proses mencapai 2434 ms – 7551 ms dengan pemakaian CPU sebesar 66% - 70%. Untuk proses enkripsi pada teks menggunakan algoritma AES, ukuran file hasil enkripsi lebih besar dari ukuran file asli, perubahan ukuran file mencapai 88% dan 100% sedangkan pemakaian CPU mencapai 61% - 67%. Selain itu pada percobaan enkripsi file dokumen menggunakan algoritma AES, untuk ukuran file setelah proses enkripsi pada file dengan ekstensi *.docx dan *.odt tidak mengalami perubahan, sedangkan untuk file yang berekstensi *.doc mengalami perubahan dari dari 80 KB menjadi 81 KB, perubahan filenya hanya 1,25% saja. Pada proses enkripsi gabungan yaitu RSA dan AES ukuran file teks jauh lebih besar dari file aslinya, perubahan filenya mencapai 388,88% dan 438,46% serta meningkatkan pemakaian CPU antara 68% - 69%.

Tabel 4.14 Uji coba dekripsi RSA 64 bit dan AES 128 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 64 bit	Text	Kode Program1.txt	70 KB	6038 ms	26 KB	66%
		File1.txt	22 KB	1888 ms	9 KB	62%
AES 128 bit	Text	Kode Program2.txt	52 KB	140 ms	26 KB	15%
		File2.txt	17 KB	62 ms	9 KB	20%
	File	Lap PKN2.doc	27.911 KB	16941 ms	27.911 KB	69%
		Proposal PKN2.doc	81 KB	78 ms	80 KB	9%
		Flowchart RSA2.odt	16 KB	31 ms	16 KB	5%
RSA & AES	Text	Kode Program3.txt	140 KB	4721 ms	26 KB	61%
		File3.txt	44 KB	1279 ms	9 KB	51%

Pada tabel dekripsi di atas dapat dijelaskan bahwa untuk proses dekripsi menggunakan algoritma RSA 64 bit waktu yang diperlukan mencapai 1888 ms – 6038 ms dengan pemakaian CPU sebesar 62% - 66%. Untuk proses dekripsi pada teks menggunakan algoritma AES 128 bit, ukuran file lebih kecil serta pada saat proses dekripsi pemakaian CPU hanya 15% - 20%. Pada percobaan dekripsi menggunakan algoritma AES pada file dokumen, ukuran file untuk enstensi *.docx dan *.odt tidak mengalami perubahan, sedangkan untuk file *.doc hasil dekripsinya memiliki ukuran file lebih lebih kecil. Untuk pemakaian CPU sangat sedikit yaitu 5% - 9%, kecuali dokumen yang memiliki ukuran file besar, yaitu 69%. Pada percobaan mengunakan algoritma gabungan antara RSA dan AES pada teks, ukuran file yang dihasilkan lebih kecil dengan pemakaian CPU sebesar 51% dan 61% dan memerlukan waktu proses sedikit lebih lama, yaitu antara 1279 ms – 4721 ms.

Tabel 4.15 Uji coba enkripsi RSA 64 bit dan AES 192 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 64 bit	Text	Kode Program.txt	26 KB	7551 ms	70 KB	169,23%	66%
		File.txt	9 KB	2434 ms	22 KB	144,44%	70%
AES 192 bit	Text	Kode Program.txt	26 KB	6271 ms	52 KB	100%	74%
		File.txt	9 KB	717 ms	17 KB	88%	25%
	File	Laporan PKN.docx	27.911 KB	18580 ms	27.911 KB	0%	77%
		Proposal PKN.doc	80 KB	78 ms	81 KB	1,25%	24%
		Flowchart RSA.odt	16 KB	47 ms	16 KB	0%	16%
RSA & AES	Text	Kode Program.txt	26 KB	10239 ms	140 KB	438,46%	72%
		File.txt	9 KB	6152 ms	44 KB	388,88%	61%

Pada tabel 4.15 dapat dijelaskan bahwa untuk proses enkripsi menggunakan algoritma RSA 64 bit ukuran file yang dihasilkan jauh lebih besar dari file aslinya, perubahan file itu sendiri mencapai 144,44% dan 169,23% sedangkan waktu yang diperlukan untuk satu kali proses mencapai 2434 ms – 7551 ms dengan pemakaian CPU sebesar 66% - 70%. Untuk enkripsi menggunakan algoritma AES dengan panjang kunci 192 bit pada teks ukuran file yang dihasilkan lebih besar dari file asli, perubahan file itu sendiri mencapai 88% dan 100%, selain itu untuk pemakain CPU pada waktu proses berjalan mencapai 25% - 74% dengan waktu yang diperlukan untuk proses mencapai 717 ms – 6271 ms. Sedangkan untuk proses enkripsi pada file dokumen dengan ekstensi file *.docx dan *.odt ukuran filenya tidak mengalami perubahan dengan pemakaian CPU sebesar 16% - 77%, sedangkan waktu yang diperlukan dalam satu kali proses mencapai 18580 ms. Untuk proses enkripsi menggunakan algoritma gabungan RSA dan AES pada teks ukuran file yang dihasilkan jauh lebih besar dari file aslinya, perubahan ukuran filenya mencapai 388,88% dan 438,46% dengan pemakaian CPU sebesar 61% - 72%. Waktu yang diperlukan untuk proses enkripsi ini antara 6152 ms – 10239 ms.

Tabel 4.16 Uji coba dekripsi RSA 64 bit dan AES 192 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 64 bit	Text	Kode Program1.txt	70 KB	6038 ms	26 KB	66%
		File1.txt	22 KB	1888 ms	9 KB	62%
AES 192 bit	Text	Kode Program2.txt	52 KB	109 ms	26 KB	19%
		File2.txt	17 KB	62 ms	9 KB	15%
	File	Lap PKN2.doc	27 911 KB	19126 ms	27.911 KB	84%
		Proposal PKN2.doc	81 KB	62 ms	80 KB	21%
		Flowchart RSA2.odt	16 KB	16 ms	16 KB	16%
RSA & AES	Text	Kode Program3.txt	140 KB	4943 ms	26 KB	64%
		File3.txt	44 KB	1592 ms	9 KB	57%

Pada tabel 4.16 dapat dijelaskan bahwa untuk proses dekripsi menggunakan algoritma RSA waktu yang diperlukan mencapai 1888 ms – 6038 ms dengan pemakaian CPU sebesar 62% - 66%. Untuk proses dekripsi menggunakan algoritma AES 192 bit pada teks waktu yang diperlukan hanya 62 ms – 109 ms jauh lebih cepat dari pada menggunakan algoritma RSA 64 bit dengan pemakaian CPU hanya 15% - 19%. Sedangkan untuk proses dekripsi pada file dokumen dengan ekstensi *.docx, waktu yang diperlukan cukaup lama yaitu 19126 ms, ini karena file yang akan didekripsi ukurannya sangat besar mencapai 27 MB dengan pemakaian CPU mencapai 84%. Hal ini berbeda dengan file yang berekstensi *.doc dan *odt yang hanya memerlukan waktu sekitar 16 ms – 62 ms dengan pemakaian CPU sebesar 16% - 21%. Untuk proses dekripsi menggunakan algoritma gabungan pada teks, waktu yang diperlukan untuk satu kali proses antara 1592 ms – 4943 ms dengan pemakaian CPU sebesar 57% - 64%.

Tabel 4.17 Uji coba enkripsi RSA 64 bit dan AES 256 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	Perubahan ukuran file	CPU Usage
RSA 64 bit	Text	Kode Program.txt	26 KB	7551 ms	70 KB	169,23%	66%
		File.txt	9 KB	2434 ms	22 KB	144,44%	70%
AES 256 bit	Text	Kode Program.txt	26 KB	6084 ms	52 KB	100%	60%
		File.txt	9 KB	733 ms	17 KB	88%	31%
	File	Laporan PKN.docx	27.911 KB	18798 ms	27.911 KB	0%	76%
		Proposal PKN.doc	80 KB	109 ms	81 KB	1,25%	30%
		Flowchart RSA.odt	16 KB	62 ms	16 KB	0%	20%
RSA & AES	Text	Kode Program.txt	26 KB	12730 ms	140 KB	438,46%	75%
		File.txt	9 KB	6162 ms	44 KB	388,88%	63%

Pada tabel uji coba di atas dapat dijelaskan bahwa untuk proses enkripsi menggunakan algoritma RSA 64 bit ukuran file yang dihasilkan jauh lebih besar dari file aslinya, perubahan file itu sendiri mencapai 144,44% dan 169,23% sedangkan waktu yang diperlukan untuk satu kali proses mencapai 2434 ms – 7551 ms dengan pemakaian CPU sebesar 66%

- 70%. Untuk proses enkripsi menggunakan algoritma AES 256 bit pada teks ukuran file yang dihasilkan lebih besar yaitu mencapai 88% dan 100% dengan waktu yang dibutuhkan sekitar 733 ms – 6084 ms, untuk pemakaian CPU sendiri mencapai 60%. Pada proses enkripsi file dokumen ukuran file yang dihasilkan tidak mengalami perubahan kecuali file dengan ekstensi *.doc yang mengalami perubahan file sebesar 1,25%. Waktu yang diperlukan untuk satu kali proses antara 62 ms – 109 ms dengan pemakaian CPU hanya 20% - 30%, kecuali untuk proses enkripsi dengan ukuran file yang besar waktu yang diperlukan mencapai 18798 ms dengan pemakaian CPU sebesar 76%. Untuk enkripsi gabungan RSA dan AES file yang dihasilkan sangat besar mencapai 438,46% dari file asli dengan waktu enkripsi 6162 ms – 12730 ms, pemakaian CPU yang diperlukan mencapai 63% - 75%.

Tabel 4.18 Uji coba dekripsi RSA 64 bit dan AES 256 bit

		Nama file	Ukuran awal file	Waktu	Ukuran akhir file	CPU Usage
RSA 64 bit	Text	Kode Program1.txt	70 KB	6038 ms	26 KB	66%
		File1.txt	22 KB	1888 ms	9 KB	62%
AES 256 bit	Text	Kode Program2.txt	52 KB	110 ms	26 KB	14%
		File2.txt	17 KB	46 ms	9 KB	24%
	File	Lap PKN2.doc	27.911 KB	17176 ms	27.911 KB	80%
		Proposal PKN2.doc	81 KB	78 ms	80 KB	17%
		Flowchart RSA2.odt	16 KB	140 ms	16 KB	24%
RSA & AES	Text	Kode Program3.txt	140 KB	5647 ms	26 KB	65%
		File3.txt	44 KB	1856 ms	9 KB	51%

Pada tebel dekripsi di atas dapat dijelaskan bahwa untuk proses dekripsi menggunakan algoritma RSA waktu yang diperlukan mencapai 1888 ms – 6038 ms dengan pemakaian CPU sebesar 62% - 66%. Untuk proses dekirpsi menggunakan algoritma AES pada teks waktu yang diperlukan sangat singkat hanya 46 ms – 110 ms dengan pemakaian CPU sebesar 14% - 24%, selain itu pada file dokumen waktu yang diperlukan untuk proses dekripsi pada file berukuran kecil hanya 78 ms – 140 ms

dengan pemakaian CPU sebesar 17% - 24%, sedangkan pada file yang berukuran besar waktu yang diperlukan mencapai 17176 ms dengan pemakaian CPU sebesar 80%. Untuk proses dekripsi menggunakan algoritma gabungan RSA dan AES pada teks, waktu yang dibutuhkan untuk satu kali proses mencapai 1856 ms – 5647 ms dengan pemakaian CPU sebesar 51% - 65%.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan pembahasan dan implementasi dari bab-bab sebelumnya, maka dapat diambil kesimpulan sebagai berikut:

1. Proses enkripsi dapat dilakukan dengan menggunakan dua algoritma yang berbeda yaitu algoritma simetris dan algoritma asimetris.
2. Proses enkripsi menggunakan algoritma RSA *text* yang dihasilkan lebih besar dari *text* asli karena algoritma RSA menggunakan proses perpangkatan pada nilai-nilai dari setiap karakter yang akan dienkripsi.
3. Proses enkripsi menggunakan algoritma AES, *text* yang dihasilkan lebih besar dari *text* asli karena adanya penambahan *header* yang berisi informasi ekstensi file.
4. File *output* yang dihasilkan dari proses enkripsi menggunakan algoritma RSA berupa (*.txt).
5. File *output* yang dihasilkan dari proses enkripsi menggunakan algoritma AES berupa (*.txt, *doc, *odt).
6. File *output* yang dihasilkan dari proses enkripsi menggunakan algoritma gabungan RSA dan AES berupa (*.txt).
7. Lama proses enkripsi tergantung dari besar file dan panjang kunci yang digunakan.
8. Pada proses enkripsi menggunakan algoritma gabungan RSA dan AES waktu yang diperlukan lebih lama karena penggunaan dua proses sekaligus.
9. Kombinasi antara algoritma RSA dan algoritma AES yang lebih bagus adalah kombinasi algoritma RSA dengan panjang kunci 32 Bit dan algoritma AES dengan panjang kunci 128 bit karena waktu untuk

proses enkripsi lebih singkat antara 3525 ms - 9253 ms dengan CPU Usage yang digunakan hanya 60% - 68%, sedangkan untuk proses dekripsi waktu yang dibutuhkan antara 1039 ms – 2425 ms dengan pemakaian CPU Usage yang digunakan sebesar 46% - 64%.

5.2 Saran

Berikut adalah saran-saran untuk pengembangan lebih lanjut tentang aplikasi yang dibuat:

1. Untuk kedepannya diharapkan dapat dilakukan perbaikan-perbaikan terhadap sistem yang telah dibuat terutama untuk penambahan kompresi file sehingga file setelah proses enkripsi ukurannya lebih kecil dari file asli.
2. Pada algoritma RSA, serta gabungan algoritma RSA dan AES kedepannya dapat melakukan proses enkripsi dan dekripsi pada file dokumen dengan ekstensi (*doc, *.docx, *odt).
3. Diharapkan aplikasi ini bisa melakukan proses enkripsi dan dekripsi tidak hanya pada file teks saja, tetapi juga bisa digunakan untuk melakukan proses enkripsi dan dekripsi pada video, gambar, atau mungkin folder dan drive.

DAFTAR PUSTAKA

[1] Ariyus, Dony. 2008. *Pengantar Ilmu Kriptografi Teori, Analisis, dan Implementasi*. Yogyakarta: Andi Offset.

[2] Hartono. 2007. *Perancangan Aplikasi Kriptografi Advanced Encryption Standard*. Jurnal Teknik Informatika.

[3] Kurniasari Novi. 2011. Penggunaan Metode RSA Untuk Enkripsi Dan Dekripsi File Teks. Malang: Skripsi.

[4] Malik, Jamaludin, Jaja. 2010. *Mudah belajar membuat aplikasi pemrograman Delphi*. Yogyakarta: Andi.

[5] MADCOMS. 2003. *Pemrograman Borland Delphi 7 (Jilid 1)*. Yogyakarta: Andi Offset.

[6] Sadikin, Rifki. 2012. *Kriptografi Untuk Kemanan Jaringan*. Yogyakarta: Andi Offset.

[7] Wikipedia. 2011. Algoritma RSA. Diakses pada http://id.wikipedia.org/wiki/RSA tanggal 14 April 2012 jam 15.00.

[8] Wikipedia. 2012. Kriptografi. Diakses pada http://id.wikipedia.org/wiki/Kriptografi tanggal 15 April 2012 jam 12.30.

INSTITUT TEKNOLOGI NASIONAL
Jl. Raya Karanglo, KM 2
MALANG

FORMULIR BIMBINGAN SKRIPSI

Nama : Moh. Abdul Hamid
Nim : 08.18.152
Masa Bimbingan : 28 April 2012 s/d 28 Oktober 2012
Judul Skripsi : **APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT**

No	Tanggal	Uraian	Paraf
1	03-07-2012	Konsultasi Bab I, II	
2	10-07-2012	Revisi Bab I, II	
3	19-07-2012	Konsultasi Bab III , IV	
4	25-07-2012	Revisi Bab III, IV	
5	28-07-2012	Konsultasi Bab V, Makalah Seminar Hasil	
6	30-07-2012	Revisi Bab V	
7	02-08-2012	Revisi Makalah Seminar Hasil	

Malang, 02 Agustus 2012
Dosen Pembimbing I

Ir. Sentot Achmadi, MSi
NIP.Y. 1039500281

Form S-4b

INSTITUT TEKNOLOGI NASIONAL
Jl. Raya Karanglo, KM 2
MALANG

FORMULIR BIMBINGAN SKRIPSI

Nama : Moh. Abdul Hamid
Nim : 08.18.152
Masa Bimbingan : 28 April 2012 s/d 28 Oktober 2012
Judul Skripsi : **APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT**

No	Tanggal	Uraian	Paraf
1	03-07-2012	Demo Program, Konsultasi Bab I, II	
2	10-07-2012	Revisi Bab I, II	
3	19-07-2012	Konsultasi Program, Bab III, IV	
4	25-07-2012	Revisi Bab III, IV	
5	28-07-2012	Konsultasi Bab V, Makalah Seminar Hasil	
6	30-07-2012	Revisi Bab V	
7	02-08-2012	Revisi Makalah Seminar Hasil dan Program	

Malang, 02 Agustus 2012
Dosen Pembimbing II

Yosep Agus Pranoto, ST
NIP. P. 1031000432

Form S-4b

INSTITUT TEKNOLOGI NASIONAL MALANG
FAKULTAS TEKNOLOGI INDUSTRI
PROGRAM STUDI TEKNIK INFORMATIKA S-1
Jl. Karanglo Km. 2 Malang

BERITA ACARA UJIAN SKRIPSI
FAKULTAS TEKNOLOGI INDUSTRI

Nama : Moh. Abdul Hamid
NIM : 0818152
Jurusan : Teknik Informatika S-1
Judul : **APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT**

Dipertahankan dihadapan Majelis Penguji Skripsi Jenjang Strata Satu (S-1) pada :
Hari : Jumat
Tanggal : 10 Agustus 2012
Nilai : 85, 65 (A)

Panitia Ujian Skripsi :

Ketua Majelis Penguji

Joseph Dedy Irawan, ST, MT
NIP.19740416 200501 1 002

Anggota Penguji :

Penguji Pertama

Febriana Santi Wahyuni, S.Kom, M.Kom
NIP.P 1031000425

Penguji Kedua

Nurlaily Vendyansyah, ST

INSTITUT TEKNOLOGI NASIONAL MALANG
FAKULTAS TEKNOLOGI INDUSTRI
PROGRAM STUDI TEKNIK INFORMATIKA S-1
Jl. Karanglo Km. 2 Malang

FORMULIR PERBAIKAN SKRIPSI

Nama : Moh. Abdul Hamid
NIM : 0818152
Jurusan : Teknik Informatika S-1
Judul : **APLIKASI KRIPTOGRAFI MENGGUNAKAN ALGORITMA RSA DAN ALGORITMA AES PADA FILE TEXT**

No	Penguji	Tanggal	Uraian	Paraf
1	Febriana Santi W, S.Kom, M.Kom	10 Agustus 2012	-	
2	Nurlaily Vendyansyah, ST	10 Agustus 2012	Perbaiki Bab V	

Anggota Penguji :

Penguji Pertama

Febriana Santi Wahyuni, S.Kom, M.Kom
NIP.P 1031000425

Penguji Kedua

Nurlaily Vendyansyah, ST

Mengetahui

Dosen Pembimbing I

Ir. Sentot Achmadi, MSi
NIP.Y. 1039500281

Dosen Pembimbing II

Yosep Agus Pranoto, ST
NIP. P. 1031000432

LAMPIRAN-LAMPIRAN

Lampiran 5. Listing Program

```
Unit Kriptografi;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, AES, ExtCtrls, ComCtrls, shellAPI, uBigIntsV3, dffutils,
  Menus, jpeg;

  const
  MsgInformation = 'Informasi';
  MsgError = 'Error.,!!!';

  StrEncStrEmpty = 'Plainteks Masih Kosong.,!!';
  StrDecStrEmpty = 'Cipherteks Masih Kosong';
  StrSrcFileNotExists = 'Input File Terlebih Dahulu.,.,!!';

  StrKeyBitMode = 'Key Bit: %s ,MaxLength: %d';

  StrOpraTime = 'Kecepatan Proses : %d ms';
  StrEncrypting = 'Status : Proses Enkrispi Sedang Berjalan.....';
  StrEncrypted = 'Status : Enkripsi Berhasil';
  StrDecrypting = 'Status : Proses Dekripsi Sedang Berjalan.....';
  StrDecrypted = 'Status : Dekripsi Berhasil';

  szChar = SizeOf(Char);

type

  TActor=(Alice);

  TKeyInt=TInteger;

  TKeyObj=class(TObject)
   ID:string;
   n:TInteger; // hasil antara p*q
   phi:TInteger; // hasil kali dari (p-1)(q-1)
   e:TInteger; //kunci publik
   d:TInteger; //kunci privat dari rumus d*e mod phi = 1)
   blocksize:integer;
   keysize:Integer; // ukuran kunci
   constructor create(newid:string; newkeysize:integer);
  end;

  TForm1 = class(TForm)
   PopupMenu1: TPopupMenu;
```

```
    Panel2: TPanel;
    imAppIcon: TImage;
    Label12: TLabel;
    Label13: TLabel;
    PageControl2: TPageControl;
    TabSheet1: TTabSheet;
    PageControl3: TPageControl;
    TabSheet5: TTabSheet;
    GroupBox8: TGroupBox;
    GroupBox9: TGroupBox;
    Button8: TButton;
    GroupBox7: TGroupBox;
    Label20: TLabel;
    Label21: TLabel;
    Label22: TLabel;
    Edit10: TEdit;
    Edit11: TEdit;
    Edit12: TEdit;
    TabSheet4: TTabSheet;
    PageControl: TPageControl;
    TabSheet3: TTabSheet;
    GroupBox1: TGroupBox;
    memSrcStr: TMemo;
    GroupBox2: TGroupBox;
    memEncStr: TMemo;
    TabSheet8: TTabSheet;
    Label23: TLabel;
    GroupBox4: TGroupBox;
    ledSrcFile: TLabeledEdit;
    btnOpenFile: TButton;
    GroupBox5: TGroupBox;
    btnSaveFile: TButton;
    ledDstFile: TLabeledEdit;
    GroupBox6: TGroupBox;
    lblTime: TLabel;
    lblState: TLabel;
    Panel1: TPanel;
    lblKeyHint: TLabel;
    ledKey: TLabeledEdit;
    cbKeyBit: TComboBox;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    AlicemakeKeyBtn: TButton;
    AliceSizeGrp: TRadioGroup;
    AlicePlainTextMemo: TMemo;
    AliceEncryptedMemo: TMemo;
    Button3: TButton;
```

```
    TabSheet2: TTabSheet;
    GroupBox19: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Button15: TButton;
    PageControl1: TPageControl;
    TabSheet9: TTabSheet;
    GroupBox20: TGroupBox;
    Memo1: TMemo;
    Button16: TButton;
    Button17: TButton;
    GroupBox21: TGroupBox;
    Memo2: TMemo;
    Button22: TButton;
    Button23: TButton;
    Button1: TButton;
    Button2: TButton;
    keygab: TRadioGroup;
    GroupBox14: TGroupBox;
    Label6: TLabel;
    Label7: TLabel;
    GroupBox15: TGroupBox;
    GroupBox26: TGroupBox;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    Button26: TButton;
    Button27: TButton;
    Button30: TButton;
    Button31: TButton;
    btnEnc: TButton;
    btnDec: TButton;
    btnAbout: TButton;
    btnExit: TButton;
    Button28: TButton;
    Button29: TButton;
    procedure AlicemakeKeyBtnClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
```

```
    procedure btnExitClick(Sender: TObject);
    procedure btnAboutClick(Sender: TObject);
    procedure btnEncClick(Sender: TObject);
    procedure btnDecClick(Sender: TObject);
    procedure btnOpenFileClick(Sender: TObject);
    procedure btnSaveFileClick(Sender: TObject);
    procedure Button15Click(Sender: TObject);
    procedure Button16Click(Sender: TObject);
    procedure Button17Click(Sender: TObject);
    procedure Button26Click(Sender: TObject);
    procedure Button27Click(Sender: TObject);
    procedure Button28Click(Sender: TObject);
    procedure Button29Click(Sender: TObject);
    procedure Button30Click(Sender: TObject);
    procedure Button31Click(Sender: TObject);

  public
    Alice:TKeyObj;
    function getkeysize(keysizegrp:TRadiogroup):integer;
    procedure makeRSAKey(Actor:TKeyObj);
    function Encrypt(const s:string; actor:TKeyObj {n,e:Tinteger}):ansistring;
    function Decrypt(const s:string; actor:TKeyObj {n,d:Tinteger}):ansistring;

  end;

var
  Form1: TForm1;
implementation

{$R *.dfm}

uses math;

constructor TKeyObj.create(newid:string; newkeysize:integer);
begin
  id:=newid;
  keysize:=newkeysize;
  n:=TInteger.Create;
  phi:=TInteger.Create;
  e:=TInteger.create;
  d:=Tinteger.create;
end;

{************ Membuat Kunci RSA **************}
procedure TForm1.MakeRSAKey(Actor:TKeyObj);
var
  p,q:TKeyInt;
```

```
  temp:TKeyint;
  primesize:integer;
begin
  screen.cursor:=crhourglass;
  with actor do
  begin
      primesize:=trunc(keysize*log10(2)/2)+1;

   p:=TKeyInt.create;
   q:=TKeyInt.create;
   temp:=TKeyInt.create;

   {* Membangkitkan Nilai Acak p *}
   p.randomOfSize(primesize);
   p.getnextprime;

   {* Membangkitkan Nilai Acak q *}
   repeat
    q.randomOfSize(primesize);
    q.getnextprime;
   until p.compare(q)<>0;

   {n=p*q}
   n.assign(p);
   n.mult(q);
    {Phi=(p-1)*(q-1)}
   Phi.assign(p);
   phi.subtract(1);
   temp.assign(q);
   temp.subtract(1);
   phi.mult(temp);

 {Bilangan Acak e < Phi relatif gcd(phi,e)=1}
   repeat
    e.random(phi);
    temp.assign(phi);
    temp.GCD(e);
   until temp.compare(1)=0;

   d.assign(e);
   d.invmod(phi);
   n.Trim;
   blocksize:=trunc(n.digitcount/log10(256));
   temp.free;
   p.free;
   q.free;
end;
```

```
  screen.cursor:=crdefault;
end;

{************ Tombol Pembangkit Kunci **********}
procedure TForm1.AlicemakeKeyBtnClick(Sender: TObject);
begin
  alice.keysize:=getkeysize(AliceSizeGrp);
  MakeRSAKey(Alice);
  with alice do
  begin
  Edit10.Text:= alice.e.convertTodecimalstring(false);
  Edit11.Text:= alice.d.convertTodecimalstring(false);
  Edit12.Text:= alice.n.ConvertToDecimalString(false);
  end;
end;


{************ Prosedur FormActivate *********}
procedure TForm1.FormActivate(Sender: TObject);
begin
  alice:=tkeyobj.create('Alice', Getkeysize(AliceSizeGrp));
  AlicemakeKeyBtnclick(sender);
end;

{************** Memilih Panjang Kunci *************}
function TForm1.getkeysize(keysizegrp:TRadiogroup):integer;
begin
  case keysizegrp.ItemIndex of // ukuran dalam bentuk bit
    0:result:=16;
    1:result:=32;
    2:RESULT:=64;
  end;
end;

{************** Fungsi Proses Enkripsi **************}
function TForm1.Encrypt(const s:ansistring; actor:TKeyObj):ansistring;
var
  nbrblocks:integer;
  i,j,start:integer;
  p:TKeyInt;
  outblock:integer;
  temps:string;
begin
  with actor do
  begin
    p:=TKeyInt.create;
    nbrblocks:=length(s) div blocksize;
```

```
    outblock:=n.digitcount;
    result:='';
    start:=1;
    for i:=0 to nbrblocks do
    begin
     p.assign(0);
     for j:=start to start+blocksize-1 do
     if j<=length(s) then
     begin
      p.mult(256);
      p.add(ord(s[j]));
     end;
     p.modpow(e,n);
     temps:=p.converttodecimalstring(false);

     while length(temps)<outblock do temps:='0'+temps;
     result:=result + temps;
     inc(start,blocksize);
    end;
  end;
end;

{************ Fungsi Proses Dekripsi **************}
function TForm1.Decrypt(const s:ansistring; actor:TKeyObj{n,d:Tinteger}):ansistring;
var
  k:integer;
  p:TKeyInt;
  q:TInteger;
  estring,dstring:string;
  ch:int64;
  t256:TInteger;
begin

  result:='';
  p:=TKeyInt.create;
  q:=TKeyInt.create;
  t256:=TInteger.create;
  t256.Assign(256);
  estring:=s;
  dstring:='';
  with actor do
  begin
    k:=n.digitcount;
    while length(estring)>0 do
    begin
     p.assign(copy(estring,1,k{-1}));
     p.modpow(d,n);
```

```
      while p.ispositive do
      begin
        p.dividerem(T256,q);
        q.converttoInt64(ch);
        dstring:=char(ch)+dstring;
      end;
      result:=result+dstring;
      dstring:='';
      delete(estring,1,k);
    end;
  end;
end;

function breakstring(s:string; linelength:integer):string;
  var i:integer;
  begin
    result:='';
    for i:=0 to length(s) div linelength do
    result:=result+copy(s,i*linelength+1,linelength)+#13;
    delete(result,length(result),1);
  end;

{********** Tombol Dekripsi **********}
procedure TForm1.Button8Click(Sender: TObject);
var
  Start, Stop: Cardinal;
begin
        Label6.Caption := StrDecrypting;
        Start := GetTickCount;
AliceEncryptedMemo.text:=decrypt(AliccPlainTextMemo.text, alice);
        Stop := GetTickCount;
        Label7.Caption := Format(StrOpraTime, [Stop - Start]);
        Label6.Caption := StrDecrypted;
end;

{********** TOmbol Enkripsi **********}
procedure TForm1.Button3Click(Sender: TObject);
var
  Start, Stop: Cardinal;
begin
        Label6.Caption := StrEncrypting;
        Start := GetTickCount;
aliceencryptedmemo.text:= encrypt(AlicePlaintextMemo.text,alice);
        Stop := GetTickCount;
        Label7.Caption := Format(StrOpraTime, [Stop - Start]);
        Label6.Caption := StrEncrypted;
end;
```

```
////////////////////////// EXIT ////////////////////////////////////
procedure TForm1.btnExitClick(Sender: TObject);
begin
Application.Terminate;
end;
////////////////////////////////////////////////////////////////////
                        {***AES***}
////////////////////////////////////////////////////////////////////
{************** Memilih Panjang Kunci AES ************}
function ukurankunci(cbKeyBit: TComboBox):integer;
begin
  case cbKeyBit.ItemIndex of
    0: result := 128;
    1: result := 192;
    2: result := 256;
  end;

end;

procedure TForm1.btnAboutClick(Sender: TObject);
var
  S: string;
begin
S := 'Aplikasi Kriptografi' + #13 + #13 +
    '*Algoritma RSA merupakan algoritma asimetris yang menggunakan'
    + #13 + 'kunci publik unruk enkripsi dan kunci privat untuk dekripsi.' + #13 +

     '*Algoritma AES merupakan algoritma simetris yang menggunakan satu'+ #13 +
    'kunci privat untuk melukan enkripsi dan dekripsi.';
  MessageBox(Handle, PChar(S), MsgInformation, MB_ICONINFORMATION);
end;

procedure TForm1.btnEncClick(Sender: TObject);
var
  Start, Stop: Cardinal;
begin
case PageControl.ActivePageIndex of
   0: // Page Control enkripsi TEXT
     begin
       if Length(memSrcStr.Text) > 0 then
       begin  //Proses Enkripsi TEXT
          Label10.Caption := StrEncrypting;
          Start := GetTickCount;
        case cbKeyBit.ItemIndex of
          0:  memEncStr.Text  :=  EncryptString(memSrcStr.Text,  ledKey.Text);  //
Menggunakan KeyBit:128
          1: memEncStr.Text := EncryptString(memSrcStr.Text, ledKey.Text, kb192);
```

```
        2: memEncStr.Text := EncryptString(memSrcStr.Text, ledKey.Text, kb256);
       end;
       Stop := GetTickCount;
        Label11.Caption := Format(StrOpraTime, [Stop - Start]);
        Label10.Caption := StrEncrypted;
      end else
       MessageBox(Handle,          StrEncStrEmpty,          MsgInformation,
MB_ICONINFORMATION);
    end;

  1: //Page Control Enkripsi File
    begin
     if FileExists(ledSrcFile.Text) then
     begin
      lblState.Caption := StrEncrypting;
      Start := GetTickCount;
      case cbKeyBit.ItemIndex of
       0: EncryptFile(ledSrcFile.Text, ledDstFile.Text, ledKey.Text);  // Menggunakan
KeyBit:128
        1: EncryptFile(ledSrcFile.Text, ledDstFile.Text, ledKey.Text, kb192);
        2: EncryptFile(ledSrcFile.Text, ledDstFile.Text, ledKey.Text, kb256);
       end;
       Stop := GetTickCount;
       lblTime.Caption := Format(StrOpraTime, [Stop - Start]);
       lblState.Caption := StrEncrypted;
      end else
       MessageBox(Handle, StrSrcFileNotExists, MsgError, MB_ICONERROR);
    end;
 end;
end;

procedure TForm1.btnDecClick(Sender: TObject);
var
 Start, Stop: Cardinal;
begin
case PageControl.ActivePageIndex of
  0:
   begin
    if Length(memEncStr.Text) > 0 then
    begin   // Proses Dekripsi TEXT
    Label10.Caption := StrDecrypting;
     Start := GetTickCount;
     case cbKeyBit.ItemIndex of
      0:  memEncStr.Text  :=  DecryptString(memSrcStr.Text,  ledKey.Text);  //
Menggunakan KeyBit:128
       1: memEncStr.Text := DecryptString(memSrcStr.Text, ledKey.Text, kb192);
       2: memEncStr.Text := DecryptString(memSrcStr.Text, ledKey.Text, kb256);
```

```
        end;
        Stop := GetTickCount;
        Label11.Caption := Format(StrOpraTime, [Stop - Start]);
        Label10.Caption := StrDecrypted;
       end else
        MessageBox(Handle,          StrDecStrEmpty,          MsgInformation,
MB_ICONINFORMATION);
     end;

   1: // Proses Dekripsi File
     begin
      if FileExists(ledSrcFile.Text) then
      begin
       lblState.Caption := StrDecrypting;
       Start := GetTickCount;
       case cbKeyBit.ItemIndex of
        0: DecryptFile(ledSrcFile.Text, ledDstFile.Text, ledKey.Text);  //Menggunakan
KeyBit:128
        1: DecryptFile(ledSrcFile.Text, ledDstFile.Text, ledKey.Text, kb192);
        2: DecryptFile(ledSrcFile.Text, ledDstFile.Text, ledKey.Text, kb256);
       end;
       Stop := GetTickCount;
       lblTime.Caption := Format(StrOpraTime, [Stop - Start]);
       lblState.Caption := StrDecrypted;
      end else
       MessageBox(Handle, StrSrcFileNotExists, MsgError, MB_ICONERROR);
     end;
  end;
end;

procedure TForm1.btnOpenFileClick(Sender: TObject);
begin
with OpenDialog do
 begin
  if Execute then
   ledSrcFile.Text := FileName;
 end;
end;

procedure TForm1.btnSaveFileClick(Sender: TObject);
begin
with SaveDialog do
 begin
  if Execute then
   ledDstFile.Text := FileName;
 end;
end;
```

```
procedure TForm1.Button15Click(Sender: TObject);
begin
 //alice.keysize:=getkeysize(keygab);
 alice.keysize:=getkeysize(keygab);
  MakeRSAKey(Alice);
  with alice do
  begin
  Edit1.Text:= alice.e.convertTodecimalstring(false);
  Edit2.Text:= alice.d.convertTodecimalstring(false);
  Edit3.Text:= alice.n.ConvertToDecimalString(false);
  end;
end;

////////////// Enkripsi TEXT RSA & AES ////////////////////////
procedure TForm1.Button16Click(Sender: TObject);
var
A: String;
Start, Stop: Cardinal;
begin
        Label8.Caption := StrEncrypting;
        Start := GetTickCount;
A:= encrypt(Memo1.text,alice);
Memo2.Text:= EncryptString(A, Edit2.Text);
        Stop := GetTickCount;
        Label9.Caption := Format(StrOpraTime, [Stop - Start]);
        Label8.Caption := StrEncrypted;
end;

//////////////////////// Dekripsi TEXT RSA & AES ////////////////////////
procedure TForm1.Button17Click(Sender: TObject);
var
B: string;
Start, Stop: Cardinal;
begin
       Label8.Caption := StrDecrypting;
       Start := GetTickCount;
B:= DecryptString(Memo1.Text, Edit2.Text);
Memo2.Text:= decrypt(B, alice);
       Stop := GetTickCount;
       Label9.Caption := Format(StrOpraTime, [Stop - Start]);
       Label8.Caption := StrDecrypted;
end;
//////////////////////// Open & Save Text ////////////////////////////////
procedure TForm1.Button26Click(Sender: TObject);
begin
 OpenDialog1.Execute;
  Form1.Caption := OpenDialog1.FileName;
```

```
  AlicePlainTextMemo.Lines.LoadFromFile(OpenDialog1.FileName);
  AlicePlainTextMemo.SelStart := 0;
end;

procedure TForm1.Button27Click(Sender: TObject);
begin
SaveDialog1.FileName := Form1.Caption;
SaveDialog1.Execute;
  AliceEncryptedMemo.Lines.SaveToFile(SaveDialog1.FileName + '.txt');
  Form1.Caption:=SaveDialog1.FileName;
end;

procedure TForm1.Button28Click(Sender: TObject);
begin
OpenDialog1.Execute;
 Form1.Caption := OpenDialog1.FileName;
 memSrcStr.Lines.LoadFromFile(OpenDialog1.FileName);
 memSrcStr.SelStart := 0;
end;
procedure TForm1.Button29Click(Sender: TObject);
begin
SaveDialog1.FileName := Form1.Caption;
SaveDialog1.Execute;
  memEncStr.Lines.SaveToFile(SaveDialog1.FileName + '.txt');
  Form1.Caption:=SaveDialog1.FileName;
end;

procedure TForm1.Button30Click(Sender: TObject);
begin
OpenDialog1.Execute;
 Form1.Caption := OpenDialog1.FileName;
 Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
 Memo1.SelStart := 0;
end;

procedure TForm1.Button31Click(Sender: TObject);
begin
SaveDialog1.FileName := Form1.Caption;
SaveDialog1.Execute;
 Memo2.Lines.SaveToFile(SaveDialog1.FileName + '.txt');
  Form1.Caption:=SaveDialog1.FileName;
end;

end.
```