**Implementasi Algoritma AES (Advance Encryption Standard) Rijndael Pada Aplikasi Keamanan Data**

**Agung Prajuhana Putra1, Herfina2, Sufiatul Maryana3, Andrian Setiawan4**

1,2,3,4 Prodi Teknik Informatika

Universitas Pakuan Bogor

**1prajuhana.putra@unpak.ac.id, 2herfina@unpak.ac.id, 3****sufiatul.maryana@unpak.ac.id****. 4setiawanandrian1@gmail.com**

**Abstract**- AES Rijndael algorithm is a modern cryptographic algorithm published by NIST (National Institute of Standards and Technology) in 2001 using block cipher mode and using symmetric keys. AES Rijndael as a substitute for DES (Data Encryption Standard) algorithm whose use began in 1977 and has ended. The length of the block cipher used is 128 bits and variations in key length are 128 bits, 192 bits and 256 bits. AES Rijndael's algorithm has a reliability that is the simplicity of the bytes transformation process so that it can streamline encryption and decryption, and has high security.

Implementation of the AES Rijndael algorithm will be done on Android devices to secure digital files, which can be used for all types of file types. The parameters used in the analysis are testing of the encryption and decryption processing time, changes in size and bits of the encrypted and decrypted files, then proving the strength of the key length against the robustness of the AES Rijndael algorithm as a symmetric algorithm.

Keywords : Cryptography, AES Rijndael, File, Android.

1. **Pendahuluan**

Algoritma AES Rijndael adalah algoritma kriptografi modern yang di publikasi oleh NIST (National Institute of Standard and Technology) tahun 2001 menggunakan mode cipher blok dan menggunakan kunci simetris. AES Rijndael sebagai pengganti dari Algoritma DES (Data Encryption Standard) yang penggunaannya dimulai tahun 1977 dan telah berakhir. Panjang cipher blok yang digunakan sebesar 128 bit dan variasi panjang kunci 128 bit, 192 bit dan 256 bit. Kehandalan-kehandalan Algoritma AES Rijndael sudah dibuktikan oleh: Michell (2011) memberikan kesimpulan algoritma AES Rijndael memiliki tingkat keamanan yang tinggi berdasarkan variasi panjang kunci yang dimiliki, serta memiliki kompleksitas waktu dan ruang yang baik karena kesederhanaannya dengan perbandingan algoritma DES. Bagus, dkk (2014) algoritma AES Rijndael lebih efisien dibandingkan dengan algoritma Camellia, menggunakan teori kompleksitas waktu sebagai parameter efisiensi. Algoritma Camellia lebih efektif dibandingkan algoritma AES Rijndael, menggunakan nilai Avalanche Effect sebagai parameter efektifitas. Algoritma AES Rijndael lebih berkualitas dibandingkan dengan algoritma Camellia,menggunakan perhitungan deviasi maksimum, koefisien korelasi, deviasi ketidakteraturan dan PSNR sebagai parameter kualitas. Penelitian dilakukan pada file berbentuk citra. Bayu, dkk (2015) algoritma AES Rijndael lebih efisien diimplementasikan di perangkat android karena memiliki kompleksitas waktu yang unggul dibandingkan dengan algoritma serpent. Penelitian dilakukan pada teks untuk pesan singkat/Short Message Service. Oleh karena itu, dalam penelitian ini akan mengimplementasikan algoritma AES Rijndael sebagai algoritma kriptografi pada aplikasi android pengamanan file digital untuk seluruh jenis file extension, kemudian akan dilakukan analisa pada proses dan hasil dari algoritma AES Rijndael, dalam penerapan di perangkat android dengan beberapa parameter yang digunakan diantaranya pengujian terhadap waktu proses enkripsi dan dekripsi, pengujian terhadap perubahan size dan perubahan bit pada file hasil enkripsi dan pengujian tentang pembuktian kekuatan algoritma simetris terletak pada panjang kunci yang dimiliki.

1. **Metode**

Metode penelitian yang digunakan adalah pola siklus hidup pengembangan sistem/System Development Life Cycle (SDLC). Pertimbangan menggunakan model ini karena mempunyai tahapan yang cukup lengkap dan terstruktur. Informasi ini menyangkut langkah-langkah yang diperlukan untuk merancang sistem tersebut. Berikut studi yang digunakan dalam pengumpulan informasi.



Gambar 1. Tahapan System Development Life Cycle (SDLC)

1. **Hasil Dan Pembahasan**

**Tahap Perencanaan Sistem**

Tahap perencanan sistem merupakan tahapan awal yang dilakukan dalam merancang sebuah sistem. Pada tahap ini dilakukan identifikasi kebutuhan dan menentukan tujuan pembuatan sistem. Tujuan dilakukannya pembuatan sistem ini adalah untuk membuat aplikasi enkripsi dan dekripsi file berbasis android.

**Analisis Sistem**

1. **Analisis Algoritma**
2. **Pembangkitan (Ekspansi) Kunci**

Pembangkitan atau ekspansi kunci dilakukan dengan tujuan mendapatkan kunci ronde atau round key yang akan digunakan untuk proses enkripsi dan dekripsi, tepatnya pada tahap transformasi AddRoundKey. Tanpa proses pembangkitan kunci maka proses enkripsi dan dekripsi tidak akan berjalan sebagaimana mestinya. Langkah pertama yaitu menyediakan blok matrik (array) 128 bit untuk menampung kunci cipher yang akan dibangkitkan.

Untuk kunci ronde ke-0 yaitu menggunakan kunci yang kita inputkan, untuk kunci ronde ke-1 s/d kunci ronde ke-10 pada AES-128 kita bangkitkan dari kunci awal, yaitu dengan cara melakukan transformasi byte Rotation Word, Subtitution Bytes dan XOR (Exclusive OR) dengan table Round Constant (RCon).

**Simulasi Ekspansi Kunci AES-128**

Plain text : ILMUKOMPUTER2012

Hexadecimal : 49 4C 4D 55 4B 4F 4D 50 55 54 45 52 32 30 31 32

Key : FMIPAILKOMUNPAK1

Hexadecimal : 46 4D 49 50 41 49 4C 4B 4F 4D 55 4E 50 41 4B 31

Tabel 1. Matrik Plain Text / State



Tabel 2. Matrik Cipher Key



Tabel 3. Ekspansi Kunci



Selanjutnya mencari kunci ronde ke-1 akan digunakan transformasi Rotation Word atau menggeser blok paling atas ke blok paling bawah pada kolom terakhir kunci ronde ke-0.



Gambar 2. Proses Rotation Word

Kemudian hasil Rotation Word di Subtitution Bytes menggunakan table S-Box.





Gambar 3. Proses Subtitution Bytes

Langkah selanjutnya, hasil Subtitution Bytes di-XOR dengan kolom 1 kunci ronde ke-0 dan kolom 1 tabel RCon.

Tabel 4. Round Constanta / RCon



Gambar 4. Proses XOR

Selanjutnya mencari kolom ke-2 sampai kolom ke-4 kunci ronde ke-1 menggunakan proses XOR, yaitu kolom ke-1 kunci ronde ke-1 di-XOR dengan kolom ke-2 kunci ronde ke-0, kemudian kolom ke-2 kunci ronde ke-1 di-XOR dengan kolom ke-3 kunci ronde ke-0, kemudian kolom ke-3 kunci ronde ke-1 di-XOR dengan kolom ke-4 kunci ronde ke-0.



Gambar 5. Proses XOR Untuk Kolom Ke-2 Kunci Ronde Ke-1

**Transformasi Proses Enkripsi**

Terdapat 4 transformasi yang akan digunakan pada proses enkripsi, yaitu:

1. Transformasi AddRoundKey
2. Transformasi SubBytes
3. Transformasi ShiftRows
4. Transformasi MixColumns

**Transformasi AddRoundKey**

Transformasi ini merupakan pencampuran anatara kunci ronde dengan state dengan operasi XOR. Transformasi ini terjadi pada ronde ke-0 dan terjadi pada ronde ke-1 sampai ronde ke-9 setelah transformasi MixColumns untuk AES-128, ronde ke-1 sampai ronde ke-11 setelah transformasi MixColumns untuk AES-192, dan ronde ke-1 sampai ronde ke-13 setelah transormasi MixColumns untuk AES-256. Serta ronde ke-10 setelah transformasi ShiftRows untuk AES-128, ronde ke-12 setelah transformasi ShiftRows untuk AES-192, dan ronde ke-14 setelah transformasi ShiftRows untuk AES-256 karena pada ronde terakhir/final round tanpa transformasi MixColumns.

**Simulasi Transformasi AddRoundKey AES-128 (Ronde 0)**

****

****

Gambar 6. Proses AddRoundKey Ronde ke-0

**Transformasi SubBytes**

Transformasi SubBytes merupakan transformasi yang dilakukan setelah AddRoundKey. Pada Transformasi ini melakukan operasi subtitusi hasil dari AddRoundKey (Ronde ke-0) menggunakan table S-Box.

**Simulasi Transformasi SubBytes AES-128 (Ronde 1)**

****

Gambar 7. Proses SubBytes Ronde ke-1

**Transformasi ShiftRows**

Transformasi ShiftRows merupakan pergeseran baris 2, 3, dan 4 pada state hasil transformasi SubBytes.

**Simulasi Transformasi ShiftRows (Ronde 1)**

****

Gambar 8. Proses ShiftRows Ronde ke-1

**Transformasi MixColumns**

Transformasi MixColumns disebut juga transformasi perkalian matrik, proses perkalian state hasil ShiftRows dengan matrik multiplication. Setiap kolom pada state dikalikan dengan matrik multiplication tersebut, bilangan didalam matrik multiplication tersebut merupakan bilangan hexadecimal. Jika nilai yang dihasilkan lebih besar dari hexadecimal 0xFF (256 desimal) maka harus di XOR dengan hexadecimal 0x11B.



Gambar 9. Matrik Multiplication MixColumns

**Simulasi Transformasi MixColumns (Ronde 1)**





Gambar 10. Proses MixColumns Ronde ke-1

S’0,1 = (02\*76) XOR (03\*6F) XOR (01\*CA) XOR (01\*7B),

S’1,1 = (01\*76) XOR (02\*6F) XOR (03\*CA) XOR (01\*7B),

S’2,1 = (01\*76) XOR (01\*6F) XOR (02\*CA) XOR (03\*7B),

S’3,1 = (03\*76) XOR (01\*6F) XOR (01\*CA) XOR (02\*7B), dan seterusnya.

**Simulasi Transformasi AddRoundKey (Ronde 1)**





Gambar 11. Proses AddRoundKey (Ronde 1)

Proses transformasi bytes pada ronde ke-1 dilakukan secara berulang sampai ronde ke-9. Setelah selesai ronde ke-9, melakukan transformasi pada ronde ke-10/final round. Pada final round transformasi yang dilakukan hanya SubBytes, ShiftRows dan AddRoundKey. Setelah selesai melakukan tahap final round, didapatkanlah hasil proses enkripsi yang di sebut ciphertext.

**Transformasi SubBytes**

Transformasi SubBytes merupakan transformasi yang dilakukan setelah AddRoundKey. Pada Transformasi ini melakukan operasi subtitusi hasil dari AddRoundKey (Ronde ke-0) menggunakan table S-Box.

**Transformasi Proses Dekripsi**

Pada Proses dekripsi AES terjadi 4 transformasi, dimana 3 transformasi merupakan inverse atau kebalikan dari transformasi pada proses enkripsi AES.

1. Transformasi AddRoundKey
2. Transformasi InverseMixColumns
3. Transformasi InverseShiftRows
4. Transformasi InverseSubBytes

**Transformasi AddRoundKey**

Transformasi AddRoundKey merupakan pencampuran antara kunci ronde ke-10/final round pada AES-128 dengan state (blokarray berisi 16 byte ciphertext). Proses dekripsi AES beroperasi berlawanan arah dengan proses enkripsi, sehingga dimulai pada tahap final round.

**Transformasi InverseShiftRows**

Setelah AddRoundKey dilanjutkan dengan transformasi InverseShiftRows, dimana terjadi pergeseran ke sebalah kanan pada baris 2, 3, dan 4. Pergeseran tersebut merupakan kebalikan dari transformasi ShiftRows.





Gambar 12. Transformasi InvShiftRows

**Transformasi InverseSubBytes**

InvSubBytes juga merupakan transformasi bytes yang berkebalikan dengan transformasi SubBytes. Pada InvSubBytes, tiap elemen pada state dipetakan dengan menggunakan tabel Inverse S-Box.

Tabel 5. Tabel Inverse S-Box



**Transformasi InverseMixColumns**

Transformasi InverserMixColumns melakukan operasi perkalian matrik antara state dengan matrik multiplication. Jika terdapat hasil perkalian yang melebihi nilai 0xFF maka harus di XOR dengan 0x11B.



Gambar 13. Matrik Multiplication InverseMixColumns

**Analisis Masalah**

Langkah-langkah yang ditempuh yang berguna untuk mengetahui berbagai masalah yang ada, sehingga dengan adanya aplikasi enkripsi dan dekripsi file berbasis android ini diharapkan bisa membantu permasalahan yang ada. Adapun beberapa masalah tersebut adalah sebagai berikut:

1. Penyalahgunaan data
2. Pemalsuan data

**Analisis Sistem**

Tahapan analisis ini berisi identifikasi dan evaluasi permasalahan yang ada serta kebutuhan apa saja yang dibutuhkan dalam pembuatan proyek perangkat lunak, dengan adanya analisa dan perancangan maka akan di didapatkan suatu gambaran mengenai kebutuhan apa saja yang dibutuhkan dalam program aplikasi ini. Langkah-langkah dalam penggunaan aplikasi dari awal hingga akhir adalah sebagai berikut:

1. Buka aplikasi enkripsi dan dekripsi file ini
2. Ketika dibuka halaman awal yang tampil yaitu splashscreen dari aplikasi ini, berisi nama algoritma yang di gunakan pada aplikasi ini, background dari tampilan splashscreen dan progress bar.
3. Setelah splashscreen tampil lalu akan tampil halaman menu utama, dihalaman ini terdapat empat widget button image, yaitu encryption, decryption, file manager dan about.
4. Lakukan proses enkripsi/dekripsi dengan cara memilih menu yang terdapat pada menu utama, yaitu menu encryption/decryption.
5. Setelah menentukan proses yang diinginkan, pada menu encryption/decryption user dapat melampirkan file yang akan di enkripsi/dekripsi, menginputkan kunci sebesar 128/192/256 bit atau 16/24/32 karakter, lakukan proses enkripsi/dekripsi file dengan menekan button image bergambar file terkunci, indikator proses telah selesai yaitu waktu akan tampil, setalah itu file yang telah berhasil di enkripsi/dekripsi di simpan.
6. Setelah melakukan proses enkripsi/dekripsi file, user dapat melihat file dengan membuka menu file manager, file tersimpan pada main storage dengan nama folder crypto.
7. **Tahap Proses Perancangan**

**Struktur Navigasi**

Struktur navigasi berfungsi untuk mengarahkan pengguna dalam menjalankan aplikasi prioritas standar pengelolaan. Berikut struktur navigasi aplikasi prioritas pengelolaan dapat dilihat pada Gambar 14.



Gambar 14. Struktur Navigasi

**Flowchart**

Perancangan system memerlukan suatu diagram alur atau flowchart program yang menggambarkan proses kerja program dari aplikasi enkripsi dan dekripsi file berbasis android. Berikut dapat dilihat pada Gambar 15.



Gambar 15. Flowchart Sistem

**Perancangan Interface**

Pada tahap perancangan interface,

dirancang agar pengguna mudah menggunakan dan design yang di tampilkan menarik dan enak di lihat.



Gambar 16. Perancangan Interface

1. **Tahap Uji Coba**

**Uji Coba Struktural**

Uji Coba Struktural dilakukan untuk mengetahui apakah struktur atau alur sistem yang dibuat sudah sesuai dengan rancangan. Hasil uji coba sistem secara structural dapat dilihat pada Table 6.

Table 6. Uji Coba Struktural

****

**Uji Coba Funsional**

Uji coba fungsional dilakukan untuk mengetahui apakah sistem yang dibuat sudah berfungsi dengan baik. Pada tahap ini dilakukan dengan baik. Hasil uji coba sistem secara fungsional dapat dilihat pada Tabel 7.

Tabel 7. Uji Coba Fungsional

|  |  |  |  |
| --- | --- | --- | --- |
| **Halaman** | **Widget** | **Fungsi** | **Keterangan** |
| Splashscreen | Text View | Menampilkan Nama Algoritma | Berfungsi |
| Splashscreen | Image View | Menampilkan Background | Berfungsi |
| Splashscreen | Progressbar | Indicator Proses Splashscreen | Berfungsi |
| Menu Utama | Button Image Encryption | Menampilkan Halaman Enkripsi | Berfungsi |
| Menu Utama | Button Image Decryption | Menampilkan Halaman Dekripsi | Berfungsi |
| Menu Utama | Button Image File Manager | Menampilkan Aplikasi File Manager + | Berfungsi |
| Menu Utama | Button Image About | Menampilkan Halaman Petunjuk Penggunaan | Berfungsi |
| Encryption | Button Image Lampirkan File | Untuk Memilih File Yang Akan di Enkripsi | Berfungsi |
| Encryption | Text View Nama File | Menampilkan Nama File Yang Dipilih | Berfungsi |
| Encryption | Edit Text Input Kunci | Menginput Kunci Untuk Proses Enkripsi | Berfungsi |
| Encryption | Text View Waktu | Menampilkan Waktu Proses Enkripsi | Berfungsi |
| Encryption | Button Image Enkripsi | Melakukan Aksi Enkripsi File | Berfungsi |
| Encryption | Button Image Save | Menyimpan File Yang Telah Dienkripsi | Berfungsi |
| Decryption | Button Image Lampirkan File | Untuk Memilih File Yang Akan di Dekripsi | Berfungsi |
| Decryption | Text View Nama File | Menampilkan Nama File Yang Dipilih | Berfungsi |
| Decryption | Edit Text Input Kunci | Menginput Kunci Untuk Proses Dekripsi | Berfungsi |
| Decryption | Text View Waktu | Menampilkan Waktu Proses Dekripsi | Berfungsi |
| Decryption | Button Image Dekripsi | Melakukan Aksi Dekripsi File | Berfungsi |
| Decryption | Button Image Save | Menyimpan File Yang Telah Didekripsi | Berfungsi |
| About | Image View | Menampilkan Petunjuk Penggunaan Aplikasi | Berfungsi |

**Uji Kompleksitas Waktu**

**Uji Coba Validasi**

Pada pengujian kompleksitas waktu, metode yang digunakan yaitu notasi Big O. Kompleksitas waktu T(n) diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan (n). Jumlah tahapan komputasi dihitung dari berapa kali suatu operasi dilaksanakan di dalam sebuah algoritma sebagai fungsi ukuran masukan (n).

Dalam menghitung time complexity dan space complexity (yang biasanya dinotasikan dengan Big-O notation), ada beberapa aturan yang perlu dilakukan disini:

1. Abaikan konstanta, misalkan O(N + 2), maka dianggap O(N) saja.
2. Abaikan non dominant terms,misalkan O(N² + N), maka dianggap O(N²) saja.

**Kompleksitas Waktu Enkripsi**

Tabel 7. Perhitungan Jumlah Eksekusi Source Code Enkripsi

|  |  |
| --- | --- |
| Source Code | Jumlah Eksekusi |
| public static byte[] encrypt(byte[] in,byte[] key){ |  |
| *Nb* = 4; | 1 |
| *Nk* = key.length/4; | 1 |
| *Nr* = *Nk* + 6; | 1 |
|  int lenght=0; | 1 |
|  byte[] padding = new byte[1]; | 1 |
|  int i; | 1 |
|  lenght = 16 - in.length % 16; | 1 |
|  padding = new byte[lenght]; | 1 |
|  padding[0] = (byte) 0x80; | 1 |
|  for (i = 1; i < lenght; i++) | n  |
|  padding[i] = 0; |  |
|  byte[] tmp = new byte[in.length + lenght]; | 1 |
|  byte[] bloc = new byte[16]; | 1 |
| *w* = *generateSubkeys*(key); | 1 |
|  int count = 0; | 1 |
|  for (i = 0; i < in.length + lenght ; i++) { | n |
|  if (i > 0 && i % 16 == 0) { |  |
|  bloc = *encryptBloc*(bloc); |  |
|  System.*arraycopy*(bloc, 0, tmp, i - 16, bloc.length); |  |
|  } |  |
|  if (i < in.length) | 1 |
|  bloc[i % 16] = in[i]; |  |
|  else{ | 1 |
|  bloc[i % 16] = padding[count % 16]; |  |
|  count++; |  |
|  } |  |
|  } |  |
|  if(bloc.length == 16){ | 1 |
|  bloc = *encryptBloc*(bloc); |  |
|  System.*arraycopy*(bloc, 0, tmp, i - 16, bloc.length); |  |
|  } |  |
|  return tmp; | 1 |
| } |  |
| **T(n) = 17 + 2n = O(n)** |
| Source Code | Jumlah Ekesekusi |
| public static byte[] encryptBloc(byte[] in) { |  |
|  byte[] tmp = new byte[in.length]; | 1 |
|  byte[][] state = new byte[4][*Nb*]; | 1 |
|  for (int i = 0; i < in.length; i++) | n |
|  state[i / 4][i % 4] = in[i%4\*4+i/4]; |  |
|  state = *AddRoundKey*(state, *w*, 0); | 1 |
|  for (int round = 1; round <*Nr*; round++) { | n |
|  state = *SubBytes*(state); |  |
|  state = *ShiftRows*(state); |  |
|  state = *MixColumns*(state); |  |
|  state = *AddRoundKey*(state, *w*, round); |  |
|  } |  |
|  state = *SubBytes*(state); | 1 |
|  state = *ShiftRows*(state); | 1 |
|  state = *AddRoundKey*(state, *w*, *Nr*); | 1 |
|  for (int i = 0; i < tmp.length; i++) | n |
|  tmp[i%4\*4+i/4] = state[i / 4][i%4]; |  |
|  return tmp; | 1 |
| } |  |
| **T(n) = 7 + 3n = O(n)** |

Perhitungan kompleksitas waktu enkripsi :**T(n) = O(n) + O(n) = O (2n) = On**

**Kompleksitas Waktu Dekripsi**

Tabel 8. Perhitungan Jumlah Eksekusi Source Code Dekripsi

|  |  |
| --- | --- |
| Source Code | Jumlah Eksekusi |
| public static byte[] decrypt(byte[] in,byte[] key){ |  |
|  int i; | 1 |
|  byte[] tmp = new byte[in.length]; | 1 |
|  byte[] bloc = new byte[16]; | 1 |
| *Nb* = 4; | 1 |
| *Nk* = key.length/4; | 1 |
| *Nr* = *Nk* + 6; | 1 |
| *w* = *generateSubkeys*(key); | 1 |
|  for (i = 0; i < in.length; i++) { | N |
|  if (i > 0 && i % 16 == 0) { |  |
|  bloc = *decryptBloc*(bloc); |  |
|  System.*arraycopy*(bloc, 0, tmp, i - 16, bloc.length); |  |
|  } |  |
|  if (i < in.length) |  |
|  bloc[i % 16] = in[i]; |  |
|  } |  |
|  bloc = *decryptBloc*(bloc); | 1 |
|  System.*arraycopy*(bloc, 0, tmp, i - 16, bloc.length); | 1 |
|  tmp = *deletePadding*(tmp); | 1 |
|  return tmp; | 1 |
| } |  |
| **T(n) = 11 + n = O(n)** |
| Source Code | Jumlah Ekesekusi |
| public static byte[] decryptBloc(byte[] in) { |  |
|  byte[] tmp = new byte[in.length]; | 1 |
|  byte[][] state = new byte[4][*Nb*]; | 1 |
|  for (int i = 0; i < in.length; i++) | n |
|  state[i / 4][i % 4] = in[i%4\*4+i/4]; |  |
|  state = *AddRoundKey*(state, *w*, *Nr*); | 1 |
|  state = *InvShiftRows*(state); | 1 |
|  state = *InvSubBytes*(state); | 1 |
|  for (int round = *Nr*-1; round >=1; round--) { | n |
|  state = *AddRoundKey*(state, *w*, round); |  |
|  state = *InvMixColumns*(state); |  |
|  state = *InvShiftRows*(state); |  |
|  state = *InvSubBytes*(state); |  |
|  } |  |
|  state = *AddRoundKey*(state, *w*, 0); | 1 |
|  for (int i = 0; i < tmp.length; i++) | n |
|  tmp[i%4\*4+i/4] = state[i / 4][i%4]; |  |
|  return tmp; | 1 |
| } |  |
| **T(n) = 7 + 3n = O(n)** |

Perhitungan kompleksitas waktu dekripsi :**T(n) = O(n) + O(n) = O(2n) = O(n)**

**Uji Coba Perubahan Size dan Perubahan bit**

**Perubahan Size**

Berdasarkan hasil pengujian dari proses enkripsi dan dekripsi file, file yang telah berhasil di enkripsi akan mengalami penambahan size file beberapa KB, hal ini di karenakan adanya proses padding, yaitu pengelompokan bit-bit sepanjang 128 bit, sehingga ketika ada kelompok bit yang kurang dari 128 akan mengalami penambahan bit dummy, sehingga akan mempengaruhi pada ukuran file.

Tabel 9. Ukuran File Hasil Enkripsi dan Dekripsi

|  |  |  |  |
| --- | --- | --- | --- |
| **Jenis File** | **Size File Asli (KB)** | **Size File Enkripsi (KB)** | **Size File Dekripsi (KB)** |
| Aplikasi (.exe) | 2.918.464 | 2.918.480 | 2.918.464 |
| 5.489.552 | 5.489.568 | 5.489.552 |
| Audio (.mp3) | 2.680.708 | 2.680.720 | 2.680.708 |
| 5.736.825 | 5.736.832 | 5.736.825 |
| Compress (.rar) | 2.625.102 | 2.625.104 | 2.625.102 |
| 5.290.754 | 5.290.768 | 5.290.754 |
| Documents (.docx) | 2.758.074 | 2.758.080 | 2.758.074 |
| 5.583.829 | 5.583.840 | 5.583.829 |
| Image (.jpg) | 2.651.905 | 2.651.920 | 2.651.905 |
| 5.286.662 | 5.286.672 | 5.286.662 |
| Video (.mp4) | 2.661.097 | 2.661.104 | 2.661.097 |
| 5.646.551 | 5.646.560 | 5.646.551 |

**Perubahan Bit**

Pada pengujian perubahan bit ini, menggunakan metode avalanche effect. Suatu avalanche effect dikatakan baik jika perubahan bit yang dihasilkan berkisar antara 45-60%. Hal ini dikarenakan perubahan tersebut berarti membuat perbedaan yang cukup sulit untuk kriptanalis melakukan serangan. Nilai avalanche effect dirumuskan dengan :

$$Avalanche Effect \left(AE\right)=\frac{Jumlah bit berubah}{Jumlah bit total}X100\%$$

Gambar 17. Perhitungan Avalanche Effect

Tabel 10. Perubahan Bit Plainttext dan Ciphertext

|  |
| --- |
| Aplikasi (.exe) |
| Jenis Kunci | Plaintext | Ciphertext | Avalanche Effect |
| 128 | 01001101010110101001000000000000000000110000000000000000000000000000010000000000000000000000000011111111111111110000000000000000 | 00110010001011100110101110100011101000000011101011100001010110110111001010101001011010111111110011110110001111001010111111110001 | $$\frac{76}{128} x 100\%=59\%$$ |
| 192 | 01001101010110101001000000000000000000110000000000000000000000000000010000000000000000000000000011111111111111110000000000000000 | 11100111110000010100001101110111111011101001011110011101111110011100001111111010101101001010111110110010000001100100011010010000 | $$\frac{78}{128} x 100\%=60\%$$ |
| 256 | 01001101010110101001000000000000000000110000000000000000000000000000010000000000000000000000000011111111111111110000000000000000 | 01101010100100011100001011000011111111110000000101000101101001001010110111100100101001000101001001011010100100111111110000100001 | $$\frac{59}{128} x 100\%=46\%$$ |

|  |
| --- |
| Audio (.mp3) |
| Jenis Kunci | Plaintext | Ciphertext | Avalanche Effect |
| 128 | 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 00011010001001001001001110100101011101110111000000011011110110001000111010001111010011011101101010101000001000111100000010100000 | $$\frac{58}{128} x 100\%=45\%$$ |
| 192 | 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 00011010110100111111111001101001111101111001111101100101101110100001001110111011100010001101001111101101000000000010010011100001 | $$\frac{69}{128} x 100\%=53\%$$ |
| 256 | 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 10011110011001000101101111101111001010110010000001010100100000100111100100100110101101101000110011000010111101010000101011010110 | $$\frac{62}{128} x 100\%=48\%$$ |

|  |
| --- |
| Compress (.rar) |
| Jenis Kunci | Plaintext | Ciphertext | Avalanche Effect |
| 128 | 01010010011000010111001000100001000110100000011100000001000000000101101000011111010100001010100000001100000000010000010100001000 | 11111011110001110101110101100100010001111111101110100111100010000100011101011011011110010010101111110101111001101100101111011000 | $$\frac{65}{128} x 100\%=50\%$$ |
| 192 | 01010010011000010111001000100001000110100000011100000001000000000101101000011111010100001010100000001100000000010000010100001000 | 01010000111010111000110110111111111011101001011110011011010100100110001111111110010000110011110000001101110100001111010001010111 | $$\frac{61}{128} x 100\%=47\%$$ |
| 256 | 01010010011000010111001000100001000110100000011100000001000000000101101000011111010100001010100000001100000000010000010100001000 | 10110000011101100110100001100110110100101001100011101110100010111000001010011000010100001001001111000001111101100110100110110001 | $$\frac{69}{128} x 100\%=53\%$$ |

|  |
| --- |
| Documents (.docx) |
| Jenis Kunci | Plaintext | Ciphertext | Avalanche Effect |
| 128 | 01010000010010110000001100000100000010100000000000000000000000000000000000000000100001110100111011100010010000000000000000000000 | 00111000011111011101100001110001011100001011100100011110011011111000010010101011100100111010011010110111110010111011010011011110 | $$\frac{69}{128} x 100\%=53\%$$ |
| 192 | 01010000010010110000001100000100000010100000000000000000000000000000000000000000100001110100111011100010010000000000000000000000 | 11010110101110001010101010100100110110011101100100011001010000001010101111111001110100000001110100000110111001010010100100110010 | $$\frac{63}{128} x 100\%=49\%$$ |
| 256 | 01010000010010110000001100000100000010100000000000000000000000000000000000000000100001110100111011100010010000000000000000000000 | 00010101010100100111101100101011111101000111000011101111110010001100110100100001000101101110101011011100111100010001101001110010 | $$\frac{64}{128} x 100\%=50\%$$ |

|  |
| --- |
| Image (.jpg) |
| Jenis Kunci | Plaintext | Ciphertext | Avalanche Effect |
| 128 | 11111111110110001111111111100001101011010111010101000101011110000110100101100110000000000000000001001001010010010010101000000000 | 01011000001010100001001101110101101111010100110010100111000000000000101101010010011010111010111111110101110110010000101011111001 | $$\frac{62}{128} x 100\%=48\%$$ |
| 192 | 11111111110110001111111111100001101011010111010101000101011110000110100101100110000000000000000001001001010010010010101000000000 | 01001001011000001100000111100010011000110101100011000111001001111101010111100010001011100011111100110010111111011101010000000101 | $$\frac{69}{128} x 100\%=53\%$$ |
| 256 | 11111111110110001111111111100001101011010111010101000101011110000110100101100110000000000000000001001001010010010010101000000000 | 11010001000011011111011101100001100011000001001111110111110011001001011111110000110011000101010010111001110001111101010101010100 | $$\frac{62}{128} x 100\%=48\%$$ |

|  |
| --- |
| Video (.mp4) |
| Jenis Kunci | Plaintext | Ciphertext | Avalanche Effect |
| 128 | 00000000000000000000000000011000011001100111010001111001011100000110110101110000001101000011001000000000000000000000000000000000 | 01001011001010100110111000001110000011010110101101011001001101011110100010010001011001001101010110100110010100011101100111010010 | $$\frac{60}{128} x 100\%=46\%$$ |
| 192 | 00000000000000000000000000011000011001100111010001111001011100000110110101110000001101000011001000000000000000000000000000000000 | 01111011111000111100011100100000110101000011010011001100100110010110101011001101100110001100101011000100111110101110111001100011 | $$\frac{71}{128} x 100\%=55\%$$ |
| 256 | 00000000000000000000000000011000011001100111010001111001011100000110110101110000001101000011001000000000000000000000000000000000 | 01000111000111010111111111010101101110101111001010110110000101110111110011110011001001110010100000010010001101010011010111101010 | $$\frac{65}{128} x 100\%=50\%$$ |

Berdasarkan hasil pengujian avalanche effect, algoritma AES-Rijndael yaitu AES-128, AES-192, dan AES 256. Algoritma AES-Rijndael adalah algoritma yang baik, karena memiliki rentang nilai avalanche effect 45%-60%.

**Uji Coba Kemungkinan**

**Pemecahan Kunci**

Pada uji coba ini dilakukan perhitungan matematis brute force attack. Brute force attack adalah kalkulasi peluang untuk pemecahan kunci. Kemungkinan kunci yang di hasilkan dari AES-128 adalah 3,4 x 1038 , AES-192 adalah 6,2 x 1057 dan AES-256 adalah 1,2 x 1077 . Asumsi kecepatan komputasi adalah 106 key/sec. dan perhitungan detik dalam 1 tahun adalah 3,1536 x 107 .

$$\frac{Kemungkinan Kunci}{Asumsi Kecepatan Komputasi}=\frac{Kemungkinan Kunci}{3,1536 x 10^{7}}$$

$$\frac{3,4 x 10^{38}}{10^{6}}=\frac{3,4 x 10^{32}}{3,1536 x 10^{7}}=1,08 x 10^{25} tahun$$

Gambar 18. Simulasi Perhitungan Brute Force Attack AES-128

Tabel 11. Tabel Kemungkinan Waktu Pemecahan Kunci

|  |  |  |
| --- | --- | --- |
| No | Panjang Kunci | Kemungkinan Waktu Pemecahan Kunci |
| 1 | 128 | 1,08 x 1025 tahun |
| 2 | 192 | 1,97 x 1044 tahun |
| 3 | 256 | 3,8 x 1063 tahun |

Berdasarkan perhitungan matematis brute force attack, membuktikan bahwa panjang kunci yang digunakan AES-256 memiliki waktu pemecahan kunci yang sangat lama, hal ini dikarenakan AES-256 melakukan lebih banyak proses perulangan transformasi bytes yaitu sebanyak 14 perulangan sehingga AES-256 memiliki tingkat keamanan yang lebih baik.

1. **Penutup**

**Kesimpulan**

Setelah dilakukan analisis, perancangan, implementasi, serta evaluasi aplikasi pengenkripsian dengan menggunakan algoritma Rijndael dapat ditarik kesimpulan hasil enkripsi merupakan hasil dari suatu transormasi bytes yang di kombinasikan dengan bytes kunci, sehingga bytes dari file asli teracak. Size file hasil enkripsi akan bertambah beberapa bytes, karena ada proses penambahan bytes pada saat pembentukan block cipher atau operasi padding. Berdasarkan pengujian kompleksitas waktu dengan notasi Big O, algoritma AES Rijndael untuk enkripsi file, memiliki notasi O(n), artinya besaran masukan berpengaruh terhadap waktu proses. Berdasarkan pengujian perubahan bit menggunakan metode Avalanche Effect, untuk AES-128, AES-192, dan AES-256 memiliki nilai Avalanche Effect 45%-60%. Nilai tersebut mengindikasikan algoritma AES Rijndael algoritma yang baik, dan aman terhadap serangan kriptoanalis. Dengan perhitungan matematis brute force attack membuktikan AES-256 memiliki tingkat ketahanan yang lebih tinggi dan lebih aman yaitu memiliki kemungkinan waktu untuk pemecahan kunci sebesar 3,8 x 108 tahun. Aplikasi yang telah dibuat telah berhasil di uji coba untuk enkripsi dan dekripsi file yaitu image(.jpg), documents(.docx), audio (.mp3), video(.mp4), compress (.rar), dan aplikasi (.exe) dengan ukuran maksimal sebesar 30 MB.

**Saran**

Saran-saran yang dapat membantu dalam aplikasi pengenkripsian dengan menggunakan algoritma Rijndael adalah sebagai berikut:

1. Dapat di kombinasikan dengan algoritma kriptografi lainnya, agar dapat meningkatkan keamanan file.
2. Diimplementasikan pada platform lain, yaitu i-phone dan desktop PC.
3. Meningkatkan size file yang dapat di enkripsi dan dekripsi.

**DAFTAR PUSTAKA**

Ana Kurniawati & Muhammad Dwiky Darmawan. 2015. Implementasi Algoritma Advanced Encryption Standard (AES) Untuk Enkripsi dan Dekripsi Pada Dokumen Teks..

Andry. 2011. Android A sampai Z. PCplus, Jakarta.

Bagus Satrio Waluyo Putro & Retantyo Wardoyo. 2014. Perbandingan Efisiensi, Efektifitas dan Kualitas Algoritma Rijndael dengan Algoritma Camellia pada Citra Digital. Berkala MIPA.

Bayu Rizki, R. Rumani M & Muhammad Nasrun. 2015. Analisis Perbandingan Antara Algoritma Kriptografi Serpent dan Aes Pada Implementasi Enkripsi Sms di Perangkat Android. E-Proceeding Engineering. Vol. 2 ISSN 2355-9365.

Haris Mufti Maulana & Pulung Nurtantio Andono. 2014. Implementasi Algoritma AES-256 Bit Untuk Enkripsi Dan Deskripsi Pesan Teks Berbasis Android.

Hariyanto, B. 2009. Sistem Operasi. Bandung : Informatika.

Kristanto, Andri. 2003. Keamanan Data pada Jaringan Komputer. Yogyakarta: Penerbit Gava Media.

Lee, W. M. 2011. Beginning Android Application Development. Wiley Publishing,Inc.

Michell Setyawati Handaka. 2011. Analisi AES Rijndael terhadap DES.

Mollin, R. A. 2007. An Introduction to Cryptography. 2nd ed. Florida: Chapman & Hall/CRC.

Munir, Rinaldi. 2006. Diktat Kuliah Kriptografi. Bandung: Program Studi Tenik Informatika, Institut Teknologi Bandung.

Munir, Rinaldi. 2008. Belajar Ilmu Kriptografi. Yogyakarta: Penerbit Andi.

Munir, Rinaldi. 2011. Kriptografi Keamanan. Bandung: Informatika Bandung.

Rahmat Tullah, Muhammad Iqbal Dzulhaq & Yudi Setiawan. 2016. Perancangan Aplikasi Kriptografi File Dengan Metode Algoritma Advanced Encryption Standard (AES). Jurnal Sisfotek Global. Vol. 6 No. 2 2088-1762.

Sadikin, Rifki. 2012. Kriptografi Untuk Keamanan Jaringan.Yogyakarta: Penerbit Andi