BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Kajian Pustaka dalam penelitian ini berupa bahasan tentang penelitian-penelitian yang relevan terhadap penelitian yang dilakukan. Penelitian yang dikaji meliputi enkripsi SMS dengan menggunakan 3D-AES Block Cipher pada aplikasi perpesanan Android serta implementasi AES untuk enkripsi SMS pada android.

Penelitian pertama yang berjudul *"SMS Encryption using 3D-AES Block Cipher on Android Message Application"*. Pada penelitian ini membahas tentang enkripsi SMS pada aplikasi Android dengan menggunakan algoritma kriptografi 3D-AES block cipher. 3D-AES block cipher berdasarkan pada AES block cipher dengan kunci yang terdiri dari fungsi rotasi kunci dengan minimal 3 iterasi dari operasi *round function* dan *key mixing.* Pada penelitian ini didapatkan hasil 3D-AES memiliki waktu enkripsi dan dekripsi yang lebih lama dibandingkan dengan AES 128-bit dengan plaintext berukuran antara 32 sampai 128 bit. 3D-AES block cipher memiliki waktu enkripsi dan dekripsi yang lebih cepat dibandingkan dengan AES 128-bit jika ukuran plaintext diatas 256 bit.

Pada penelitian kedua yang berjudul "*SMS Encryption using AES Algorithm on Android* " membahas tentang implementasi metode AES 128-bit untuk enkripsi SMS pada android. Hasil penelitian yang telah dilakukan adalah terbentuknya program enkripsi SMS pada Android dengan *speed requirement* yang telah terpenuhi, *user interface* yang sederhana, otentifikasi pengirim pesan, serta kemungkinan untuk mendeteksi jika pesan telah rusak atau dirusak selama proses transmisi. Aplikasi ini menjamin keamanan proses *end-to-end* transfer data tanpa meyebabkan *corrupt* pada data segmen.

Dalam penelitian ini, akan digunakan metode *Elliptic Curve Diffie-Hellman* serta *Advance Encryption Standard* sebagai solusi untuk melakukan enkripsi pesan SMS pada telepon seluler berbasis android. Metode ECDH digunakan sebagai metode pertukaran kunci rahasia sebelum melakukan enkripsi. Kunci rahasia hasil dari metode ECDH tersebut yang nantinya akan digunakan sebagai kunci private dalam proses enkripsi dengan mengunakan algoritma AES. Perbandingan objek dan penelitian dari masing-masing referensi akan ditunjukkan pada Tabel 2.1

Tabel 2.1 Tabel Kajian Pustaka

No	Judul	Objek	Metode	Hasil
1.	*SMS Encryption Using 3D-AES Block Cipher on Android Message Application* (Suriyani Ariffin, 2013)	*Short Message Service (SMS)*	3D-AES block cipher	Hasil perbandingan waktu enkripsi dan dekripsi 3D-AES lebih cepat dari AES 128-bit untuk panjang plaintext diatas 256 bit.

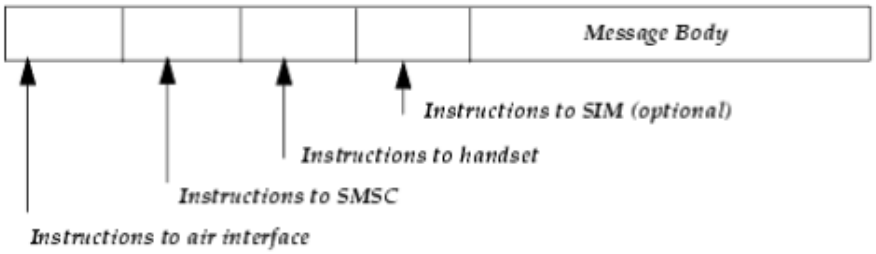
2.	*SMS Encryption using AES Algorithm on Android* (Rohan Rayarikar, 2012)	*Short Message Sevice (SMS)*	Advance Enryption Standard (AES)	Aplikasi enkripsi SMS berbasis android dengan menggunakan algoritma enkripsi AES 128-bit
3.	Usulan Penulis: Enkripsi SMS Pada Telepon Seluler Berbasis Android Dengan Menggunakan Metode ECDH dan AES	*Short Message Sevice (SMS)*	Advance Enryption Standard (AES)	Proses algoritma yang digunakan dalam sistem mampu menghasilkan pengamanan SMS pada telepon seluler berbasis android yang mempunyai tingkat kelayakan yang baik

2.2 SMS

Pengertian SMS (*Short Message Service*) adalah salah satu *Instant Messaging* (IM) yang memungkinkan pengguna bertukar pesan singkat kapanpun walaupun pengguna sedang melakukan panggilan data atau suara (Prihartini, 2006).

2.2.1 Struktur SMS

Struktur pesan pada sebuah paket SMS dapat dilihat pada Gambar 2.1.



Gambar 2.1 Struktur SMS (Prihartini, 2006)

SMS terdiri dari *header* dan *message boy. Header* pada SMS merupakan kumpulan instruksi-instruksi yang berupa informasi yang diperlukan dalam pengiriman pesan, sedangkan pada bagian *message body* berisi pesan yang akan diterima.

Pada gambar2.1 terlihat bahwa pada sebuah paket SMS terdiri dari *header* dan *message body*. *Header* pesan terdiri dari instruksi-instruksi kepada komponen-komponen yang bekerja dalam jaringan SMS. Pada instruksi-instruksi tersebut, terdapat informasi yang diperlukan selama pengiriman pesan seperti informasi validitas pesan, dan informasi-informasi lainya. Pada bagian *message body* terdapat isi dari pesan yang akan dikirimkan (Clements, 2003).

Panjang maksimal dari isi suatu pesan pada sebuah paket SMS berukuran maksimal 160 karakter, dimana setiap karakter memiliki panjang 7 bit. Beberapa

aplikasi standar telepon selular dapat mendukung panjang pesan dengan karakter sepanjang 8 bit (panjang pesan maksimum 140 karakter) dan karakter yang lebih panjang lainya seperti 16 bit namun karakter sepanjang 8 bit dan 16 bit ini tidak didukung oleh semua aplikasi standar telepon selular. Pada umumnya karakter sepanjang 7 bit dan 8 bit digunakan untuk menampilkan data seperti gambar dan simbol (Pettersom, 2007).

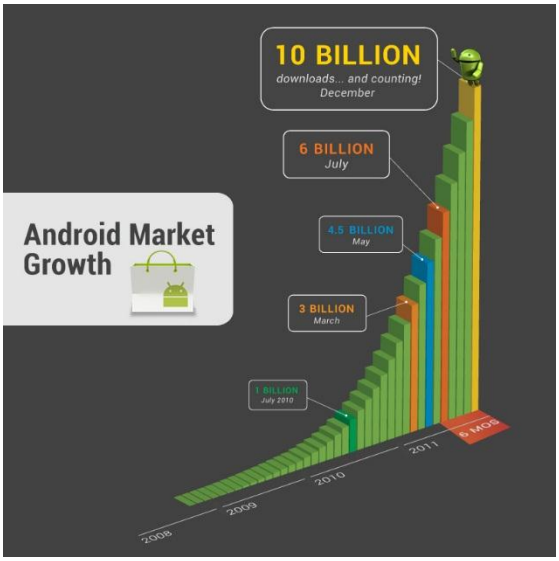
2.2.2 Arsitektur SMS

SMS merupakan alat pertukaran informasi antara dua *mobile subscriber*. Elemen-elemen utama pada arsitektur SMS terdiri dari *Short Message Entity* (SME) yang dapat mengirim atau menerima pesan singkat, *SMS Service Center* (SMSC) yang berfungsi menyampaikan pesan singkat antara SME dengan MS, juga menyimpan dan meneruskan pesan ingkat dan *E-mail Gateway* yang terkoneksi dengan elemen-elemen pada GSM sebagai *channel* penghantar (Prihartini, 2006)

2.3 Android

Android merupakan system informasi yang biasa disematkan pada gadget baik itu smartphone maupun Tabelt (Wahadyo, 2012). Android bersifat terbuka yang didukung oleh Google dan bersifat multi-perangkat yang maksudnya dari banyak macam perangkat nantinya hanya membutuhkan satu platform saja (Simon, 2011). Android dibangun menggunakan kombinasi dari pemrograman java dan layout berbasis XML serta konfigurasi dari beberapa file (Ostrander, 2012).

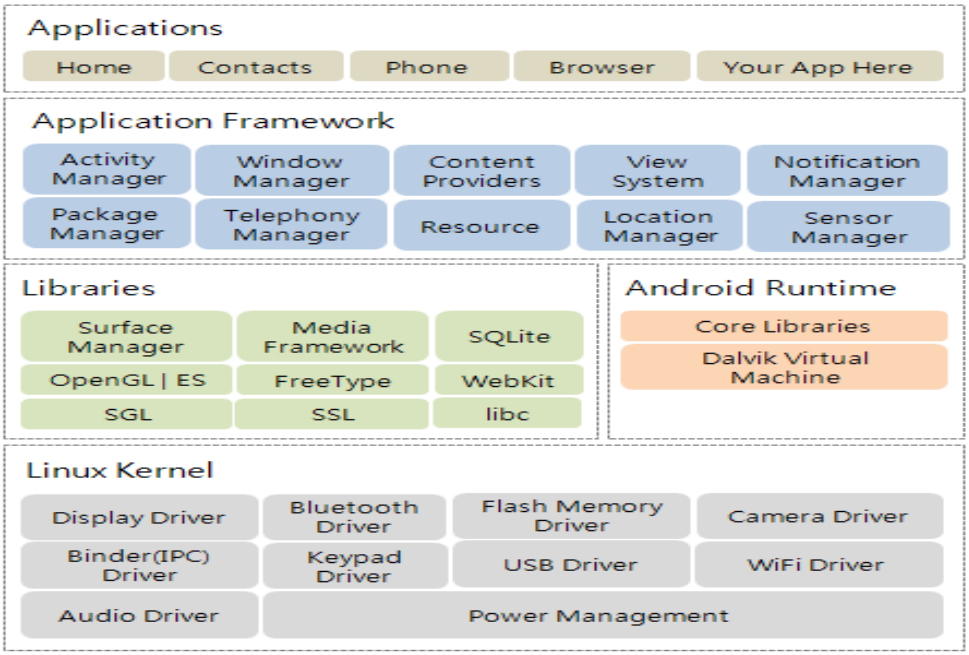
Perkembangan android sangat cepat. Pada Mei 2011, total pengguna android di dunia mencapai seratus juta penguna. Dari seratus juta pengguna, terdapat 400.000 pengguna baru setiap harinya. Aplikasi yang tersedia dan siap pakai mencapai 200.000 aplikasi dan total 4,5 triliun kali aplikasi diunduh dari android Market (Wahadyo, 2012).



Gambar 2.2 Perkembangan Android

2.3.1 Konsep Platform Android

Sistem pada android terdiri dari banyak tumpukan. Dari tumpukan-tumpukan itu dapat memberikan kemudahan bagi pengembang untuk mengakses perangkat-perangkat keras dalam aplikasinya namun tetap pada aturan atau framework yang telah ditentukan (Vogel, 2014).



Gambar 2.3 Simulasi konsep pekerjaan Android (Vogel, 2014)

Pada Gambar 2.3 diatas didapatkan 5 level komponen platform android. Berikut penjelasan masing-masing komponen:

- *Application*
  Level ini merupakan level dimana banyak aplikasi yang dibuat oleh pengembang-pengembang yang kini bermunculan.
- *Application Framework*
  Level ini merupakan API dimana yang memungkinkan interaksi level yang tinggi dari aplikasi android dengan sistem android.
- *Libraries & Runtime*
  Level ini merupakan library untuk kerangka aplikasi sebagai fungsi umum (seperti penyimpanan data) serta menjadi library inti dari Java untuk menjalankan aplikasi android.
- *Hardware Abstraction Layer*
  HAL berfungsi sebagai antarmuka standar yang memungkinkan sistem android untuk memanggil layer device driver. Implementasi HAL biasanya dibangun ke modul *shared library* (.so files)
- *Linux Kernel*
  Kernel merupakan lapisan paling bawah pada arsitektur android. Dalam arsitektur android, kernel merupakan *abstraction layer* antara hardware dan keseluruhan software. Linux merupakan sistem operasi yang handal

dalam manajemen memori dan proses. Kernel linux menyediaka driver layer, kamera, keypad, WiFi, *Flash Memory*, audio, dan IPC (*Interprocess Communication*) untuk mengatur aplikasi dan lubang keamanan.

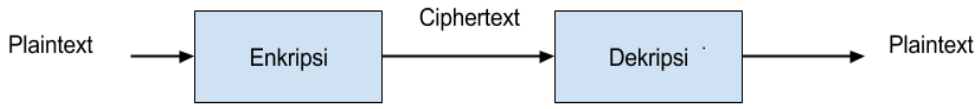
2.3.2 Konsep Pengembangan Android

Konsep dari pengembangan android sendiri hampir sama dengan pengembangan-pengembangan aplikasi lain yaitu menggunakan *Application Programming Interface* (API), terkadang juga menggunakan *Java Database Connectivity* (JDBC) atau framework lain yang berfungsi sebagai jembatan untuk mengakses MySQL atau database lainya. Namun perbedaan dari pengembangan android dan pengembangan aplikasi lain adalah pengemasan dan struktur untuk membuat ponsel lebih tahan dari tabrakan aplikasi yaitu diantaranya adalah: (Allen, 2012).

- Aktifitas
  Setiap pembangunan antarmuka pengguna menggunakan sebuah aktifitas android dirancang untuk mendukung banyak kegiatan kecil, sehingga dapat memungkinkan pengguna untuk terus menekan ikon untuk membuka kegiatan baru dan menekan tombol back untuk kembali, seperti yang mereka lakukan di *web browser*.
- Layanan
  Layanan merupakan kegiatan yang berumur pedek dan dapat ditutup setiap saat. Disisi lain, layanan dirancang untuk terus berjalan bahkan bias bersifat independen dari aktifitas apapun. Layanan mirip dengan *daemon* pada sistem operasi lain.
- *Content Providers*
  Content Providers memberikan tingkat abstraksi untuk data yang tersimpan pada perangkat untuk dapat diakses oleh beberapa aplikasi. Model pembangunan pada android mendukung untuk pengembang membuat data sendiri baik untuk aplikasi lain ataupun tentunya untuk aplikasi yang dikembangkan itu sendiri.
- Intents
  Intents adalah sistem notifikasi yang berjalan pada perangkat baik perubahan perangkat keras (misalnya ketika SMS masuk). Pengguna tidak hanya merespon notifikasi tersebut, namun juga dapat membuat sendiri sistem notifikasi.

2.4 Kriptografi

Pesan (*message*) merupakan data yang dapat dibaca serta dimengerti maknanya. Pesan dapat disebut juga dengan *plaintext.* Enkripsi merupakan proses untuk menyembunyikan pesan dalam bentuk lain yang tidak mudah dimengerti untuk menyembunyikan substansinya (Schneier, 1996). *Ciphertext* merupakan pesan yang telah dilakukan proses enkripsi dan proses untuk mengembalikan *ciphertext* menjadi *plaintext* disebut dengan dekripsi. Keterkaitan tersebut digambarkan pada Gambar 2.4.



Gambar 2.4 Enkripsi dan dekripsi pada *plaintext* (Schneier, 1996)

Menurut Stallings (2005), kriptografi adalah bagian dari kriptologi yang berhubungan dengan desain algoritma untuk enkripsi dan dekripsi yang bertujuan untuk metakinkan kerahasiaan dan keaslian pesan.

Konsep penggunaan kriptografi menurut Fidens (2006) yaitu :

1. Kerahasiaan (*Confidentiallity*) yaitu proses penyembunyian data dari orang-orang yang tidak mempunyai otoritas
2. Integritas (*Integrity*) yaitu prises untuk menjaga agar sebuah data tidak diubah-ubah waktu dikirim maupun disimpan.
3. Penghindaran penolakan (*Non-Repudiation*) yaitu proses untuk menjaga bukti-bukti bahwa suatu data berasal dari seseorang.
4. Autentikasi (*Autenthication*) yaitu proses untuk menjamin keaslian data.

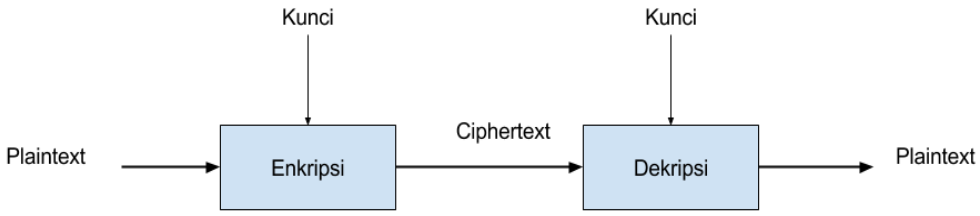
2.4.2 Algoritma Kriptografi

Algoritma kriptografi (*chipper*) adalah fungsi matematika yang digunakan untuk enkripsi dan dekripsi (Schneier, 1996). Berdasarkan kunci yang dipakai, algoritma kriptografi dibedakan atas 2 jenis, yakni:

1) Algoritma kunci simetris (*symmetric-key cryptographic*)

Dalam algoritma kunci simetris, kunci yang digunakan untuk melakukan enkripsi sama dengan kunci yang digunakan untuk dekripsi (Schneier, 1996). Istilah lain yang digunakan untuk algoritma ini adalah algoritma kunci privat. Keamanan kriptografi ini terletak pada kerahasiaan kuncinya. Contoh algoritma kunci simetris adalah DES (*Data Encryption Standard*), *Blowfish*, *Twofish*, dan *AES*.

Proses enkripsi dan dekripsi algoritma kunci simetris dapat dilihat pada Gambar 2.5



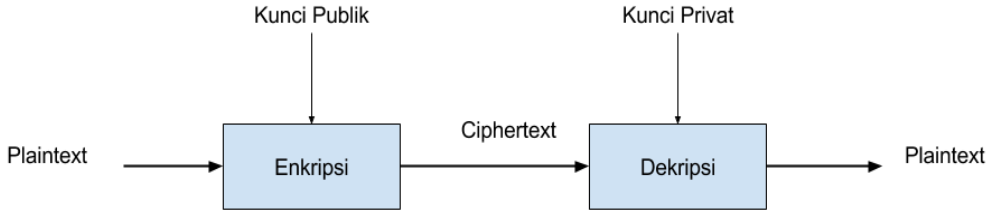
Gambar 2.5 Enkripsi dan dekripsi algoritma kunci simetris (Schneier, 1996)

Algoritma kriptografi simetris dibagi menjadi dua jenis yaitu algoritma aliran (*stream ciphers*) serta algoritma blok (*block ciphers*). Pada algoritma aliran (*stream ciphers)*, proses enkripsi berorientasi pada satu bit atau satu *byte* data. Sedangkan

untuk algoritma blok (*block cipher*), proses enkripsi berorientasi pada sekumpulan bit atau *byte* data (per blok).

2) Algoritma kunci asimetris

Algoritma kunci asimetris merupakan algoritma dengan kunci enkripsi serta dekripsi yang berbeda (Schneier, 1996). Algoritma ini disebut juga algoritma kriptografi kunci publik. Dalam algoritma ini, setiap orang yang memiliki kunci publik dapat melakukan proses enkripsi suatu pesan, data ataupun informasi, namun untuk melakukan proses dekripsi dari *ciphertext* hanya orang yang memiliki kunci privat. Proses enkripsi dan dekripsi algoritma kunci asimetris dapat dilihat pada Gambar 2.6



Gambar 2.6 Enkripsi dan dekripsi algoritma kunci asimetris (Schneier, 1996)

2.5 Diffie-Hellman Key Exchange

Diffie-Hellman merupakan suatu algoritma kunci public yang pertama kali ditemukan pada tahun 1976. Algoritma ini memperoleh keamananya dari sulitnya menghitung logaritma diskrit pada bilangan yang sangat besar. Algoritma Diffie-Hellman hanya dapat digunakan untuk pertukaran kunci (simetri) dan tidak dapat digunakan untuk enkripsi dan dekripsi maupun untuk tanda tangan digital (Hendra, 2014).

Berikut merupakan parameter umum yang digunakan dalam mekanisme pertukaran kunci diffie-hellman, yaitu:

1. Terdapat sedikitnya dua orang user yang berinteraksi.
2. Adanya kesepakatan terhadap bilangan prima yang besar, n dan g sedemikian sehingga $g < n$
3. Bilangan n dan g tidak perlu rahasia, sehingga antar user dapat bertukar melalui saluran yang tidak aman sekalipun.

Berikut merupakan ilustrasi proses penukaran kunci Diffie-Hellman dengan dua orang user, yaitu:

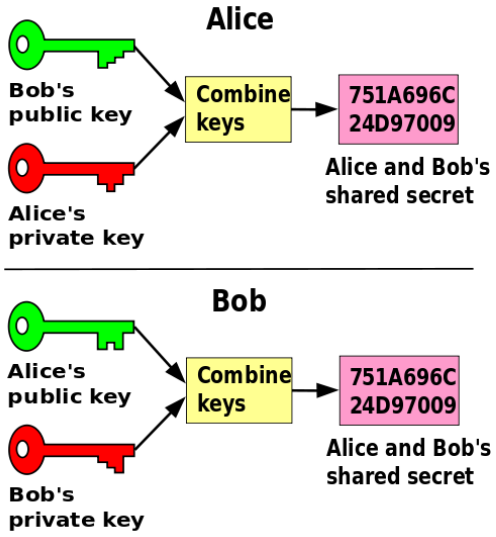
1. User pertama memilih secara acak sebuah bilangan big integer a dan mengirimkanya ke user kedua.

$$A = g^a mod\ p \qquad (2.8)$$

2. User kedua juga memilih secara acak sebuah bilangan big integer b dan mengirimkanya ke user pertama.

$$B = g^b mod\ p \qquad (2.9)$$

3. User pertama menghitung nilai $s1 = B^a \, mod \, p$, user kedua juga menghitung nilai $s2 = A^b \, mod \, p$
4. Baik user pertama dan user kedua kini memiliki nilai $s1$ dan $s2$ yang sama. Dimana $A^b \, mod \, p = g^{ab} \, mod \, p = g^{ba} \, mod \, p = B^a \, mod \, p$



Gambar 2.7 Mekanisme Diffie-Hellman (Schneier, 1996)

2.5.1 Elliptic Curve Cryptography (ECC)

Elliptic curve diperkenalkan pertama kali oleh Victor Miller dan N. Koblitz sebagaimana alternative terhadap sistem kunci publik seperti DSA serta RSA. *Elliptic curve cryptography* merupakan salah satu bentuk kriptografi asimetri. Kriptografi asimetri memungkinkan penggunaan kunci yang berbeda untuk enkripsi dan dekripsi data (Hendra, 2014).

Kelompok kurva eliptik yang digunakan dalam kriptografi adalah didefinisikan melalui dua jenis dari *field* yaitu $GF(p)$ serta $GF(2^m)$. $GF(p)$, dimana p merupakan suatu prima dan $GF(2^m)$ dimana masing-masing elemen adalah suatu binary polymonial dengan derajat m (yang dapat dinyatakan sebagai string m-bit karena masing-masing koefisien adalah 0 atau 1). Persamaan kurva eliptik secara umum yaitu

$$E: y^2 = x^3 + ax + b. \tag{2.10}$$

Nilai kurva elliptic merupakan bilangan real. Operasi bilangan real mempunyai kekurangan yaitu tidak secepat operasi bilangan bulat dan kurang presisi karena terdapat pembulatan. Salah satu solusinya adalah dengan menggunakan kurva elliptik yang berada di *prime field* (medan bilangan prima). Persamaan kurva elliptic di medan bilangan prima adalah sebagai berikut:

$$E(Fp): y^2 mod \, p = x^3 + ax + b \, mod \, p \tag{2.11}$$

Nilai a, b dan p pada persamaan diatas merupakan parameter domain dari ECC di medan bilangan prima. Selain medan bilangan prima, pada ECC juga terdapat *binary field* (medan bilangan biner). Persamaan kurva elliptik pada medan bilangan biner adalah sebagai berikut:

$$E(F2^m): y^2 + xy = x^3 + ax + b, \text{b} \neq 0 \qquad (2.12)$$

Beberapa algoritma yang menggunakan ECC antara lain ECDSA dan ECDH. ECDSA merupakan singkatan Elliptic Curve Digital Signature Algorithm. ECSA digunakan unutk membuat serta melakukan verifikasi tanda tangan digital. ECDH merupakan singkatan dari Elliptic Curve Diffie Hellman. ECDH digunakan untuk melakukan pertukaran kunci diatas kanal yang tidak aman.

2.5.2 Elliptic Curve Domain Parameter

Domain parameter yang terdapat pada *Elliptic Curve* yaitu (p, a, b, G, n, h) untuk prima, dan $(m, f(x), a, b, G, n, h)$ untuk binary. Kunci privat K_a adalah suatu bilangan bulat yang telah disetujui pleh masing – masing pihak serta dipilih secara acak dalam jangkauan $[1, n - 11]$) dan sebuah kunci public K_aG(dimana $K_aG = K_a\ x\ G$ sehingga pasangan kunci untuk user pertama adalah (k_a, k_aG) dan pasangan kunci untuk user kedua adalah (k_b, k_bG), dan masing-masing user melakukan pertukaran kunci public (k_aG, k_aG) dan kemudian masing-masing menghitung sebuah *shared secret* dengan melakukan perkalian kunci privat dengan kunci publik pihak lawan, karena $(k_ak_bG = k_a(k_bG) = k_b(k_aG)$. Satu-satunya informasi yang diekspose oleh kedua belah pihak user adalah kunci publik, maka tidak ada pihak selain kedua user tersebut yang mengetahui kunci privat mereka masing-masing, kecuali pihak tersebut dapat memecahkan *Elliptic curve Discreate Algorithm problem.* Banyak protokol standard berbasis ECDH menurunkan suatu kunci simetris dari k_ak_bG menggunakan *hash-based key derivation function* (Hendra, 2014).

Tidak semua kurva pada parameter domain aman, hal tersebut dikarenakan untuk mendapatkan kurva yang aman dibutuhkan perhitungan yang memakan waktu serta sulit untuk diimplementaskan. Sebagai hasilnya berbagai badan internasional menerbitkan parameter domain kurva eliptik untuk beberapa ukuran field yang umum, sehingga domain parameter sering dikenal dengan "standard curves" atau "named curves". Hal tersebut mengacu berdasarkan nama atau pengenal unik yang di definisikan pada dokumen standard. Domain parameter memiliki dua jenis parameter yaitu parameter yang sesuai dengan kurva Koblitz(k) dan parameter terverifikasi yang dipilih secara random (r). Parameter k lebih mudah diimplementasikan daripada parameter r. Pemakaian kurva standar adalah terkait dengan alasan keamanan, karena tidak semua kurva eliptik aman untuk digunakan, dan pemakaian kurva standar juga untuk menjaga *interoperable* (Hendra, 2014).

Parameter domain atau kurva standar yang digunakan dalam penelitian ini adalah dengan secp256k1. Kurva standar secp256k1 digunakan pada *ellpitical*

curve yang didefinisikan pada *Standards for Efficient Cryptography (SEC)*. Kurva standar secp256k1 didesain untuk membuat komputasi yang dilakukan lebih efisien. Kurva standar secp256k1 merupakan domain parameter prima, sehingga secp 256k1 memiliki domain parameter berupa (p, a, b, G, n, h) dengan properti sebagai berikut

- $P =$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFC2F

  $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^1$
- $a =$ 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
- $b =$ 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007
  - G (*base point*) $=$ 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
  - n (*order*) $=$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
  - h (*cofactor*) $= 01$

Karena a bernilai 0, maka persamaan kurva menjadi E: $y^2 = x^3 + ax + b$ (Brown, 2010).

2.5.3 Elliptic Curve Diffie-Hellman Key Exchange

Elliptic Curve Diffie-Hellman (ECDH) merupakan sebuah protokol perjanjian kunci anonim yang memungkinkan dua pihak, A dan B, untuk membangun kunci rahasia bersama (*share secret key*) melalui saluran yang tidak aman, di mana masing-masing pihak memiliki pasangan kunci public dan kunci private berbasis *elliptic curve*. *Share secret* tersebut nantinya dapat digunakan sebagai kunci untuk kriptografi kunci simetris (Hendra, 2014).

Dalam penjelasan diatas, telah dijelaskan secara terpisah mengenai algoritma Elliptic Curve dan Diffie-Hellman. Berikut merupakan mekanisme ECDH dalam mendapatkan nilai *shared secret*, yaitu:

a. Pertama kali, diperlukan persetujuan antara pengguna A dan B mengenai parameter domain elliptic curve yang digunakan.

b. Pengguna A memilih sebuah integer $PT(A)$ sebagai kunci privat. Setelag itu pengguna melakukan *generate* kunci publik dengan $PU(A) = PT(A)x\ G$, dimana $PT(AP)x\ G$ merupakan $(G + G + \cdots + G)$ sebanyak $PT(A)$ kali. Setelah itu, pengguna A mengirimkan kunci publik ke pengguna B.

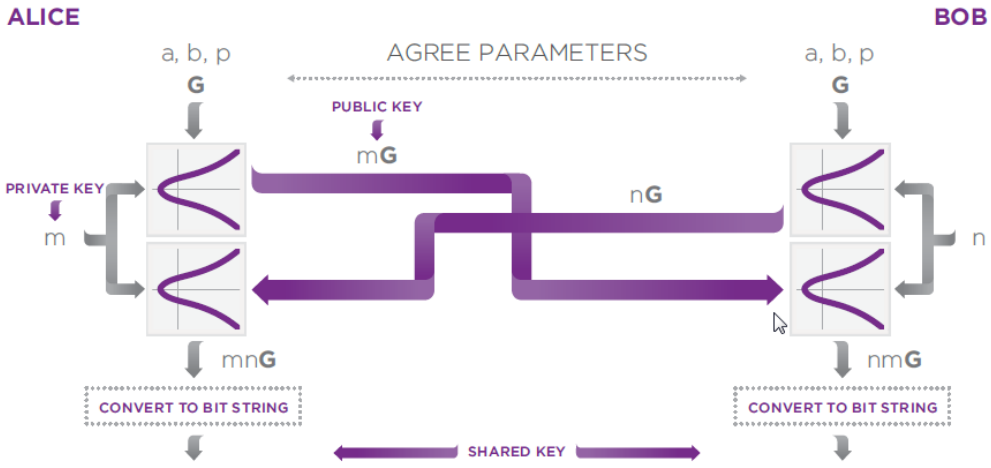
c. Pengguna B kemudian memilih sebuah intege$PT(B)$ sebagai kunci privat. Setelah itu pengguna melakukan *generate* kunci publik $PU(B) = PT(B)x\ G$ dimana $PT(B)\ x\ G$ merupakan $(G + G + \cdots + G)$

sebanyak $PT(B)$ kali. Pengguna B kemudian mengirimkan kunci publik kepada user A.

d. Pengguna A melakukan kalkulasi kunci *shared secret,* yaitu:
   - $K = PT(A)x\ PU(B)$

e. Pengguna B melakukan kalkulasi kunci *shared secret,* yaitu:
   - $K = PT(B)x\ PU(A)$

f. Kedua pengguna A dan B akan mendnapatkan kunci *shared secret* yang sama.



Gambar 2.8 Proses Elliptic Curve Diffie-Hellman (Hendra, 2014)

2.6 AES

2.6.1 Input dan Output

Input dan output untuk AES masing-masing terdiri dari deret 128 bit (digit-digit dengan nilai 0 atau 1). Deret ini kadang disebut sebagai blok dan jumlah bit di dalamnya disebut sebagai panjangnya. *Cipher key* (*key* yang digunakan untuk enkripsi atau dekripsi) untuk AES adalah deret 128 bit, 192 bit, atau 256 bit. Bit-bit di dalam deret ini dihitung mulai 0 dan berakhir pada panjang deret dikurangi 1. Angka I yang diberikan pada sebuah bit disebut juga sebagai indeksnya, dan akan berada dalam *range* $0 \le i \le 128$, $0 \le i \le 192$, atau $0 \le i \le 256$ bergantung pada panjang bok atau kunci (Federal Information Processing Standards Publication, 2001).

2.6.2 *byte*

Unit dasar untuk pemrosesan dalam AES adalah *byte*, sebuah deret yang terdiri dari 8 bit yang dianggap sebagai satu entitas (Federal Information Processing Standards Publication, 2001). Deret input, output dan *key* diproses sebagai *array bytes* yang dibentuk dengan cara membagi deret tadi menjadi grup-grup 8 bit berurutan untuk membentuk *array bytes*. Untuk sebuah *input, output* atau *key*

yang disimbolkan dengan a, *bytes* dalam *array* yang dihasilkan ditunjuk menggunakan dua jenis bentuk, a_n atau a[n], dimana ada di antara *range* berikut:

Panjang *key* =128 bit, $0 \leq n \leq 16$;

Panjang *key* =192 bit, $0 \leq n \leq 24$;

Panjang *key* =256 bit, $0 \leq n \leq 32$;

Sebuah nilai *byte* dalam AES ditampilkan sebagai gabungan tiap-tiap bit individual (0 atau 1) di dalam tanda kurung dengan urutan $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. *Byte-byte* ini diterjemahkan sebagai elemen *finite field* menggunakan representasi polynomial:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i\, x^i$$

Misalnya {01100011} mengidentifikasi elemen *finite field* yang spesifik yaitu $x^6 + x^5 + x + 1$ (Federal Information Processing Standards Publication, 2001).

Cukup mudah juga untuk menampilkan nilai *byte* menggunakan notasi heksadesimal dengan tiap-tiap dua grup 4 bit direpresentasikan oleh satu karakter, dapat dilihat pada Tabel 2.2

Tabel 2.2 Representasi heksadesimal dari suatu bit

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

Maka berdasarkan tabel 2.2, elemen {01100011} dapat direpresentasikan sebagai {63}, dimana karakter yang menunjukkan grup 4 bit dengan bit bernilai tinggi berada di kiri. Beberapa operasi *finite field* melibatkan satu bit tambahan (b_8) di ujung kiri dari sebuah *byte* yang tersusun dari 8 bit. Saat bit tambahan ini ada, ia akan tampil sebagai {01} sebelum *byte*; contoh, sebuah deret 9 bit akan ditampilkan sebagai {01}{1b} (Federal Information Processing Standards Publication, 2001).

2.6.3 *Array bytes*

Array bytes (*array* yang berisi sejumlah *bytes*) direpresentasikan pada bentuk berikut:

$a_0 a_1 a_2 \dots\dots\dots a_{15}$

Urutan *byte-byte* dan bit dalam *bytes* didapatkan dari sebuah rangkaian input berukuran 128 bit sebagai berikut:

$input_0 input_1 input_2 \dots\dots\dots input_{126} input_{127}$

$a_0 = input_0 input_1 \dots\dots\dots input_7;$

$a_0 = input_8 input_9 \dots\dots\dots input_{15}$

.

.

.

$a_{15} = input_{120} input_{121} \dots\dots\dots input_{127};$

Pola tersebut dapat diperluas menjadi rangkaian yang lebih panjang (contoh: untuk *key* dengan ukuran 192 dan 25 bit), sehingga bentuk umum dari pola rangkaian akan menjadi:

$a_{15} = input_{8n} input_{8n+1} \dots\dots\dots input_{8n+7};$

2.6.4 *State*

Pada dasarnya, operasi-operasi algoritma AES dilakukan pada sebuah *array bytes* dua dimensi yang disebut *state*. *State* terdiri dari 4 baris *byte,* setiap baris terdiri dari Nb buah *byte*, dimana Nb adalah panjang blok dibagi dengan 32 (Federal Information Processing Standards Publication, 2001). Pada *array state* yang dinotasikan dengan simbol S, setiap *byte* tunggal memiliki dua buah parameter, dimana nomor baris r berada pada *range* $0 \leq r \leq 4$ dan nomor kolom c berada pada *range* $0 \leq c \leq Nb.$ Hal ini memungkinkan sebuah *byte* tunggal dari *state* untuk dapat dinyatakan sebagai $S_{r,c}$ atau $S[r, c]$ sebagai standar yang digunakan disini Nb = 4, yang berarti jumlah kolom c adalah rentang $0 \leq c \leq 4$ (Munir, 2006).

Pada awal proses enkripsi (*cipher*) dan deskripsi (*inverse cipher*), input yaitu *array bytes* $in_0, in_0, \dots in_{15} -$ disalin ke dalam *array state* sebagaimana yang diilustrasikan pada Gambar 2.4. Operasi-operasi *cipher* dan *invers cipher* kemudian dilakukan pada *array state* tersebut, sehingga didapat nilai akhir yang kemudian disalin ke dalam *output – array bytes* $out_0, out_1, \dots out_{15}$ (Federal Information Processing Standards Publication, 2001).

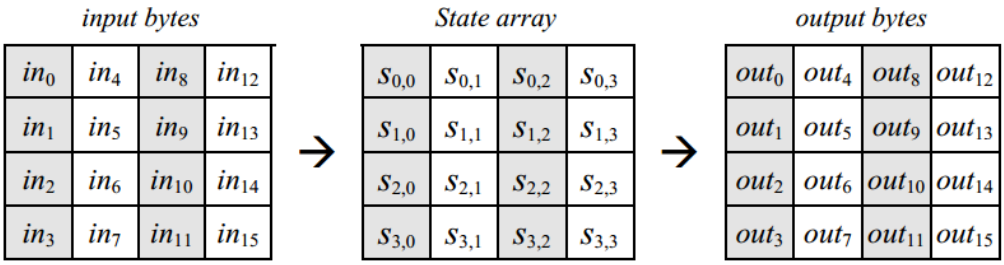
Dengan demikian, pada proses awal dari enkripsi (*cipher*) dan deskripsi (*inverse cipher*), array input,in, disalin ke dalam array state dengan skema berikut:

$S[r, c] \leftarrow in[r + 4c]$ untuk $0 \leq r \leq 4$ dan $0 \leq c \leq 4\ Nb$

Dan pada proses akhir enkripsi dan dekripsi, state disalin ke dalam array output out sebagai berikut:

$out[r + 4c] \leftarrow S[r, c]$ untuk $0 \leq r \leq 4$ dan $0 \leq c \leq 4\ Nb$ (Munir, 2006).

Gambar 2.9 menunjukkan keterkaitan antara *input bytes*, *state array* dan *Output bytes*



Gambar 2.9 *State array*, Input dan Output

2.6.5 Algoritma kriptografi AES

Algoritma kriptografi AES (*Advance Encryption Standard*) yang sering juga disebut dengan algoritma kriptografi Rijndael sama seperti algoritma kriptografi DES yang menggunakan subtitusi, permutasi serta sejumlah putaran (*cipher* berulang). *Round key* merupakan proses putaran kunci internal yang berbeda dalam setiap putaranya (Munir, 2006).

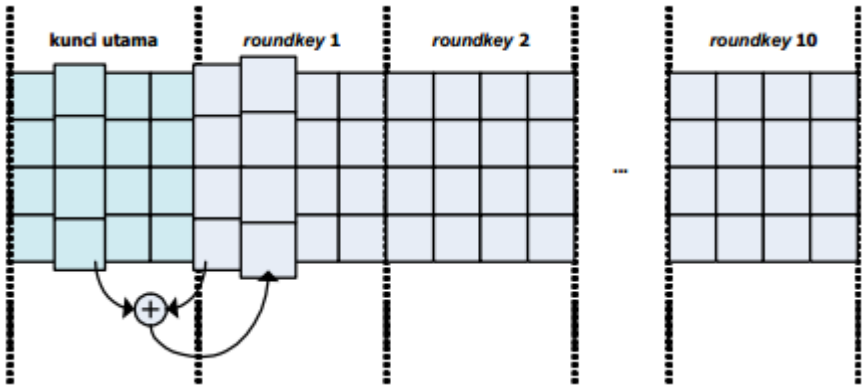
Algoritma kriptografi AES termasuk dalam klasifikasi algoritma kriptografi kunci simetri, kunci yang digunakan pada enkripsi sama dengan kunci yang digunakan untuk dekripsi. Berbeda dengan DES yang berorientasi bit, Rijndael beroprasi pada *byte*. Algoritma AES dapat bekerja dalam tiga macam ukuran yakni 128 bit (16 *byte*), 192 bit (24 *byte*) dan 256 bit (32 *byte*) (Kurniawan, 2007). Skripsi ini akan membandingkan antara AES 128 bit, AES 192 bit, dan AES 256 bit.

2.6.6 Proses Inisialisasi Kunci Internal

Dalam algoritma kriptografi AES terdapat n kali iterasi bergantung pada panjang kunci, dimana masing-masing iterasi akan menggunakan kunci yang berbeda. Untuk AES 128 bit diperlukan 11 kali iterasi serta 10 kunci internal yang berbeda $(K_1, K_2, \dots, K_{10})$. Kunci internal ini diturunkan dari kunci eksternal yang berukuran 128 bit (array 4 x 4 dengan masing-masing elemen sebesar 1 byte). Kunci terdiri atas kunci utama (kunci eksternal yang diinputkan user) dan 10 buah kunci putaran (*round key*) (Munir, 2006).

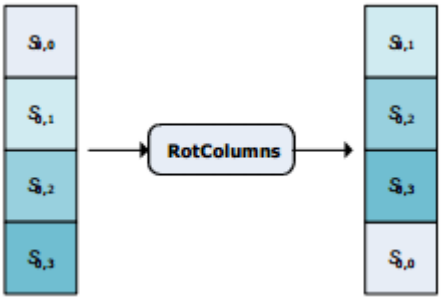
Proses inisialisasi kunci internal pada algoritma kriptografi AES terdiri dari dua bagian, yaitu proses untuk memperoleh nilai elemen-elemen pada kolom kedua hingga kolom keempat untuk masing-masing *round key* dan proses untuk memperoleh nilai elemen-elemen pada kolom pertama (Kurniawan, 2007).

Gambar 2.10 menunjukkan proses untuk memperoleh nilai elemen-elemen pada kolom kedua hingga keempat. Untuk kolom kedua hingga keempat dari masing-masing *round key*, elemen-elemenya diperoleh dari hasil XOR antara satu kolom sebelumnya dengan kolom yang sama pada *array* kunci sebelumnya



Gambar 2.10 Skema proses inisialisasi kunci pada AES

Untuk kolom pertama pada masing-masing *round key*, setiap elemen diperoleh melalui tiga transformasi yang dilaksanakan secara berurutan yaitu *RotCol*, *SubBytes* dan XOR. Transformasi *RotCol* adalah operasi perputaran elemen berdasarkan barisnya, baris pertama menjadi baris keempat, baris kedua menjadi baris pertama, baris ketiga menjadi baris kedua, dan baris keempat menjadi baris ketiga (Munir, 2006). Dapat dilihat pada Gambar 2.11.



Gambar 2.11 Skema proses *RotCol* pada AES

Hasil dari transformasi tersebut menjadi masukan untuk transformasi *SubBytes,* yakni operasi subtitusi yang dilakukan terhadap masing-masing *byte* dengan nilai yang ditunjukkan pada matriks subtitusi *S-Box*. Transformasi *SubBytes* dalam proses inisialisasi kunci internal pada AES ini sama dengan transformasi *SubBytes* pada proses enkripsi (Stalling, 2005).

Hasil dari transformasi *SubBytes* akan di-XOR-kan dengan kolom pertama pada kunci sebelumnya dan *Rcon* untuk *round key* tersebut. *Rcon* adalah *array 4* x 1 yang merupakan deretan konstanta yang digunakan dalam proses menghasilkan *round key*, dapat dilihat pada Tabel 2.5. Dari proses inisialisasi kunci internal akan diperoleh 11 kunci internal yang diperoleh dari kunci eksternal yang diinputkan oleh user dan 10 *round key*. Ukuran untuk setiap kunci internal sebesar 128 bit dengan format *array* 4 x 4 dengan masing-masing elemen sebesar 1 *byte* (Kurniawan, 2007).

Tabel 2.3 Tabel *Rcon* untuk 10 *round key* AES

Rcon$_1$	*Rcon*$_2$	*Rcon*$_3$	*Rcon*$_4$	*Rcon*$_5$	*Rcon*$_6$	*Rcon*$_7$	*Rcon*$_8$	*Rcon*$_9$	*Rcon*$_{10}$
01	01	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

2.6.7 Contoh proses inisialisasi kunci internal

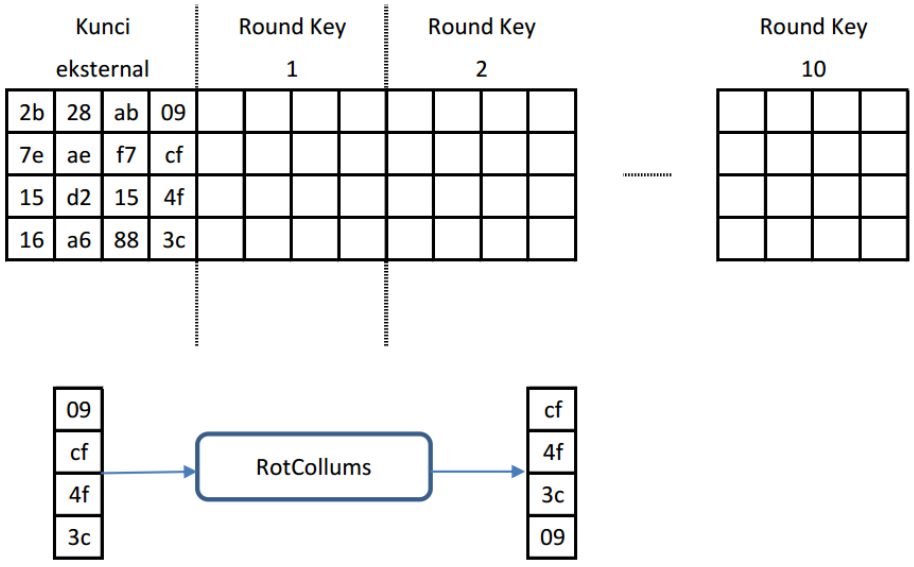
Berikut ini adalah contoh tahapan dalam proses inisialisasi kunci internal:

1) Misalkan *array* kunci eksternal diberikan oleh Gambar 2.12.

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

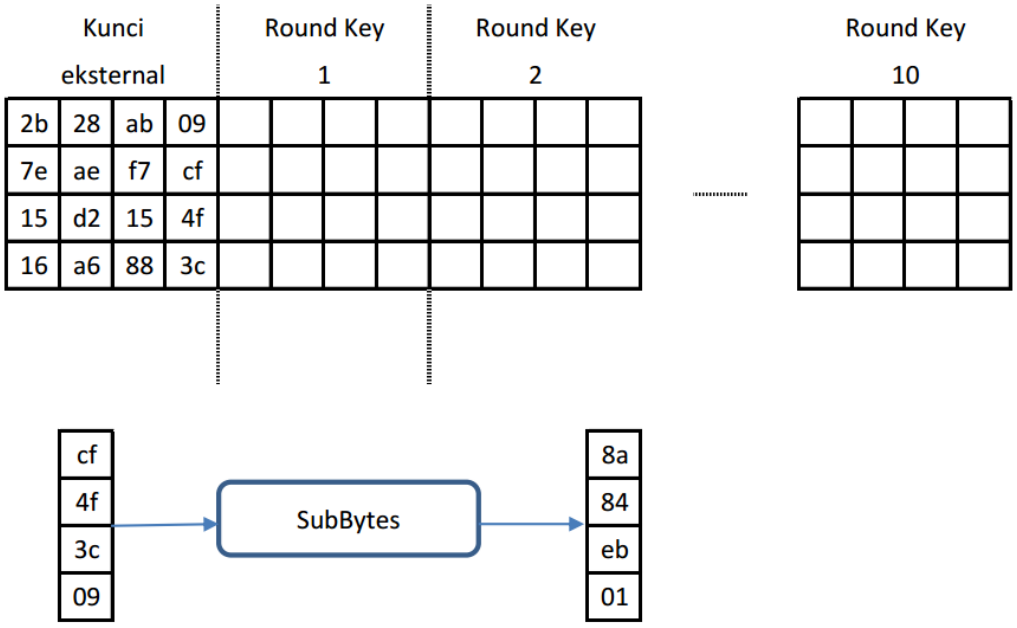
Gambar 2.12 Contoh *array* kunci eksternal

2) Untuk memperoleh kolom pertama pada *round key* 1, dilakukan transformasi *RotCol* pada kolom terakhir dari *array* kunci eksternal (Gambar 2.13).



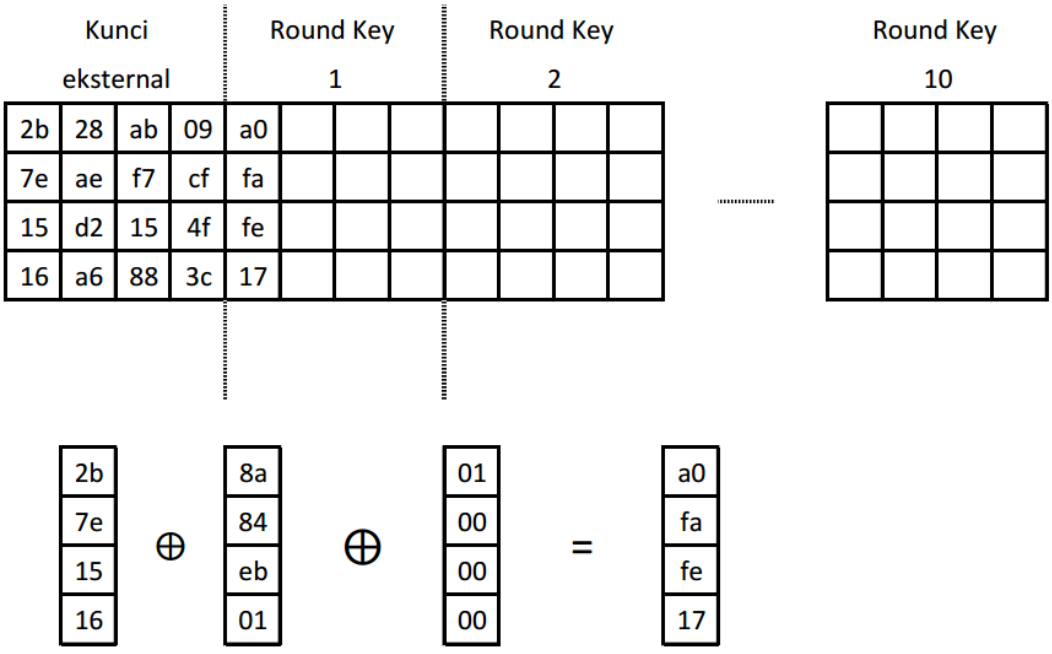
Gambar 2.13 Contoh transformasi *RotCol*

3) Transformasi *SubBytes* pada kolom yang terakhir *array* kunci eksternal yang sudah dilakukan transformasi *RotCol* (Gambar 2.14).



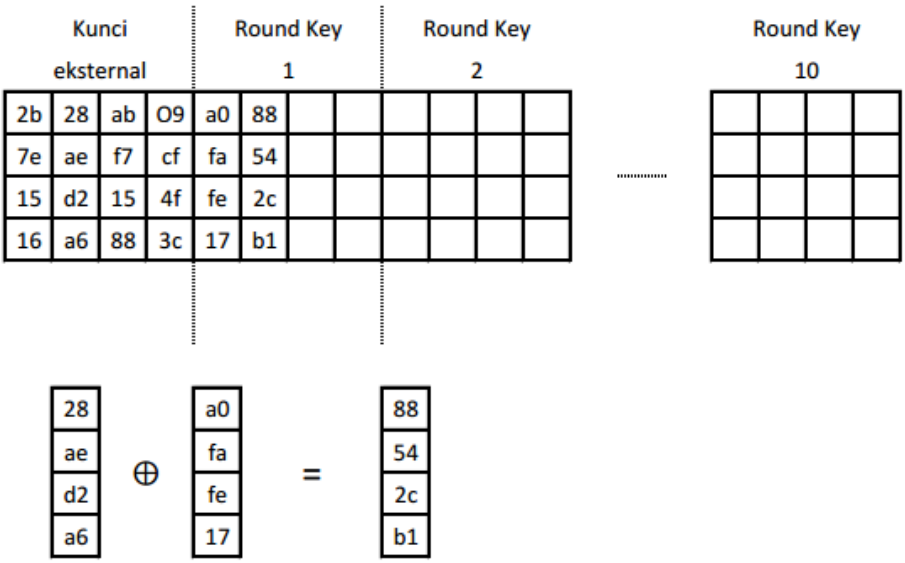
Gambar 2.14 Contoh transformasi *SubBytes*

4) Kolom ke-1 *array* kunci eksternal di-XOR-kan dengan kolom hasil *SubBytes* tersebut dan *Rcon*1, dan kemudian hasilnya merupakan kolom pertama dari *round key* 1 (Gambar 2.15).



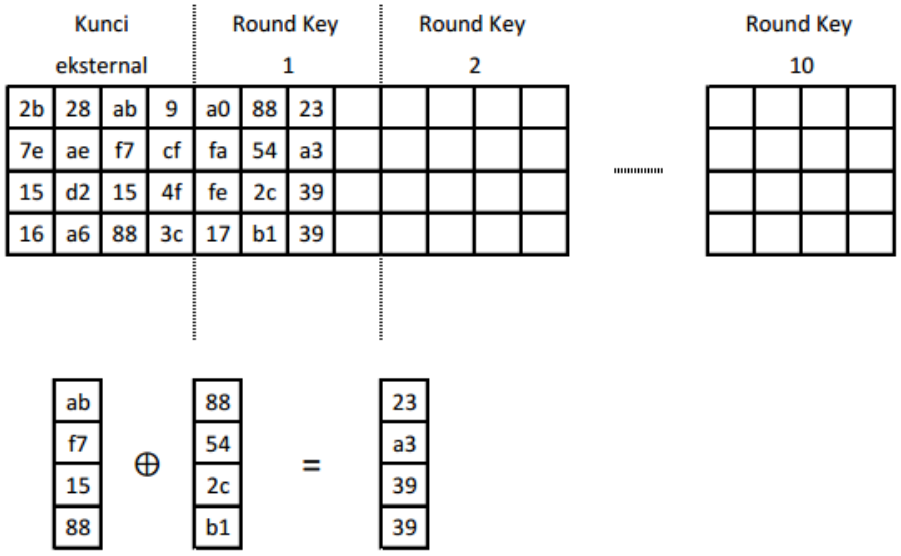
Gambar 2.15 Cara memperoleh kolom pertama *round key* 1

5) Untuk memperoleh kolom kedua dari *round key* 1, dilakukan peng-XOR-an kolom pertama *round key* 1 dengan kolom kedua array kunci eksternal (Gambar 2.16).



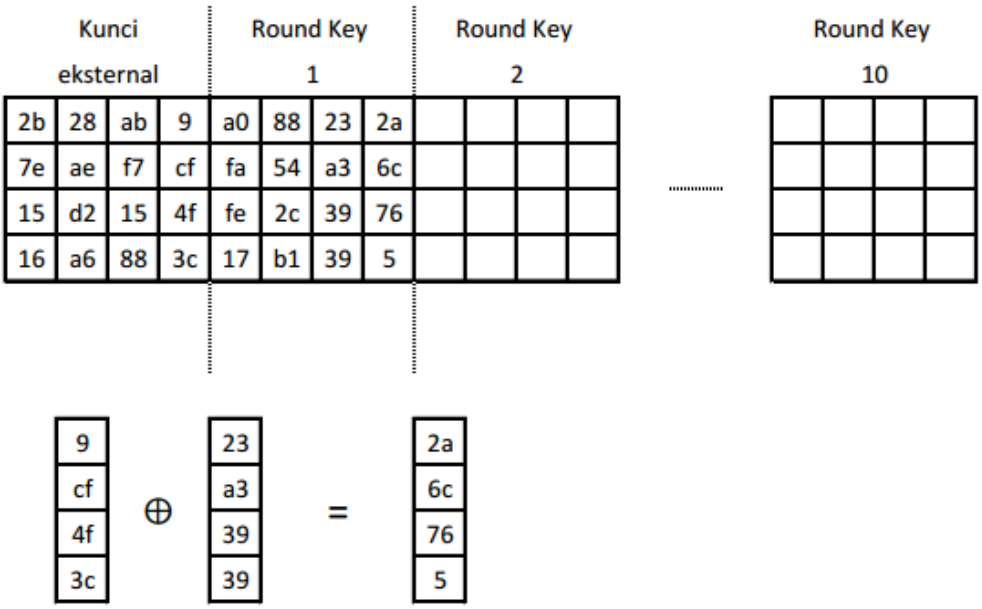
Gambar 2.16 Cara memperoleh kolom kedua *round key* 1

6) Untuk memperoleh kolom ketiga dari *round key* 1, dilakukan peng-XOR-an kolom kedua *round key* 1 dengan kolom ketiga array kunci eksternal (Gambar 2.17).



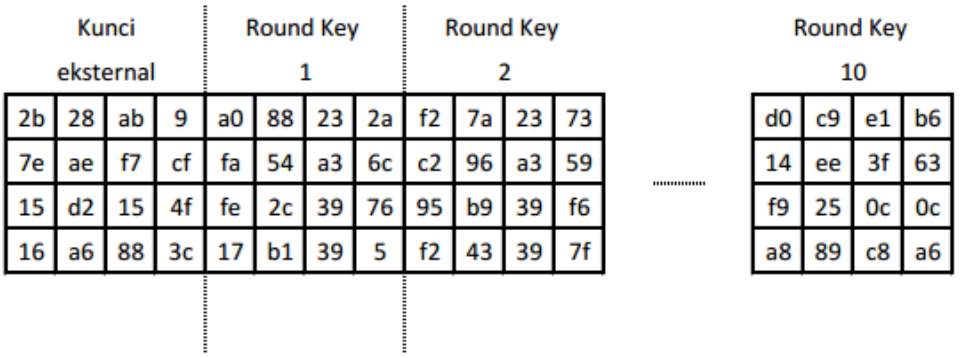
Gambar 2.17 Cara memperoleh kolom ketiga *round key* 1

7) Untuk memperoleh kolom keempat dari *round key* 1, dilakukan peng-XOR-an kolom ketiga *round key* 1 dengan kolom keempat array kunci eksternal (Gambar 2.18).



Gambar 2.18 Cara memperoleh kolom keempat *round key* 1

8) Langkah 2 hingga 7 diulang 9 kali untuk memperoleh *round key* 2 hingga *round key* 10 (Gambar 2.19).



Gambar 2.19 Kunci eksternal dan *round key*

2.6.8 Proses enkripsi

Garis besar algoritma AES yang beroprasi pada blok 128 bit dengan kunci 128 bit adalah sebagai berikut:

1) *Initial round*. Dalam proses ini terdapat proses *AddRoundKey*. Proses *AddRoundKey* merupakan proses hasil operasi XOR antara *plaintext* dengan kunci (*ciphertext)*.

2) Putaran sebanyak N_r-1 kali. Proses yang dilakukan pada setiap putaran adalah:

   a. *SubBytes*: melakukan penggantian *byte-byte* dalam state dengan menggunakan Tabel *S-Box*.

   b. *ShiftRows:* proses menggeser baris-baris *array state* secara *wrapping.*

c. *MixColumns*: melakukan proses pengacakan *byte-byte* pada masing-masing kolom *array state.*

d. *AddRoundKey*: melakukan proses XOR antara *state* sekarang dengan kunci *round key*.

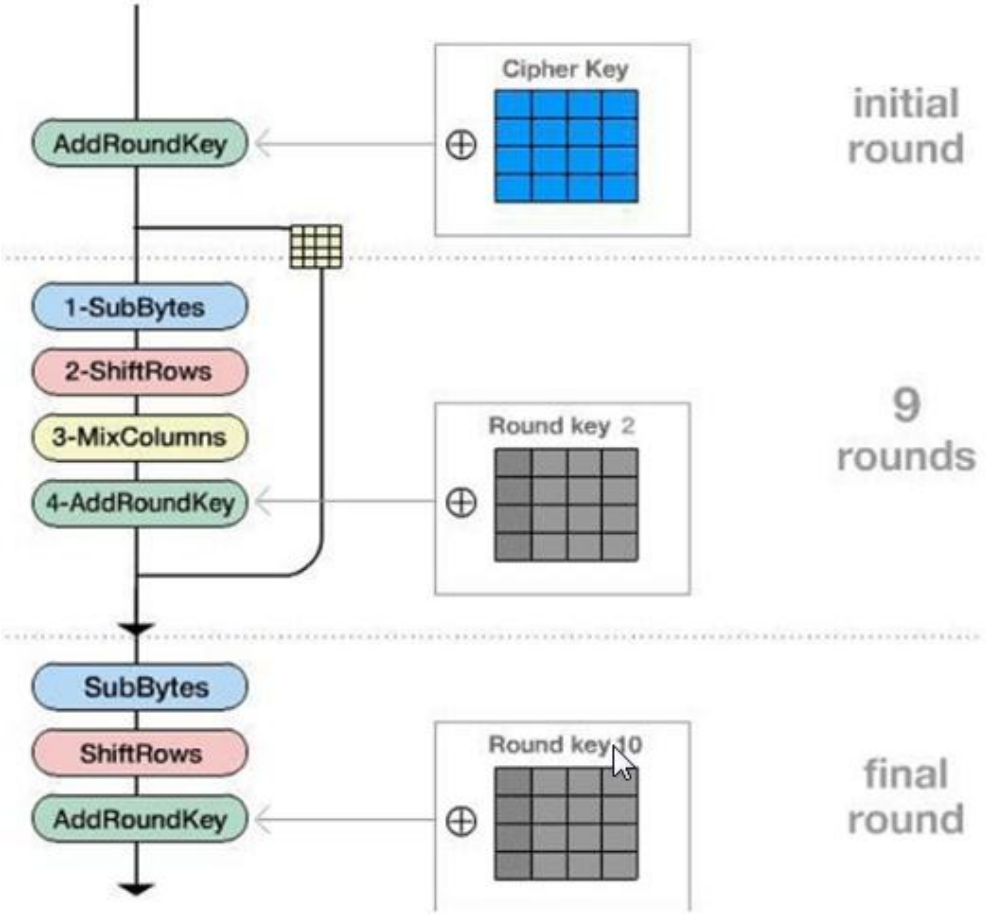
3) *Final round*: merupakan proses putaran terakhir. Dalam proses ini terdapat 3 proses yang dilakukan secara beruntun, yakni:

a. *SubBytes*

b. *ShiftRows*

c. *AddRoundKey*

Skema proses enkripsi AES dapat dilihat pada Gambar 2.20



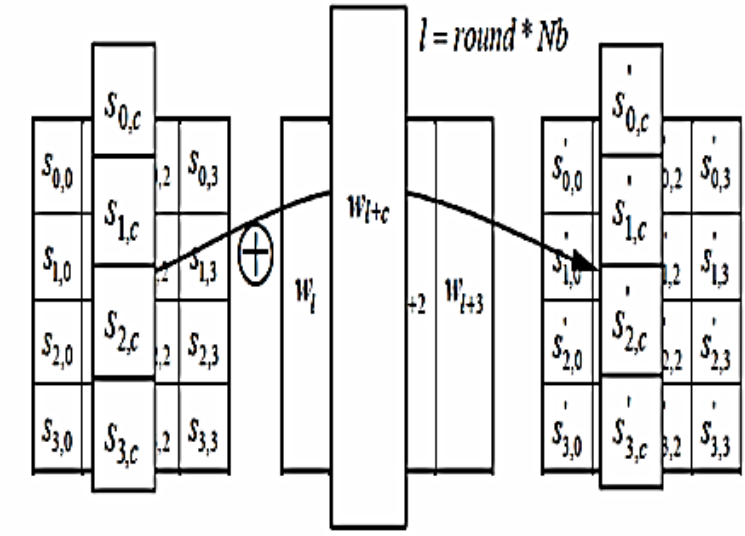
Gambar 2.20 Skema proses enkripsi AES (Kurniawan, 2007)

Proses enkripsi dalam algoritma kriptografi AES dilakukan sebanyak 10 kali putaran, 1 kali *initial round*, 9 kali *round(s)* dan 1 kali *final round* untuk AES 128. Pada AES 192 dilakukan sebanyak 12 kali putaran, terdiri dari 1 kali *initial round*, 11 kali *round(s)* dan 1 kali *final round.* Pada AES 256 terdiri dari 14 kali putaran, dengan satu kali *initial round*, 13 kali *round(s)* dan 1 kali *final round*.

Algoritma AES memiliki 3 parameter:

1) *Plaintext*: *array* yang berukuran 16 byte, yang berisi data masukan.
2) *Ciphertext*: *array* yang berukuran 16 byte, yang berisi hasil enkripsi.
3) *Key*: *array yang berukuran* 16 *byte,* 24 *byte,* ataupun 32 *byte* yang berisi kunci *ciphering*(disebut juga *cipher key*) (Munir, 2006).

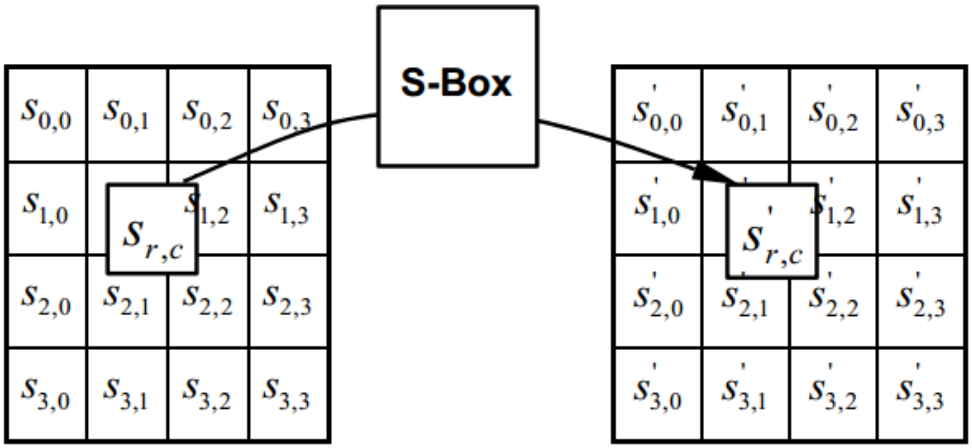
Dalam *intial round*, transformasi *AddRoundKey*() dilakukan terhadap kunci utama. Sedangkan dalam 10 *round* yang lain, proses *AddRoundKey* dilakukan terhadap kunci putaran (*round key*). Proses *AddRoundKey* didefinisikan sebagai operator XOR antara *array state* dengan *round key*. Operasi XOR dilakukan pada masing-masing *byte* dalam *array* sehingga menghasilkan nilai baru pada *array* hasil dengan ukuran *array* hasil sama dengan ukuran *array state* awal dan *array key*, yaitu sebesar 4x4. Hasil untuk masing-masing baris dan kolom pada *array state* diperoleh dari hasil operasi XOR antara *array state* awal dengan *array key* untuk baris dan kolom yang sama. Ilustrasi dari proses *AddRoundKey* dapat dilihat pada Gambar 2.21 (Kurniawan, 2007).



Gambar 2.21 Transformasi *AddRoundKey* (Gladman, 2001)

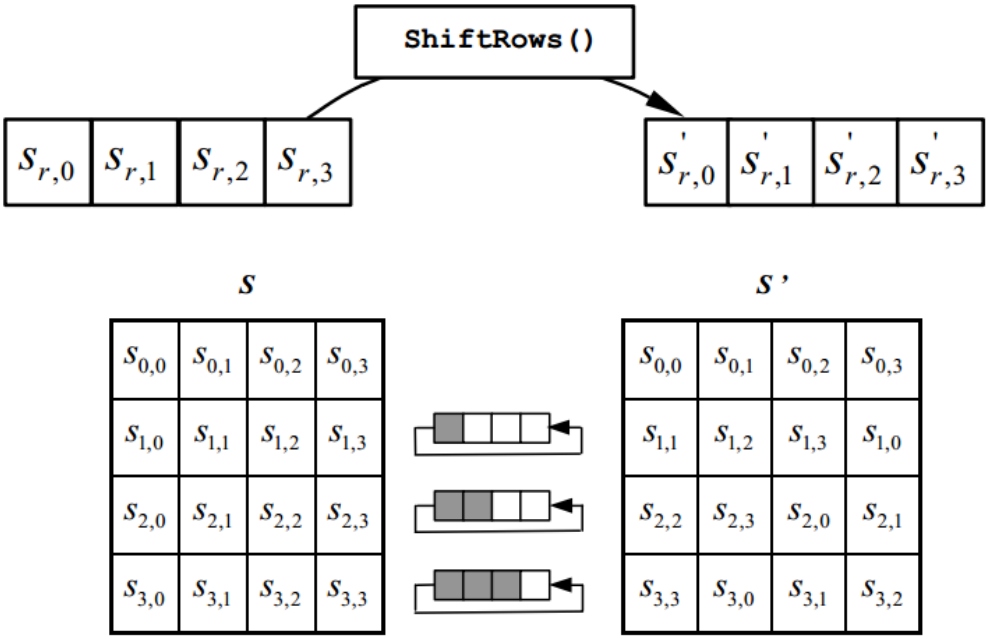
Transformasi *SubBytes*() memetakan setiap *byte* dari *array state* dengan menggunakan Tabel subtitusi *S-Box.* Tabel *S-Box* dapat dilihat pada bagian lampiran.

Cara pensubtitusian adalah sebagai berikut: untuk setiap *byte* pada *array*,misalkan S[r,c] = xy, maka nilai subtitusinya, yang dinyatakan dengan S[r,c], adalah elemen di dalam *S-Box* yang merupakan perpotongan baris x dengan kolom y. Gambar 2.22 menunjukkan transformasi *SubBytes* (Munir, 2006).



Gambar 2.22 Tranformasi *SubBytes* (Gladman, 2001)

Transformasi *ShiftRows*() melakukan pergeseran secara *wrapping* (siklik) pada 3 baris terakhir dari *array state*. Jumlah pergeseran bergantung pada nilai baris 9r0. Baris r = 1 digeser sejauh 1 *byte*, baris r = 2 digeser sejauh 2 *byte*, dan baris r = 3 digeser sejauh 3 *byte*. Baris r = 0 tidak digeser. Gambar 2.23 memperlihatkan transformasi *ShiftRows.*



Gambar 2.23 Transformasi *ShiftRows* (Gladman, 2001)

Transformasi *MixColumns*() dilakukan setelah transformasi *ShiftRows*. Proses *MixColumns()* merupakan sumber utama dari difusi pada algoritma AES. Difusi adalah prinsip yang menyebarkan pengaruh satu bit *plaintext* atau kunci ke sebanyak mungkin *ciphertext* sehingga tidak dapat di prediksi. Prinsip difusi juga menyembunyikan hubungan statistic antara *plaintext*, *ciphertext*, dan kunci sehingga membuat kriptanalisis menjadi sulit (Munir, 2006).

Transformasi *MixCoumns*() adalah proses mengalikan setiap kolom dari *aray state* dengan polinom $a(x) mod\ (x^4 + 1)$. Setiap kolom diperlakukan sebagai polinom 4 suku pada $GF(2^8)$. Polinom $a(x)$yang ditetapkan pada persamaan 2.4

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (2,4)$$

Berikut merupakan perkalian matriks seperti pada persamaan 2.5

$$s'(x) = a(x) \oplus s(x) \quad (2.5)$$

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

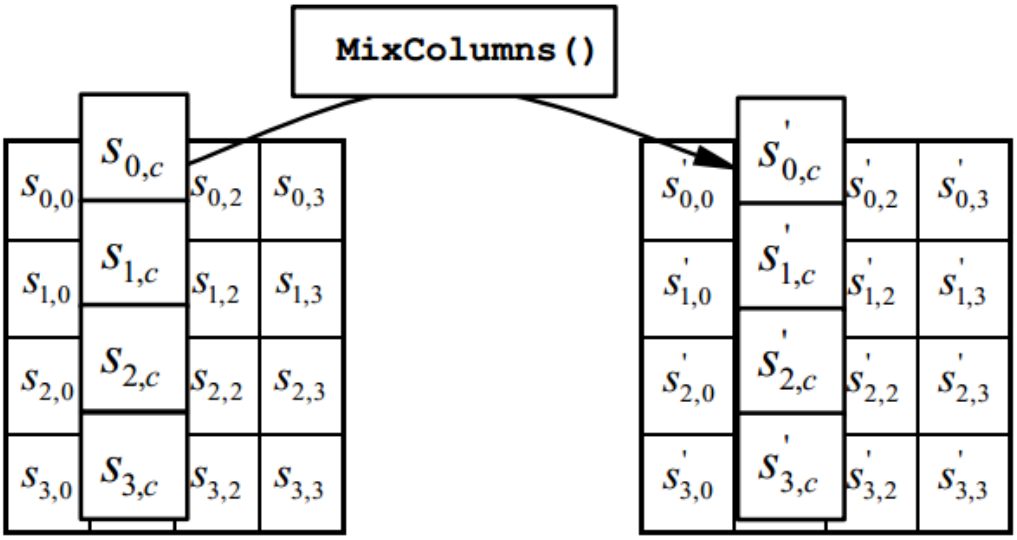
Hasil dari perkalian matriks diatas yaitu setiap *byte* dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap *byte* tersebut pada persamaan 2.6

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

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

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

$$s'_{3c} = (\{03\} \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \cdot s_{3,c}) \quad (2.6)$$



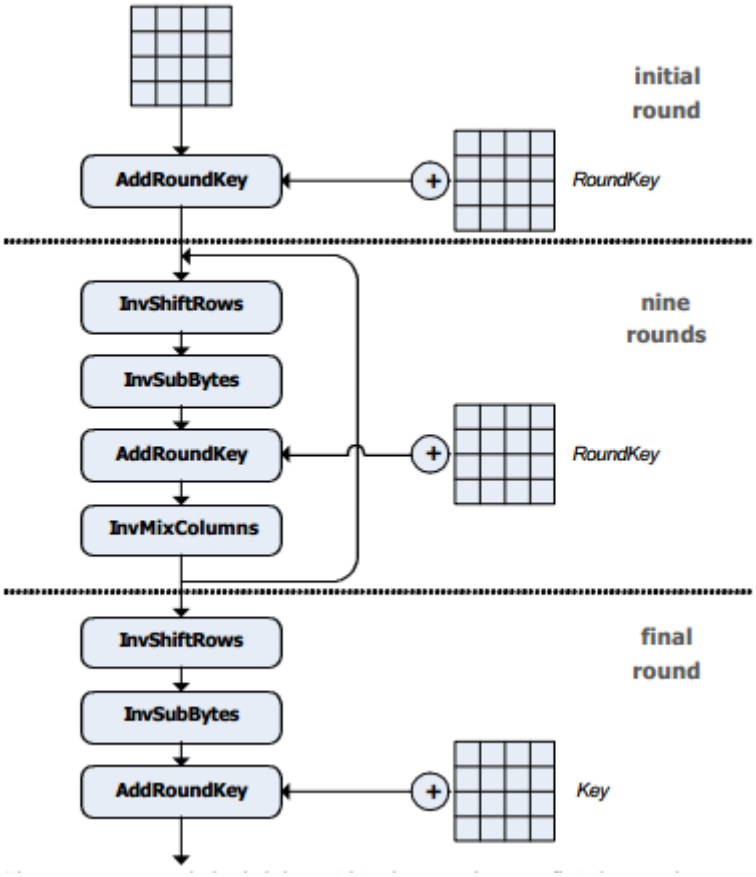
Gambar 2.24 Transformasi *MixColumns* (Gladman, 2001)

2.6.9 Proses Dekripsi

Proses dekripsi dalam algoritma AES (Rijndael) merupakan rangkaian pembalikan dari proses-proses yang dilakukan dalam proses enkripsinya. Gambar 2.25 menunjukkan skema global proses dekripsi AES.

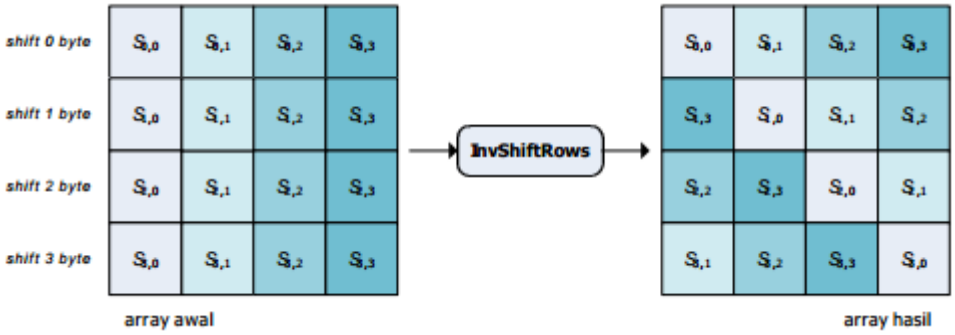
Transformasi *AddRoundKey* tidak mengalami pembalikan (invers) karena proses *AddRoundKey* merupakan operasi XOR. Pembalikan dalam proses ini terletak pada urutan kunci yang digunakan. Jika pada proses enkripsi, urutan kunci internal yang digunakan adalah kunci eksternal (kunci utama), *round key* 1, *round key* 2, *round key* 3, dan seterusnya hingga *round key* 10 $(\mathrm{K}_0, \mathrm{K}_1, \mathrm{K}_2, \mathrm{K}_3, \ldots, \mathrm{K}_{10})$; maka pada proses dekripsi urutan kunci yang digunakan adalah kebalikanya yakni *round key* 10, *round key* 9, *round key* 8, dan seterusnya sampai *round key* 1, kemudian diakhiri dengan kunci eksternal $(\mathrm{K}_{10}, \mathrm{K}_9, \mathrm{K}_8, \mathrm{K}_7, \ldots, \mathrm{K}_0)$ (Kurniawan, 2007).

Transformasi *InvShiftRows* didefinisikan sebagai operasi pergeseran *byte-byte* yang terletak pada bentuk pergeseranya. Pembalikan dalam transformasi ini terletak pada bentuk pergeseranya. Jika pada proses enkripsi, transformasi *ShiftRows* melakukan pergeseran ke kiri, maka pada proses dekripsi transformasi *InvShiftRows* melakukan pergeseran ke kanan (Federal Information Processing Standards Publication, 2001).



Gambar 2.25 Skema global proses dekripsi AES (Kurniawan, 2007)

Banyaknya pergeseran yang dilakukan sama dengan banyaknya pergeseran dalam transformasi *ShifRows*, yakni baris ke-0 digeser sejauh 0 kolom, baris ke-1 digeser sejauh 1 kolom, baris ke-2 digeser sejauh 2 kolom dan pada baris ke-3 digeser sejauh 3 kolom. Gambar 2.26 menunjukkan transformasi *InvShiftRows* (Kurniawan, 2007).



Gambar 2.26 Transformasi *InvShiftRows* (Kurniawan, 2007)

Transformasi *InvSubBytes* didefinisikan sebagai operasi subtitusi yang dilakukan pada masing-masing byte dalam *array state*. Masing-masing *byte* dalam *array state* disubtitusikan dengan nilai baru sesuai dengan nilai yang ditunjukkan dalam matriks subtitusi *S*-Box balikan (*InvS*-Box). Tabel *InvS-Box* ditunjukkan pada bagian lampiran.

Transformasi *InvMixColumns* (Gambar 2.27) didefinisikan sebagai operasi yang dilakukan terhadap *byte-byte* yang terletak pada masing-masing kolom. Operasi yang dilakukan terhadap masing-masing kolom adalah operasi perkalian matriks, sama seperti *MixColumns*. Hal yang membedakan antara transformasi *MixColumns* dan *InvMixColumns* adalah matriks 4x4 yang menjadi matriks pengalinya.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Gambar 2.27 Transformasi *InvMixColumns* (Gladman, 2001)

Hasil dari perkalian matriks tersebut, setiap *byte* dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap *byte* tersebut pada persamaan 2.7

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

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

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

(2.7) (Federal Information Processing Standards Publication, 2001).

2.6.10 Perbedaan AES 128 bit, AES 192 bit dan AES 256 bit

Algoritma enkripsi AES memiliki banyak kesamaan pada setiap jenisnya. Hal tersebut karena AES menggunakan ukuran blok tetap yaitu 128 bit dan menggunakan *round* sekuen yang identik, dimana setiap jenis terdiri dari beberapa pengacakan yang linier maupun non-linier.

Perbedaan dari AES 128 bit, AES 192 bit dan AES 256 bit, yaitu panjang kunci yang berbeda serta jumlah *round* pada setiap jenis algoritma AES. Pada algoritma AES 128 bit menggunakan panjang kunci 128 bit dengan jumlah putaran 10 putaran, pada algoritma AES 192 bit menggunakan panjang kunci 128 bit dengan jumpah putaran 12 putaran, serta pada algoritma AES 256 bit menggunakan panjang kunci 256 bit dengan jumlah putaran 14 putaran. Berikut merupakan perbandingan jumlah *round* dan *key* pada Tabel 2.6

Tabel 2.4 Perbandingan Jumlah Round dan Key

	Jumlah Key (Nk)	Ukuran Blok (Nb)	Jumlah Putaran (Nr)
AES – 128	4	4	10
AES – 192	6	4	12
AES – 256	8	4	14