BAB II

LANDASAN TEORI

Beberapa telaah literatur yang diperlukan dalam melakukan implementasi steganografi LSB dan *library* enkripsi AES-128 ke banyak dokumen PDF adalah sebagai berikut.

2.1 Steganografi

Steganografi adalah seni dan sains berkomunikasi dengan suatu cara dimana cara tersebut menyembunyikan keberadaan dari komunikasi tersebut (Champakamala, dkk., 2014). Steganografi membuat komunikasi menjadi tidak terlihat dengan menyembunyikan informasi yang dimaksud pada informasi lain (Champakamala, dkk., 2014). Terdapat tiga istilah yang umum digunakan untuk membahas steganografi, yaitu *cover*, *secret*, dan *stego* (Champakamala, dkk., 2014). Istilah *cover* mengarah ke deskripsi dari orisinil, pesan, data, audio, video yang murni, bersih, dan sebagainya (Hmood, dkk., 2010). *Secret message, secret file, secret text*, dan sejenisnya adalah pesan yang akan disembunyikan (Brainos II, tanpa tahun). *Stego message, stego file*, dan sejenisnya adalah data yang menjadi hasil akhir dari proses steganografi (Brainos II, tanpa tahun). *Stego file* merupakan *cover file* yang telah disisipkan dengan *secret text* (Brainos II, tanpa tahun).

2.1.1 Least Significant Bit

Salah satu *spatial domain technique* yang mudah digunakan adalah *Least Significant Bit* (LSB) (Champakamala, dkk., 2014). LSB adalah *bit* yang sangat kurang berpengaruh pada nilai *byte* dari sebuah *pixel* pada gambar yang

ditampilkan (Champakamala, dkk., 2014). LSB *based image steganography* mencantumkan informasi tersembunyi pada *least significant bit* dari *pixel* pada *cover image* (Champakamala, dkk., 2014). Konsep ini mengeksploitasi teknologi dimana tingkat presisi dari kebanyakan gambar jauh lebih teliti dibandingkan dengan apa yang bisa diinterpretasikan oleh manusia (Champakamala, dkk., 2014). Sehingga, perubahan kecil terhadap gambar (biasanya warna) tidak akan dapat dibedakan dengan gambar aslinya apabila hanya dilihat secara kasat mata (Champakamala, dkk., 2014).

Kessler (2015) memberikan contoh implementasi LSB *steganography*. Sebuah *array* dari *bytes* memiliki *bit-bit* sebagai berikut (Kessler, 2015).

10010101 00001101 11001001 10010110

00001111 11001011 10011111 00010000

Apabila huruf "G" (dengan ASCII 71 dan memiliki bit 01000111) ingin disembunyikan ke dalamnya, maka *bit-bit* tersebut akan berubah menjadi berikut (Kessler, 2015).

1001010**0** 0000110**1** 1100100**0** 1001011**0**

0000111**0** 1100101**1** 1001111**1** 0001000**1**

Dari *array* di atas, terlihat bahwa *bit* terakhir dari 8 *bytes* tersebut merupakan *bit* yang menyusun huruf "G" (Kessler, 2015).

2.2 Advanced Encryption Standard

Advanced Encryption Standard (AES) adalah algoritma *cryptography* yang telah disetujui oleh *Federal Information Processing Standards* (FIPS) yang dapat digunakan untuk melindungi data elektronik (Federal Information Processing

Standards, 2001). AES dipublikasikan oleh National Institute of Standards and Technology (NIST) pada tahun 2001 (Stallings, 2014). AES bertujuan untuk menggantikan *Data Encryption Standard* (DES) yang menjadi standar untuk berbagai macam penerapan (Stallings, 2014). Algoritma AES adalah sebuah *symmetric block cipher* yang dapat melakukan enkripsi dan dekripsi dari informasi (Federal Information Processing Standards, 2001). Enkripsi mengubah data menjadi bentuk yang tidak dapat dimengerti, disebut *ciphertext*; dekripsi pada *ciphertext* mengubah data kembali menjadi bentuk awalnya, disebut *plaintext* (Federal Information Processing Standards, 2001). Algoritma AES dapat menggunakan *cryptographic keys* berukuran 128-, 192-, dan 256-*bit* untuk melakukan enkripsi dan dekripsi data dalam bentuk *block* berukuran 128-*bit* (Federal Information Processing Standards, 2001).

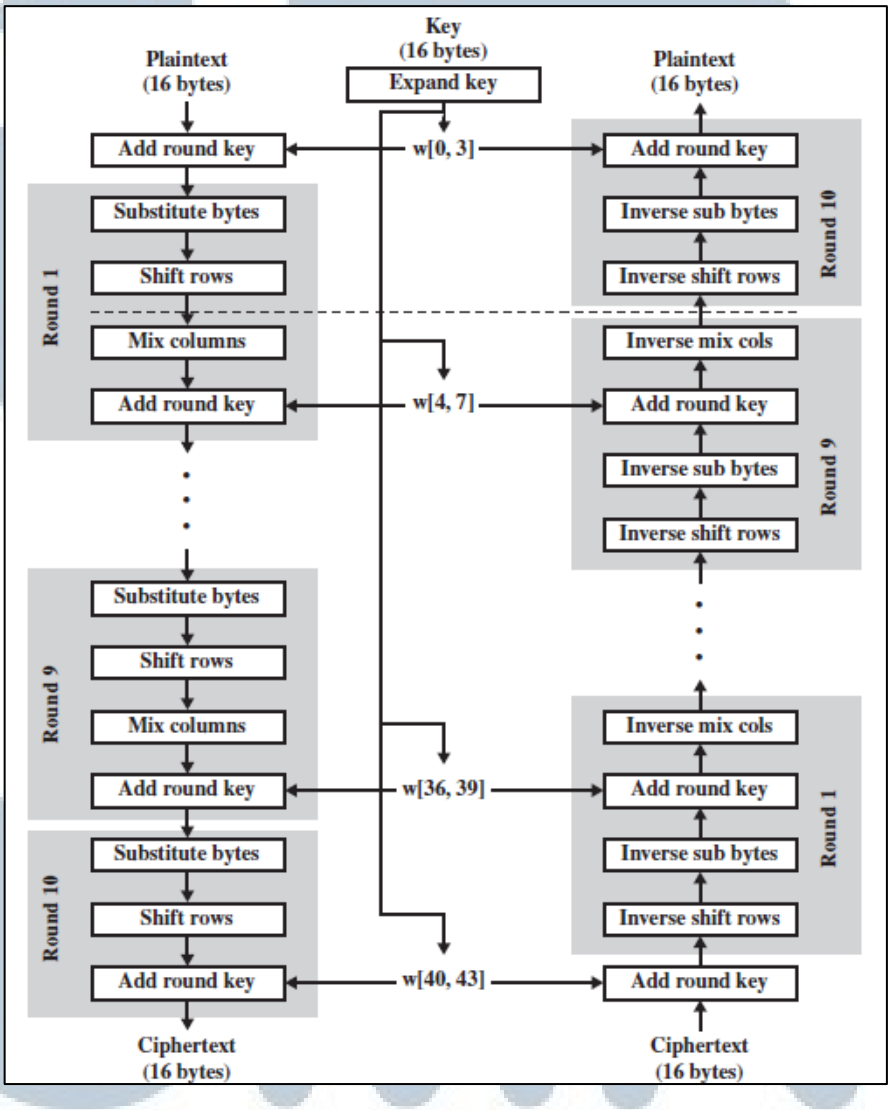
AES terdiri atas empat sub-proses yang berbeda (Stallings, 2014). Salah satu sub-prosesnya, yaitu *ShiftRows*, merupakan permutasi sederhana (Stallings, 2014). Sedangkan tiga sub-proses lainnya, yaitu *Substitute Bytes*, *MixColumns*, dan *AddRoundKey* merupakan proses substitusi (Stallings, 2014).

Jumlah putaran pada AES tidak tetap, tergantung pada ukuran *key* yang digunakan (Swenson, 2008). AES dengan *key* 128-*bit* menggunakan 10 putaran, 192-*bit* 12 putaran, dan 256-*bit* 14 putaran (Swenson, 2008). Setiap putaran menggunakan 128-*bit round key* berbeda, yang berasal dari *key* AES di awal (Swenson, 2008).

Sebelum putaran pertama dimulai, proses *AddRoundKey* dilakukan menggunakan *plaintext* dan *key* awal (Swenson, 2008, Stallings, 2014). Proses ini

melakukan *XOR plaintext* dengan *key* (Swenson, 2008, Stallings, 2014). Hasilnya digunakan sebagai *input* pada putaran pertama (Swenson, 2008, Stallings, 2014).

Menurut Stallings (2014), Setiap putaran pada enkripsi AES memiliki empat sub-proses sebagai berikut.



Gambar 2.1 Enkripsi dan Dekripsi AES
(Stallings, 2014)

a. Substitusi *Byte* (*Substitute Bytes*)

Enam belas *input bytes* yang disusun dalam bentuk matriks 4x4 akan disubstitusikan menggunakan sebuah *fixed table* (*S-box*) yang berasal dari desain

AES (Swenson, 2008, Stallings, 2014). Hasilnya adalah sebuah matriks yang terdiri atas empat baris dan empat kolom (Swenson, 2008, Stallings, 2014). Sebagai contoh, *hexadecimal* 19 akan diubah menjadi D4 sebagai hasil substitusi menggunakan *S-box* (Swenson, 2008, Stallings, 2014).

		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

Gambar 2.2 Substitution Box (S-Box) yang Digunakan pada Enkripsi AES (Stallings, 2014)

b. Pergeseran Baris (*ShiftRows*)

Setiap baris dari matriks akan digeser ke kiri (Stallings, 2014). Semua angka yang melewati batas akan dimasukkan kembali dari sisi kanan setiap baris (Stallings, 2014). Langkah-langkah dari pergeseran baris dilakukan sebagai berikut (Stallings, 2014).

- Baris pertama tidak digeser.
- Baris kedua digeser satu (*byte*) posisi ke kiri.
- Baris ketiga digeser dua posisi ke kiri.
- Baris keempat digeser tiga posisi ke kiri.

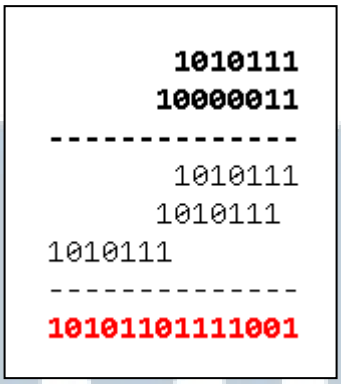
Hasilnya adalah sebuah matriks baru berisikan 16 *bytes* yang sama, tetapi tergeserkan posisinya (Stallings, 2014).

c. Mengubah Isi Kolom (*MixColumns*)

Tahap ini mengoperasikan matriks kolom demi kolom (Swenson, 2008). Pada dasarnya, operasi yang dilakukan adalah mengalikan dua 8-*bit* angka, setelah itu mengambil sisa dari modulo 283 menggunakan *binary XOR long division* (Swenson, 2008). Seperti pembagian (*division)* biasa, namun menggunakan *binary XOR* daripada pengurangan (Swenson, 2008). Operasi ini dilambangkan menggunakan tanda • (Swenson, 2008).

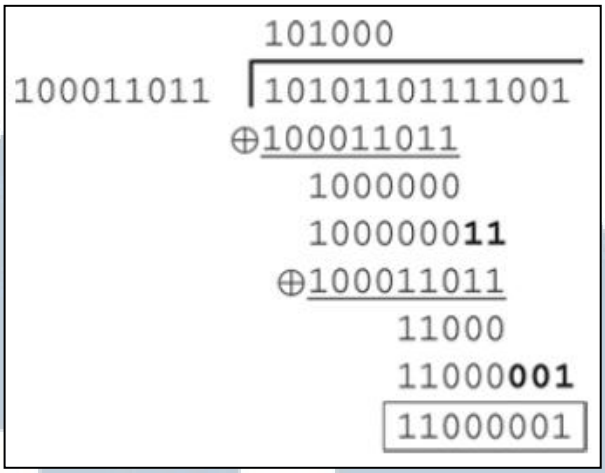
Sebagai contoh, 87 • 131 (atau, 01010111 • 10000011) akan dihitung dengan langkah sebagai berikut (Swenson, 2008).

1. Kalikan kedua angka tersebut seperti biasa (perkalian dalam *bit*). Pertambahan dilakukan menggunakan aturan *XOR*.



Gambar 2.3 Perkalian 87 dengan 131 pada operasi AES

2. Lakukan *binary XOR long division* dengan 283 (100011011) untuk mendapatkan sisa dari hasil.



Gambar 2.4 Binary XOR Long Division
(Swenson, 2008)

Jadi, 01010111 • 10000011 menghasilkan 11000001.

Menurut Swenson (2008), *matrix multiplication* dilakukan pada setiap kolom matriks yang menjadi *input*.

$$\begin{bmatrix} b_{0,c} \\ b_{1,c} \\ b_{2,c} \\ b_{3,c} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \otimes \begin{bmatrix} a_{0,c} \\ a_{1,c} \\ a_{2,c} \\ a_{3,c} \end{bmatrix}$$

Gambar 2.5 Matrix Multiplication AES dalam Decimal
(Swenson, 2008)

$a_{i,c}$ adalah kolom yang menjadi *input*, $b_{i,c}$ adalah hasil dari perkalian matriks, dan operator ⊗ berarti perkalian matriks dilakukan menggunakan • operator dan *XOR* daripada menggunakan pertambahan pada umumnya (Swenson, 2008). Kolom b akan menggantikan kolom a (Swenson, 2008). Perhitungan di atas dapat dituliskan lebih jelas pada Gambar 2.6 (Swenson, 2008).

$$
\begin{aligned}
b_{0,c} &= (2 \bullet a_{0,c}) \oplus (3 \bullet a_{1,c}) \oplus a_{2,c} \oplus a_{3,c} \\
b_{1,c} &= a_{0,c} \oplus (2 \bullet a_{1,c}) \oplus (3 \bullet a_{2,c}) \oplus a_{3,c} \\
b_{2,c} &= a_{0,c} \oplus a_{1,c} \oplus (2 \bullet a_{2,c}) \oplus (3 \bullet a_{3,c}) \\
b_{3,c} &= (3 \bullet a_{0,c}) \oplus a_{1,c} \oplus a_{2,c} \oplus (2 \bullet a_{3,c})
\end{aligned}
$$

Gambar 2.6 Rincian Matrix Multiplication AES (Swenson, 2008)

Proses *MixColumns* tidak dilakukan pada putaran terakhir (Swenson, 2008, Stallings, 2014). Pada AES-128, *MixColumns* tidak dilakukan pada putaran ke-10 (Swenson, 2008, Stallings, 2014).

d. Tambah *round key* (*AddRoundKey*)

16 *bytes* dari matriks akan dianggap sebagai 128-*bit* dan di-*XOR* dengan 128-*bit* dari *round key* (Stallings, 2014). Jika ini adalah putaran terakhir, maka hasilnya merupakan *ciphertext* (Stallings, 2014). Jika tidak, 128-*bit* yang menjadi hasil akan dianggap sebagai 16 *bytes input* dan putaran baru akan dimulai (Stallings, 2014).

Proses dekripsi dari *ciphertext* AES menyerupai proses enkripsi dalam urutan terbalik (Swenson, 2008). Terdapat proses *AddRoundKey* sebelum putaran pertama dari dekripsi *ciphertext* AES dimulai (Swenson, 2008). *AddRoundKey* dilakukan dengan menggunakan *key* dengan urutan yang terbalik (*key* untuk putaran terakhir pada proses enkripsi digunakan pertama kali, *key* untuk putaran kedua terakhir pada proses enkripsi digunakan kedua kali, dan seterusnya) (Swenson, 2008). *Ciphertext* akan di-*XOR* dengan *key*, menghasilkan *input* pada proses dekripsi selanjutnya (Swenson, 2008).

Menurut Stallings (2014), setiap putaran dari dekripsi AES memiliki empat sub-proses sebagai berikut.

a. InvShiftRows

Setiap baris dari matriks akan digeser ke kanan (kebalikkan dari *ShiftRows*) (Stallings, 2014). Semua angka yang melewati batas akan dimasukkan kembali dari sisi kiri setiap baris (Stallings, 2014). Langkah-langkah dari pergeseran baris dilakukan sebagai berikut (Stallings, 2014).

- Baris pertama tidak digeser.
- Baris kedua digeser satu (*byte*) posisi ke kanan.
- Baris ketiga digeser dua posisi ke kanan.
- Baris keempat digeser tiga posisi ke kanan.

Hasilnya adalah sebuah matriks baru berisikan 16 *bytes* yang sama, tetapi tergeserkan posisinya (Stallings, 2014).

b. InvSubBytes

16 *input bytes* yang disusun dalam bentuk matriks 4x4 akan disubstitusikan kembali menggunakan *inverse S-box* (Stallings, 2014). Menggunakan contoh yang sama dengan proses enkripsi sebelumnya, D4 akan dikembalikan menjadi 19 menggunakan bantuan *inverse S-box* (Stallings, 2014).

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

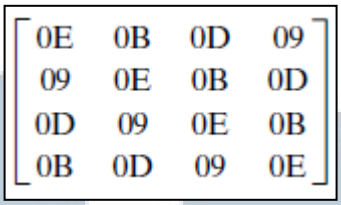
Gambar 2.7 Inverse S-box
(Stallings, 2014)

c. AddRoundKey

Ciphertext akan di-*XOR* dengan *round key* (Stallings, 2014). Jika *AddRoundKey* dilakukan bukan pada putaran terakhir, hasilnya akan menjadi input pada proses selanjutnya (Stallings, 2014). Jika dilakukan pada putaran terakhir, hasilnya adalah *plaintext* (Stallings, 2014).

d. InvMixColumns

InvMixColumns dihitung menggunakan perhitungan yang sama dengan perhitungan pada saat melakukan proses enkripsi (Stallings, 2014). Namun, matriks yang digunakan pada saat melakukan *matrix multiplication* untuk setiap kolom *input* diganti menjadi matriks pada Gambar 2.8 berikut (Stallings, 2014).



Gambar 2.8 Matriks yang Digunakan pada InvMixColumns dalam Hexadecimal (Stallings, 2014)

Proses *InvMixColumns* tidak dilakukan pada putaran terakhir dari proses dekripsi AES (Stallings, 2014). Pada AES-128, *InvMixColumns* tidak dilakukan pada putaran ke-10 dari proses dekripsi (Stallings, 2014).

2.2.1 AES Key Expansion

AES *key expansion algorithm* menerima *input* berupa *four-word* (16 *bytes*) *key* dan menghasilkan 44 *words* (176 *bytes*) (Stallings, 2014). 176 *bytes* yang dihasilkan cukup untuk digunakan pada setiap proses *AddRoundKey* pada AES-128 (11 kali dengan 16 *bytes* setiap proses) (Stallings, 2014).

Key yang diberikan di awal menjadi empat *word* pertama dari *expanded key* (Stallings, 2014). Setelah itu, *word* selanjutnya akan diisi secara bertahap (Stallings, 2014). Setiap *word* w[i] akan bergantung pada *word* sebelumnya, w[i-1], dan *word* keempat sebelumnya, w[i-4] (Stallings, 2014). Jika posisi *word* bukan merupakan kelipatan dari empat, maka *word* yang baru dihitung dengan melakukan *XOR* w[i-1] dengan w[i-4] (Stallings, 2014). Jika posisi *word* adalah kelipatan dari empat, perhitungan yang lebih kompleks digunakan (Stallings, 2014).

Menurut Stallings (2014), *word* dengan posisi kelipatan dari empat dihitung menggunakan cara sebagai berikut.

1. *one-byte circular left shift* dilakukan pada w[i-1]. Sebagai contoh, sebuah *input word* [B_0, B_1, B_2, B_3] akan menjadi [B_1, B_2, B_3, B_0].
2. Setiap *byte* disubstitusikan dengan menggunakan *S-box* seperti pada saat menyubtitusikan pada proses *SubBytes*.
3. Hasil dari substitusi akan di-*XOR* dengan Rcon[j], dimana j adalah putaran dimana proses tersebut sedang berjalan.
4. Hasilnya akan di-*XOR* kembali dengan w[i-4].

Pseudocode dari AES *key expansion algorithm* ditunjukkan pada Gambar 2.9 berikut.

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++)   w[i] = (key[4*i], key[4*i+1],
                                      key[4*i+2],
                                      key[4*i+3]);

     for (i = 4; i < 44; i++)
     {
      temp = w[i - 1];
      if (i mod 4 = 0)   temp = SubWord (RotWord (temp))
                               ⊕ Rcon[i/4];
      w[i] = w[i-4] ⊕ temp
     }
}
```

Gambar 2.9 Key Expansion Algorithm
(Stallings, 2014)

Rcon, atau disebut juga *round constant*, adalah sebuah *word* dimana tiga *rightmost bytes*-nya selalu bernilai nol (Stallings, 2014). Oleh karena itu, hasil dari *XOR* sebuah *word* dengan Rcon adalah hanya dengan melakukan *XOR* pada *leftmost byte* dari *word* tersebut (Stallings, 2014). *Round constant* berbeda untuk setiap putaran dan didefinisikan sebagai Rcon[j] = (RC[j], 0, 0, 0), dengan RC[1]

= 1, RC[j] = 2 • RC[j-1]. Rcon untuk j=1 sampai j=10 ditunjukkan pada Gambar 2.10 (Stallings, 2014).

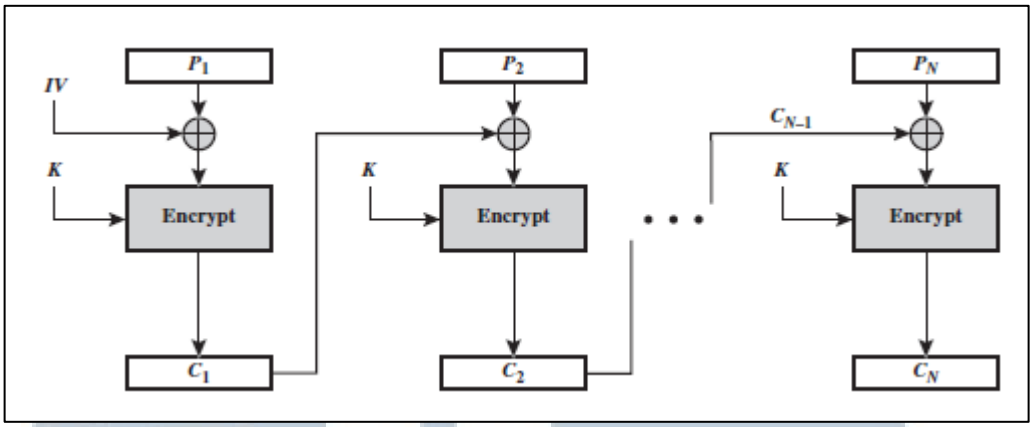
j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

Gambar 2.10 Round Constant untuk Putaran 1 sampai 10 (Stallings, 2014)

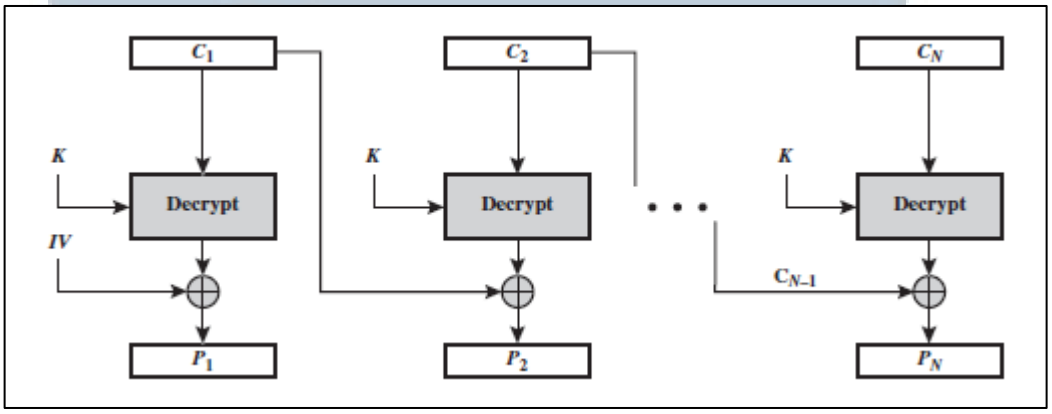
2.2.2 Cipher Block Chaining

Cipher-block Chaining (CBC) adalah mode enkripsi yang menambahkan *XOR* setiap *plaintext block* dengan *ciphertext block* sebelumnya (Stallings, 2014). Hasilnya baru akan dienkripsi seperti biasa (Stallings, 2014). Setiap *ciphertext block* selanjutnya bergantung pada *block* sebelumnya (Stallings, 2014). *Plaintext block* pertama di-*XOR* dengan *random initialization vector* (yang biasanya disebut dengan *IV*) (Stallings, 2014). Mode CBC memerlukan *padding* untuk *plaintext block* terakhir agar menjadi *block* penuh jika *block* tersebut hanya sebagian atau *partial* (Stallings, 2014).

Enkripsi dalam mode CBC tidak dapat dilakukan secara parallel (Li, dkk., 2012). Hal ini dikarenakan *ciphertext block* selanjutnya bergantung pada hasil *ciphertext block* sebelumnya (Li, dkk., 2012). Sedangkan pada dekripsi mode CBC, penerima mengetahui semua *ciphertext block* setelah menerima pesan yang terenkripsi (Li, dkk., 2012). Oleh karena itu, penerima dapat melakukan dekripsi pesan secara parallel (Li, dkk., 2012).



Gambar 2.11 Enkripsi pada Mode CBC
(Stallings, 2014)



Gambar 2.12 Dekripsi pada Mode CBC
(Stallings, 2014)

Menurut Stallings (2014), mode CBC dibuat dengan tujuan untuk menghilangkan kekurangan yang terdapat pada mode *Electronic Code Book* (ECB), yaitu dua *plaintext* yang identik menghasilkan dua *ciphertext* yang identik. Syben (2011) menyebutkan beberapa kelebihan dan kekurangan dari mode CBC. Kelebihan dari mode CBC adalah sebagai berikut.

- Dekripsi dapat dilakukan secara bersamaan untuk setiap *block*.
- *Initialization Vector* yang berbeda akan menghasilkan *ciphertext* yang berbeda.

- Pola dari *plaintext* tersamarkan.
- Sedangkan kekurangan dari mode CBC adalah sebagai berikut.
- Enkripsi dari *plaintext* harus dilakukan secara sekuensial.
- *Bit error* di satu *block* memengaruhi dua *blocks*.

2.2.3 javax.crypto

javax.crypto adalah sebuah *package* pada *library* Java yang menyediakan *classes* dan *interfaces* untuk operasi *cryptography* (Oracle, tanpa tahun). Operasi *cryptography* tersebut antara lain enkripsi, *key generation*, dan *Message Authentication Code* (MAC) (Oracle, tanpa tahun). Enkripsi yang didukung antara lain *symmetric*, *asymmetric*, *block*, dan *stream ciphers* (Oracle, tanpa tahun). Beberapa *class* dan *interface* yang digunakan untuk melakukan enkripsi AES-128 menggunakan *package* ini adalah sebagai berikut.

- *javax.crypto.KeyGenerator*

Class ini menyediakan fungsionalitas untuk membuat *secret* (*symmetric*) *key* (Oracle, tanpa tahun). Obyek dari *class KeyGenerator* dibuat menggunakan *method getInstance* dari *class* ini (Oracle, tanpa tahun). Pembuatan *key* dilakukan dengan memanggil *method init* dan menyertakan ukuran *key* yang digunakan pada parameter (Oracle, tanpa tahun).

- *javax.crypto.SecretKey*

Interface ini adalah sebuah *secret* (*symmetric*) *key* (Oracle, tanpa tahun). *Interface* ini tidak berisikan *method* atau *constant* apapun (Oracle, tanpa tahun).

- *javax.crypto.Cipher*

Class Cipher berfungsi untuk melakukan enkripsi dan dekripsi (Oracle, tanpa tahun). *Method getInstance* dipanggil untuk membuat sebuah obyek *Cipher* dengan menyertakan transformasi yang diinginkan (Oracle, tanpa tahun). Transformasi yang dimaksud adalah sebuah *string* yang mendeskripsikan operasi yang akan dilakukan terhadap *input* untuk menghasilkan *output* (Oracle, tanpa tahun). *String* transformasi terbentuk dari "algoritma/mode/padding" atau "algoritma" (Oracle, tanpa tahun). Salah satu contoh string transformasi adalah "DES/CBC/PKCS5Padding" (Oracle, tanpa tahun).

- *javax.crypto.spec.IvParameterSpec*

Class ini digunakan untuk membuat *Initialization Vector* (IV) (Oracle, tanpa tahun). Salah satu contoh mode enkripsi yang menggunakan IV yaitu mode CBC pada DES (Oracle, tanpa tahun). Pembuatan obyek dari *class IvParameterSpec* dapat menggunakan *constructor* dengan menyertakan *array of bytes* pada parameternya sebagai IV tersebut (Oracle, tanpa tahun).

- *javax.crypto.spec.SecretKeySpec*

Class ini digunakan untuk membuat *secret key* (Oracle, tanpa tahun). Obyek dari *class* ini dapat dibuat menggunakan *constructor* dengan menyertakan *array of bytes* yang akan menjadi *secret key* dan algoritma yang digunakan pada parameternya (Oracle, tanpa tahun).

2.3 PDF

Portable Document Format (PDF) adalah sebuah *platform independent file format* untuk merepresentasikan dokumen (Alizadeh, dkk., 2012). Teks dan

gambar yang ada di dalam PDF ditampilkan dengan cara yang sama di semua *platform* (Alizadeh, dkk., 2012). Pada awalnya, PDF adalah *format* dokumen yang dimiliki oleh Adobe (Alizadeh, dkk., 2012). Namun, pada tanggal 1 Juli 2008, *International Organization for Standardization* (ISO) memublikasikan PDF sebagai *open standard* di bawah ISO 32000-1:2008 (Alizadeh, dkk., 2012). Standar ini dapat dilihat pada situs Adobe (Alizadeh, dkk., 2012).

Dokumen PDF terdiri atas koleksi beberapa obyek yang menentukan keluaran dan fungsionalitas dari dokumen (Alizadeh, dkk., 2012). Salah satu obyek yang paling sering digunakan adalah *stream object* (Alizadeh, dkk., 2012). Sebagai contoh, teks terkandung di dalam *stream object* (Alizadeh, dkk., 2012). Obyek-obyek lainnya seperti *numbers, strings, arrays,* dan *dictionaries* (Alizadeh, dkk., 2012).

File PDF biasanya terkompresi untuk menghemat memori (Alizadeh, dkk., 2012). Keseluruhan *source code* dari sebuah *file* PDF dapat dilihat dengan melakukan dekompresi terhadap *file* PDF tersebut (Alizadeh, dkk., 2012). Dekompresi *file* PDF dapat dilakukan dengan memanfaatkan program seperti pdftk atau QPDF (Alizadeh, dkk., 2012). Proses dekompresi dari *file* PDF tidak memakan waktu yang lama (Alizadeh, dkk., 2012). Bahkan untuk mendekompresi 1 GB *file* PDF hanya dibutuhkan waktu sekitar satu menit (Alizadeh, dkk., 2012). Oleh karena itu, kompresi terhadap *file* PDF tidak menambah keamanan karena *source code file* PDF tersebut dapat dibaca kembali dengan cepat (Alizadeh, dkk., 2012).

File PDF memiliki berbagai operator yang berfungsi untuk mengatur bagaimana suatu konten ditampilkan (Alizadeh, dkk., 2012). Tampilan teks pada

file PDF dapat diatur dengan memanfaatkan *Tc*, *Tw*, dan *Tj Operator* (Alizadeh, dkk., 2012). *Tc Operator* mendefinisikan spasi antar karakter, *Tw Operator* mendefinisikan spasi antar kata, dan *Tj Operator* dapat mengendalikan posisi dari setiap individu karakter yang ada pada sebuah *text string* (Alizadeh, dkk., 2012).

2.3.1 Tj Operator

Tj Operator digunakan untuk menampilkan *text strings* di dalam *file* PDF (Alizadeh, dkk., 2012). *Tj Operator* terdiri dari sebuah *array of strings and numbers* yang merepresentasikan karakter-karakter dan nilai spasi yang digunakan di antara karakter tersebut (Alizadeh, dkk., 2012). Untuk setiap nilai *Tj*, posisi teks pada saat itu akan diubah dengan mengurangi nilai dari posisi saat itu (Alizadeh, dkk., 2012). Jika nilai *Tj* berikutnya adalah negatif berarti karakter selanjutnya dipindahkan sedikit ke kanan (spasi bertambah) (Alizadeh, dkk., 2012). Sedangkan jika nilai *Tj* berikutnya adalah positif berarti karakter selanjutnya dipindahkan lebih dekat dengan karakter sebelumnya (spasi berkurang) (Alizadeh, dkk., 2012). *Tj Operator* banyak digunakan untuk mendefinisikan jarak spasi yang berubah-ubah antar karakter pada teks *justify* (Alizadeh, dkk., 2012).

`[(A)35(w)35(a)200(y)] TJ`	Away
`[(A)35(w)35(a)-200(y)] TJ`	Awa y

Gambar 2.13 Contoh dari Tj Operator

Alizadeh, dkk. (2012) membuat implementasi steganografi LSB pada dokumen PDF menggunakan Python 2, karena *syntax* yang mudah dan biasanya memerlukan *code* yang lebih singkat daripada *scripting languages* lainnya. Selain

itu, Python 2 memiliki *re module* untuk dapat memanfaatkan *regular expression* (Alizadeh, dkk., 2012). Blok *Tj* dicari dengan menggunakan regular expression "\[(.*)\][ ]?TJ" (Alizadeh, dkk., 2012). Setelah Blok *Tj* didapat, setiap *Tj value* dari blok tersebut dicari dengan menggunakan *regular expression* "[>)](-?[0-9]+)[<(]" (Alizadeh, dkk., 2012). *Secret message* disembunyikan di empat *bits* terakhir pada *Tj value* yang memiliki rentang -447 sampai -337 dan -320 sampai -257. *Tj value* ini dipilih untuk menghindari perubahan pada *Tj value* yang frekuensi kemunculannya sangat tinggi atau sangat rendah (Alizadeh, dkk., 2012).

2.4 Peak Signal-to-Noise Ratio

Peak Signal-to-Noise Ratio (PSNR) adalah cara untuk mengukur kualitas dari dua gambar (Veldhuizen, 1998). Semakin tinggi nilai PSNR, semakin serupa gambar satu dengan yang lainnya (Varnan, 2011). Gambar 2.14 menunjukkan cara untuk menghitung nilai PSNR.

$$PSNR = 10\log_{10}\frac{(2^n-1)^2}{\sqrt{MSE}}$$

Gambar 2.14 Peak Signal-to-Noise Ratio
(Varnan, 2011)

Sedangkan *Mean-squared Error* (MSE) dihitung seperti pada Gambar 2.15 berikut. x(i, j) adalah gambar orisinil, y(i, j) adalah gambar lainnya, i dan j adalah posisi pixel dari gambar berukuran M × N (Varnan, 2011).

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x(i,j)-y(i,j))^2$$

Gambar 2.15 Mean-squared Error
(Varnan, 2011)