BAB 2

Tinjauan Pustaka

2.1 Tinjauan Perusahaan

Pada tinjauan perusahaan yang didapat dari Profil Perusahaan (*Company Profile*) PT. Prospera Consulting Engineers berikut ini akan membahas mengenai sejarah berdirinya PT. Prospera Consulting Engineers, biodata perusahaan, bidang aktivitas perusahaan, pengalaman perusahaan, dan struktur organisasi.

2.1.1 Sejarah PT. Prospera Consulting Engineers

PT. Prospera Consulting Engineers, merupakan salah satu perusahaan konsultan yang telah cukup dikenal di Indonesia. Didirikan pada tahun 1980 sebagai jawaban atas tuntutan pembangunan Nasional yang banyak membutuhkan jasa konsultasi perencanaan teknik.

PT. Prospera Consulting Engineers dalam waktu yang relatif saingkat berkembang cukup pesat. Beberapa tahun setelah pendirian, seiring dengan bertambahnya permintaan/kebutuhan akan jasa konsultasi, layanan yang diberikan kemudian berkembang menjadi lebih luas lagi, mulai dari perencanaan umum, jasa survei, studi kelayakan, perencanaan teknis, umum, transportasi, pariwisata, pos dan telekomunikasi, pertanian, perindustrian, pertambangan dan energi, serta bidang-bidang lain.

PT. Prospera Consulting Engineers, berkat dukungan tenaga ahli yang berpengalaman dalam berbagai bidang dan dukungan peralatan yang modern, dalam bidang-bidang tersebut di atas telah memperlihatkan prestasi yang luas dan handal, seperti pada bangunan gedung dan pabrik, jalan dan jembatan, pelabuhan

laut, bandar udara, jaringan irigasi, bangunan pengolahan dan sistem jaringan pendistribusian air minum, perbaikan kampung dan lingkungan, perencanaan kota dan wilayah, perencanaan pedesaan, perencanaan kawasan industri, dan analisis dampak lingkungan.

PT. Prospera Consulting Engineers telah menjadi mitra kerja yang terpercaya, baik bagi instansi pemerintah, swasta, maupun perseorangan. Begitu pula dengan badan keuangan internasional, seperti telah terjalin hubungan yang cukup baik sehubungan dengan ikut sertanya PT. Prospera Consulting Engineers bersama konsultan internasional dalam proyek-proyek internasional.

2.1.2 Biodata Perusahaan

Biodata perusahaan berikut ini adalah paparan nama dan alamat perusahaan, tahun berdirinya perusahaan, serta legalitas perusahaan berdasarkan hukum yang berlaku di Indonesia yang dapat dilihat pada Tabel 2.1.

Tabel 2.1 Biodata PT. Prospera Consulting Engineers

Nama Perusahaan	:	PT. Prospera Consulting Engineers
Alamat	:	Jl. Mampang Prapatan XVIII No. 11 A, Warung Buncit-Jakarta Selatan 12120
Studio	:	Jl. Dempo III No. 4, Kebayoran Baru-Jakarta Selatan
Akte Notaris Pendirian	:	Imas Fatimah, S.H.
Nomor Akta	:	68
Tanggal	:	30 September 1980
Akte Notaris Perubahan	:	Ny. Budiono Widjaja, S.H.
Nomor	:	38
Tanggal	:	26 Februari 2002
NPWP	:	01.364.330.9-061.000
Akte Notaris Salinan	:	Ny. Budiono Widjaja, S.H.
Nomor	:	29
Tanggal	:	28 Juni 2002
SIUJK	:	0904.1.80.88.00267
IUJK Pengawasan	:	1004675317100230

IUJK Perencanaan	:	1004675317100237
TDP	:	09.03.1.74.09139
SIUP	:	27780146-P/09-05/PB/II/2000
Sertifikasi Badan Usaha	:	B00803171-4-DKI0270.

2.1.3 Bidang Aktivitas Perusahaan

PT. Prospera Consulting Engineers sebagai perusahaan konsultan memiliki berbagai bidang aktivitas yang diantaranya adalah sebagai berikut:

a. Pembangunan Lingkungan
    1. Sistem dan Pembagian Air Minum
    2. Drainage dan Air Limbah
    3. Sistem Pembuangan Sampah
    4. Analisis Dampak Lingkungan
b. Perencanaan Kota dan Daerah/Wilayah/Kawasan
    1. Perencanaan Pembangunan Daerah Pedesaan dan Transmigrasi
    2. Perencanaan Fisik Tata Ruang Daerah / Perkebunan (PIR)
    3. Perencanaan Kota Kecamatan, Kota Kab. / Kotamadya dan Metropolitan
    4. Perencanaan Daerah Kabupaten dan Propinsi
    5. Perencanaan Kota
    6. Perencanaan Kota Baru
    7. Perencanaan Pengembangan Lingkungan dan Kebutuhan Daerah Setempat
    8. Perencanaan Ruang Terbuka Hijau
    9. Perencanaan dan Pengembangan Daerah Rekreasi / Pariwisata
    10. Perencanaan Transportasi Kota
    11. Perencanaan dan Pengembangan Kawasan Industri

12. Desain Kawasan Perkotaan (*Urban Design*)
13. Pengawasan Pembangunan Kawasan Perkotaan (*Urban Development Control*)

c. Pembangunan Sumber-sumber Air
    1. Irigasi
    2. Sumber-sumber Air
    3. Pengendalian Banjir
    4. Perencanaan Daerah Aliran Sungai

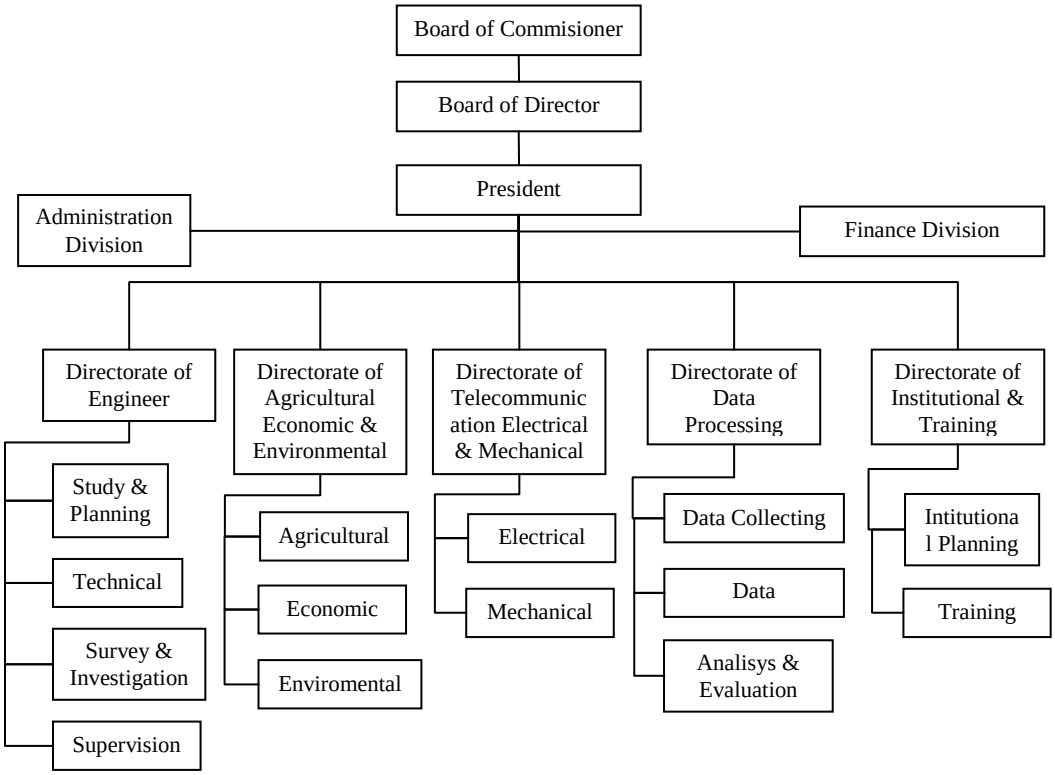
d. Bangunan dan Arsitektur
    1. Perumahan, Kantor, Bangunan Komersial, Bangunan Umum, dll.
    2. Bangunan Sejarah
    3. Interior
    4. Pertamanan

e. Transportasi
    1. Jalan Raya dan Manajemen lalu Lintas
    2. Jembatan dan Terowongan
    3. Bandar Udara dan Terminal
    4. Pelabuhan laut

f. Bidang-Bidang Lain
    1. Penyelidikan Tanah
    2. Inventarisasi Lahan
    3. Pemetaan
    4. Topografi dan hidografi.

2.1.4 Struktur Organisasi

Berikut ini adalah struktur organisasi di PT. Prospera Consulting Engineers seperti pada Gambar 2.1. di bawah ini.



Gambar 2.1 Struktur Organisasi PT. Prospera Consulting Engineers

2.2 Landasan Teori

Pada bagian ini menjelaskan tentang teori-teori yang menunjang dalam pemecahan masalah yang dianggap relevan dengan pokok bahasan.

2.2.1 Pengertian Berkas Digital

Berkas digital (*file*) adalah identitas dari data yang disimpan di dalam sistem berkas yang dapat diakses dan diatur oleh pengguna. Sebuah berkas memiliki nama yang unik dalam direktori di mana ia berada. Alamat direktori dimana suatu berkas ditempatkan diistilahkan dengan *path*.

2.2.1.1 Jenis Berkas Digital

Berkas digital (*file*) dapat digolongkan menurut jenisnya [3], diantaranya adalah:

1. *File* Induk

   Suatu *file* yang diperlukan untuk memperlancar operasi sistem dan diperbaharui secara teratur.

2. *File* Transaksi

   Digunakan untuk memperbaharui *file* induk dengan informasi yang baru

3. *File* Penunjang

   Merupakan kutipan dari file untuk menjaga kemungkinan adanya file asli yang rusak.

4. *File* Riwayat Hidup

   Informasi yang dikumpulkan selama periode waktu tertentu dan biasanya digunakan untuk menyusun laporan statistik.

5. *File* Data Transaksi

   Merupakan sebuah rekaman pada pita atau piringan untuk setiap transaksi yang melengkapi *file* induk.

6. *File* Kesalahan

   Rekaman tentang kesalahan yang disimpan pada file untuk pembentulan di kemudian hari.

7. *File* Laporan

   Merupakan turunan laporan tercetak yang ditahan pada piringan atau pita menunggu *printer* siap mencetak.

8. *File* Sementara

   File yang disiapkan untuk memproses peralihan.

9. *File* Pustaka (*Library*)

   File yang digunakan untuk menyimpan program aplikasi, program utiliti, dan program lain.

10. *File* Kerja (*Work*)

    File ini berisi *record-record* yang disusun sedemikian rupa sehingga dapat dibuat sebuah program dan dipakai oleh program lain sebagai *input*.

11. *File* Program

    File ini berisi perintah-perintah untuk memproses data. Perintah dapat ditulis dalam bahasa pemrograman, bahasa rakitan atau bahasa mesin.

2.2.1.2 Ekstensi Berkas Digital

Setiap berkas digital dalam media penyimpanan memiliki tanda pengenal atau ciri-ciri yang menyatakan jenis berkas tersebut. Umumnya pengenal tipe berkas tertera pada nama berkas tersebut, yaitu tiga huruf paling kanan setelah titik. Fungsinya adalah untuk mengetahui atau membedakan jenis berkas. Berikut ini adalah contoh macam-macam ekstensi berkas digital ditunjukkan pada Tabel 2-2:

Tabel 2.2 Ekstensi Berkas Digital

Ekstensi	Jenis	Aplikasi yang digunakan
001, 002, ...	Bagian-bagian dari suatu *file* tunggal	FFSJ, HJSplit, JJSplit, split (Unix)
3GP	*File video* untuk perangkat *mobile phone*	*Media Player Classic*
ADI	*File* grafis	AutoCAD
BAS	*Source code* modul bahasa pemrograman BASIC	*Microsoft Visual Basic*, *Microsoft Visual Studio*, dan semua aplikasi editor teks
DOC, DOCX	Dokumen MS *Word*	*Microsoft Word* dan semua aplikasi pengolah kata
HTM, HTML	Dokumen *internet*	*Internet Explorer*, *Mozilla Firefox*, dan semua aplikasi penjelajah
JAVA	*Source code* bahasa pemrograman *Java*	NetBeans IDE dan semua aplikasi editor teks
JPEG, JPG	*File* gambar	*Paint, Microsoft Office Picture Manager*, dan semua aplikasi pengolah gambar

2.2.2 Pengertian Kriptografi

Kriptografi (*cryptography*) berasal dari bahasa Yunani yaitu *cryptos* artinya rahasia (*secret*) dan *graphein* artinya tulisan (*writing*). Jadi kriptografi berarti tulisan rahasia (*secret writing*).

Kriptografi, secara umum adalah ilmu dan seni untuk menjaga kerahasiaan berita [9]. Selain pengertian tersebut, kriptografi merupakan ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, keabsahan data, integritas data, serta autentikasi data [4]. Ada empat tujuan mendasar dari ilmu kriptografi ini yang juga merupakan aspek keamanan informasi, yaitu:

1. Kerahasiaan, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka/mengupas informasi yang telah disandi.
2. Integritas data, adalah berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam data yang sebenarnya.
3. Autentikasi, adalah berhubungan dengan identifikasi/pengenalan, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.

4. Non-repudiasi atau nirpenyangkalan, adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman/terciptanya suatu informasi oleh yang mengirimkan/membuat.

2.2.2.1 Algoritma Sandi

Algoritma sandi adalah algoritma yang berfungsi untuk melakukan tujuan kriptografis. Algoritma tersebut harus memiliki kekuatan untuk melakukan [10]:

1. Konfusi/pembingungan (*confusion*), dari teks terang sehingga sulit untuk direkonstruksikan secara langsung tanpa menggunakan algoritma dekripsinya.
2. Difusi/peleburan (*difusion*), dari teks terang sehingga karakteristik dari teks terang tersebut hilang.

sehingga dapat digunakan untuk mengamankan informasi. Pada implementasinya sebuah algoritma sandi harus memperhatikan kualitas layanan/*Quality of Service* (QoS) dari keseluruhan sistem dimana dia diimplementasikan. Algoritma sandi yang handal adalah algoritma sandi yang kekuatannya terletak pada kunci, bukan pada kerahasiaan algoritma itu sendiri. Teknik dan metode untuk menguji kehandalan algoritma sandi adalah kriptanalis.

Dasar matematis yang mendasari proses enkripsi dan dekripsi adalah relasi antara dua himpunan yaitu yang berisi elemen teks terang/*plaintext* dan yang berisi elemen teks sandi/*ciphertext*. Enkripsi dan dekripsi merupakan fungsi transformasi antara himpunan-himpunan tersebut. Apabila elemen-elemen teks terang dinotasikan dengan P, elemen-elemen teks sandi dinotasikan dengan C, sedang untuk proses enkripsi dinotasikan dengan E, dekripsi dengan notasi D.

Enkripsi : $E(P) = C$

Dekripsi : $D(C) = P$ atau $D(E(P)) = P$

Secara umum berdasarkan kesamaan kuncinya, algoritma sandi dibedakan menjadi:

1. Kunci-simetris/*symetric-key*, sering disebut juga algoritma sandi konvensional karena umumnya diterapkan pada algoritma sandi klasik.
2. Kunci-asimetris/*asymetric-key*.

Berdasarkan arah implementasi dan pembabakan jamannya dibedakan menjadi:

1. Algoritma sandi klasik/*classic cryptography*.
2. Algoritma sandi modern/*modern cryptography*.

Berdasarkan kerahasiaan kuncinya dibedakan menjadi:

1. Algoritma sandi kunci rahasia/*secret-key*.
2. Algoritma sandi kunci publik/*publik-key*.

Pada skema kunci-simetris, digunakan sebuah kunci rahasia yang sama untuk melakukan proses enkripsi dan dekripsinya. Sedangkan pada sistem kunci-asimetris digunakan sepasang kunci yang berbeda, umumnya disebut kunci publik (*public key*) dan kunci pribadi (*private key*), digunakan untuk proses enkripsi dan proses dekripsinya. Bila elemen teks terang dienkripsi dengan menggunakan kunci pribadi maka elemen teks sandi yang dihasilkannya hanya bisa didekripsikan dengan menggunakan pasangan kunci pribadinya. Begitu juga sebaliknya, jika kunci pribadi digunakan untuk proses enkripsi maka proses dekripsi harus menggunakan kunci publik pasangannya.

1. Algoritma Sandi Kunci-Simetris

Skema algoritma sandi akan disebut kunci-simetris apabila untuk setiap proses enkripsi maupun dekripsi data secara keseluruhan digunakan kunci yang sama. Skema ini berdasarkan jumlah data per proses dan alur pengolahan data didalamnya dibedakan menjadi dua kelas, yaitu *block-cipher* dan *stream-cipher*.

a. *Block-Cipher*

Block-cipher adalah skema algoritma sandi yang akan membagi-bagi teks terang yang akan dikirimkan dengan ukuran tertentu (disebut blok) dengan panjang *t*, dan setiap blok dienkripsi dengan menggunakan kunci yang sama. Pada umumnya, *block-cipher* memproses teks terang dengan blok yang relatif panjang lebih dari 64 *bit*, untuk mempersulit penggunaan pola-pola serangan yang ada untuk membongkar kunci. Untuk menambah kehandalan model algoritma sandi ini, dikembangkan pula beberapa tipe proses enkripsi, yaitu:

1. ECB, *Electronic Code Book*.
2. CBC, *Cipher Block Chaining*.
3. OFB, *Output Feed Back*.
4. CFB, *Cipher Feed Back*.

b. *Stream-Cipher*

Stream-cipher adalah algoritma sandi yang mengenkripsi data persatuan data, seperti *bit*, *byte*, *nible* atau per lima *bit* (saat data yang dienkripsi berupa data *Boudout*). Setiap mengenkripsi satu satuan data digunakan kunci yang merupakan hasil pembangkitan dari kunci sebelumnya.

Berikut ini adalah beberapa contoh algoritma yang menggunakan kunci-simetris:

1. DES - *Data Encryption Standard*.
2. *Blowfish*.
3. *Twofish*.
4. MARS
5. IDEA
6. 3DES - DES diaplikasikan 3 kali.
7. AES - *Advanced Encryption Standard*, yang bernama asli *Rijndael*.

2. Algoritma Sandi Kunci-Asimetris

Skema ini adalah algoritma yang menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsinya. Skema ini disebut juga sebagai sistem kriptografi kunci publik karena kunci untuk enkripsi dibuat untuk diketahui oleh umum (*public-key*) atau dapat diketahui siapa saja, tapi untuk proses dekripsinya hanya dapat dilakukan oleh yang berwenang yang memiliki kunci rahasia untuk mendekripsinya, disebut *private-key*. Dapat dianalogikan seperti kotak pos yang hanya dapat dibuka oleh tukang pos yang memiliki kunci tapi setiap orang dapat memasukkan surat ke dalam kotak tersebut.

Keuntungan algoritma model ini, untuk berkorespondensi secara rahasia dengan banyak pihak tidak diperlukan kunci rahasia sebanyak jumlah pihak tersebut, cukup membuat dua buah kunci, yaitu kunci publik bagi para korensponden untuk mengenkripsi pesan, dan kunci pribadi untuk mendekripsi pesan. Berbeda dengan skema kunci-simetris, jumlah kunci yang dibuat adalah sebanyak jumlah pihak yang diajak berkorespondensi. Berikut ini adalah contoh

fungsi enkripsi dan dekripsi algoritma sandi kunci-asimetris jika Ahmad dan Bejo hendak bertukar informasi:

1. Ahmad dan Bejo masing-masing membuat 2 buah kunci, yaitu:
   1) Ahmad membuat dua buah kunci, kunci-publik $K_{publik[Ahmad]}$ dan kunci-privat $K_{privat[Ahmad]}$
   2) Bejo membuat dua buah kunci, kunci-publik $K_{publik[Bejo]}$ dan kunci-privat $K_{privat[Bejo]}$
2. Mereka berkomunikasi dengan cara:
   1) Ahmad dan Bejo saling bertukar kunci-publik. Bejo mendapatkan $K_{publik[Ahmad]}$ dari Ahmad, dan Ahmad mendapatkan $K_{publik[Bejo]}$ dari Bejo.
   2) Ahmad mengenkripsi teks-terang P ke Bejo dengan fungsi $C = E(P, K_{publik[Bejo]})$
   3) Ahmad mengirim teks-sandi C ke Bejo
   4) Bejo menerima C dari Ahmad dan membuka teks-terang dengan fungsi $P = D(C, K_{privat[Bejo]})$
3. Hal yang sama terjadi apabila Bejo hendak mengirimkan pesan ke Ahmad, yaitu:
   1) Bejo mengenkripsi teks-terang P ke Ahmad dengan fungsi $C = E(P, K_{publik[Ahmad]})$

2) Ahmad menerima C dari Bejo dan membuka teks-terang dengan fungsi

$$P = D(C, K_{privat[Ahmad]})$$

Berikut ini adalah beberapa contoh algoritma sandi kunci-asimetris:

1. *Knapsack*.
2. *Rivert-Shamir-Adelman* (RSA).
3. *Diffie-Hellman*.

2.2.2.2 Fungsi *Hash* Kriptografi

Fungsi *hash* kriptografi adalah fungsi *hash* yang memiliki beberapa sifat keamanan tambahan sehingga dapat dipakai untuk tujuan keamanan data. Umumnya digunakan untuk keperluan autentikasi dan integritas data. Fungsi *hash* adalah fungsi yang secara efisien mengubah *string input* dengan panjang berhingga menjadi *string output* dengan panjang tetap yang disebut nilai *hash*. Berikut ini adalah fungsi-fungsi *hash* kriptografi:

1. Tahan preimej (*Preimage resistant*): bila diketahui nilai hash *h* maka sulit (secara komputasi tidak layak) untuk mendapatkan m dimana *h* = hash(*m*).
2. Tahan preimej kedua (*Second preimage resistant*): bila diketahui input m_1 maka sulit mencari input m_2 (tidak sama dengan m_1) yang menyebabkan hash(m_1) = hash(m_2).
3. Tahan tumbukan (*Collision-resistant*): sulit mencari dua input berbeda m_1 dan m_2 yang menyebabkan hash(m_1) = hash(m_2).

Berikut ini adalah beberapa contoh algoritma fungsi *hash* kriptografi:

1. MD4
2. MD5

3. SHA-0
4. SHA-1
5. SHA-256
6. SHA-512

2.2.3 Pengertian Kompresi Data

Dalam ilmu komputer dan teori informasi, kompresi data adalah proses encoding informasi menggunakan *bit* lebih sedikit daripada representasi asli yang digunakan. Kompresi berguna karena membantu mengurangi konsumsi sumber daya yang mahal, seperti ruang *hard disk* atau *bandwidth* transmisi. Pada sisi negatifnya, data dikompresi harus didekompresi untuk digunakan, dan pengolahan tambahan ini mungkin merugikan beberapa aplikasi. Sebagai contoh, skema kompresi untuk video mungkin memerlukan perangkat keras yang mahal untuk video yang akan didekompresi dan cukup cepat untuk dilihat ketika sedang didekompresi (pilihan untuk dekompresi video secara penuh sebelum menonton mungkin nyaman, dan membutuhkan ruang penyimpanan untuk dekompresi video). Oleh sebab itu, terjadi persaingan dalam rancangan skema kompresi data di berbagai faktor, termasuk tingkat kompresi, jumlah distorsi yang diperkenalkan (jika menggunakan skema kompresi *lossy*), dan sumber daya komputasi yang dibutuhkan untuk kompresi dan dekompresi data.

Prinsip umum yang digunakan pada proses kompresi adalah mengurangi duplikasi data sehingga memori untuk merepresentasikan menjadi lebih sedikit daripada representasi data digital semula. Data digital yang telah dikompresi dapat dikembalikan ke bentuk data digital semula (dekompresi) dimana hal ini

tergantung pada aplikasi yang mendukung kompresi tersebut. Ketika suatu aplikasi mampu mengkompresi data yang tidak dibutuhkan maka aplikasi tersebut juga mampu mengembalikan data yang dikompresi tersebut sehingga menjadi data digital semula (dekompresi). Namun terdapat juga suatu aplikasi yang dapat mengkompresi tetapi ketika dekompresi dapat menggunakan aplikasi lain contohnya aplikasi *winzip* untuk mengkompresi dengan aplikasi *winrar* untuk dekompresi. Terdapat dua metode kompresi data, yaitu kompresi data *lossless* dan kompresi data *lossy*.

2.2.3.1 Kompresi Data *Lossless*

Kompresi data *lossless* adalah kelas dari algoritma data kompresi yang memungkinkan data yang asli dapat disusun kembali dari data kompresi. Kompresi data *lossless* digunakan dalam berbagai aplikasi seperti format ZIP dan GZIP. *Lossless* juga sering digunakan sebagai komponen dalam teknologi kompresi data *lossy*. Kompresi *lossless* digunakan ketika terdapat informasi yang penting pada data dalam kondisi asli.

Kompresi data *lossless* menghasilkan data yang identik dengan data aslinya, hal ini dibutuhkan oleh banyak tipe data, contohnya: *executable code*, *word processing files, tabulated numbers*, dan sebagainya. Rasio kompresi (rasio kompresi yaitu, ukuran file yang dikompresi dibanding yang tak terkompresi dari file) dengan metode ini sangat rendah. Metode ini cocok untuk kompresi citra yang mengandung informasi penting yang tidak boleh rusak akibat kompresi, misalnya gambar hasil diagnosa medis. Contoh metode *lossless* adalah metode *Run-Length Encoding, Huffman, delta* dan *Lempel-Ziv-Welch (*LZW*)*.

2.2.3.2 Kompresi Data *Lossy*

Kompresi data *lossy* adalah suatu metode untuk mengkompresi data dan men-dekompresinya dengan hasil data yang diperoleh mungkin berbeda dari yang aslinya tetapi cukup dekat perbedaaanya. Kompresi data *lossy* ini paling sering digunakan untuk kompres data multimedia (*audio, video,* dan gambar).

Format kompresi *lossy* mengalami *generation loss* yaitu jika dilakukan berulang kali kompresi dan dekompresi akan menyebabkan data kehilangan kualitas secara progresif. Hal ini berbeda dengan kompresi data *lossless,* ketika pengguna yang menerima file terkompresi secara *lossy* (misalnya untuk mengurangi waktu *download*) data yang diambil dapat sedikit berbeda dari yang asli di-*level bit* ketika tidak dapat dibedakan oleh mata dan telinga manusia untuk tujuan paling praktis.

Metode ini menghasilkan rasio kompresi yang lebih besar daripada metode *lossless*. Misal terdapat gambar asli berukuran 12.249 *bytes,* kemudian dilakukan kompresi dengan JPEG kualitas 30 dan berukuran 1.869 *bytes* berarti gambar tersebut 85% lebih kecil dan rasio kompresi 15%. Contoh metode *lossy* adalah metode CS&Q (*Coarser Sampling and*/*or Quantization*), JPEG, dan MPEG.

2.2.4 Metodologi dan Metode Pengembangan Perangkat Lunak

Metodologi dan metode pengembangan perangkat lunak yang digunakan adalah metodologi *System Development Life Cycle* (SDLC) dengan pemodelan *waterfall*, metodologi *Object Modeling Technique* (OMT) dengan pemodelan *Unified Modeling Language* (UML) metode enkripsi kunci-simetris dengan

algoritma *Rijndael*, dan metode kompresi *lossless* dengan algoritma *Lempel-Ziv-Welch* (LZW).

2.2.4.1 Metodologi System Development Life Cycle (SDLC)

System Development Life Cycle (SDLC) dalam rekayasa sistem dan rekayasa perangkat lunak, adalah proses pembuatan dan pengubahan sistem serta model dan metodologi yang digunakan untuk mengembangkan sistem-sistem tersebut. Konsep ini umumnya merujuk pada sistem komputer atau informasi. Dalam rekayasa perangkat lunak, konsep *System Development Life Cycle* (SDLC) mendasari berbagai jenis metodologi pengembangan perangkat lunak. Terdapat enam tahap dalam metode *System Development Life Cycle* (SDLC) yang dimodelkan oleh *waterfall* seperti pada Gambar 2.2 [6], yaitu:

1. *Requirements Analysis and Definition*

   Mengumpulkan kebutuhan secara lengkap kemudian kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh program yang akan dibangun. Fase ini harus dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap.

2. *System and Software Design*

   Desain dikerjakan setelah kebutuhan selesai dikumpulkan secara lengkap.

3. *Implementation and Unit Testing*

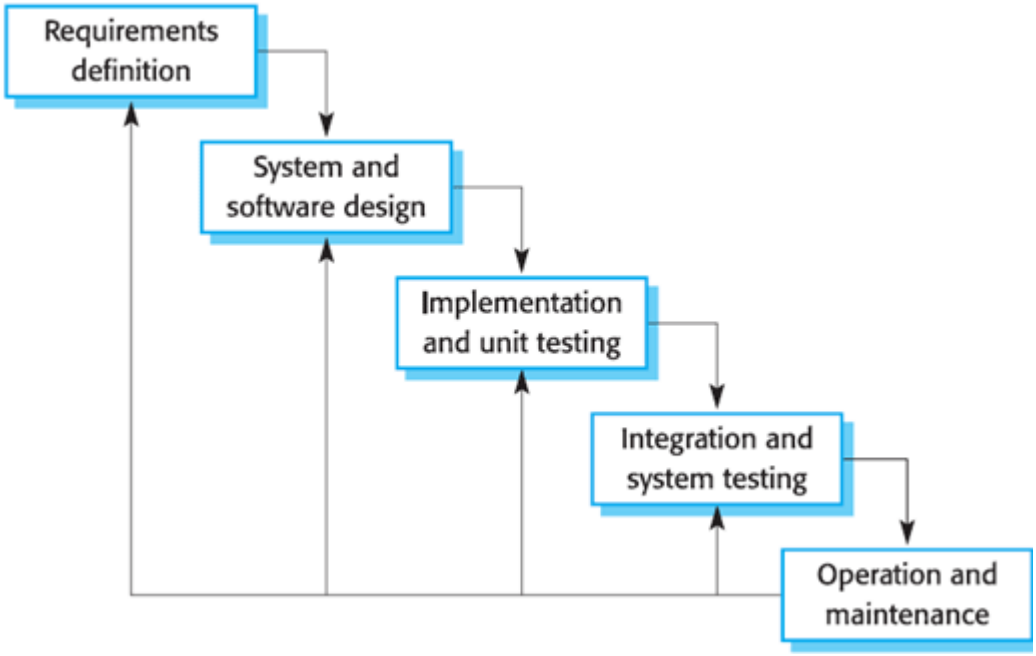
   Desain program diterjemahkan ke dalam kode-kode dengan menggunakan bahasa pemrograman yang sudah ditentukan. Program yang dibangun langsung diuji baik secara unit.

4. *Integration and System Testing*

   Penyatuan unit-unit program kemudian diuji secara keseluruhan.

5. *Operation and Maintenance*

   Mengoperasikan program di lingkungannya dan melakukan pemeliharaan, seperti penyesuaian atau perubahan karena adaptasi dengan situasi sebenarnya.



Gambar 2.2 Model *Waterfall* [6]

2.2.4.2 Metodologi Object Modeling Technique (OMT)

Teknik pemodelan objek (OMT) adalah pemodelan obyek bahasa untuk memodelkan dan merancang. Ini dikembangkan sekitar tahun 1991 oleh Rumbaugh, Blaha, Premerlani, Eddy dan Lorensen sebagai metodologi untuk mengembangkan sistem berorientasi obyek, dan untuk mendukung pemrograman berorientasi obyek.

OMT dikembangkan sebagai pendekatan untuk pengembangan perangkat lunak. Tujuan pemodelan [11] adalah:

1. Entitas pengujian fisik sebelum membangun perangkat lunak (simulasi).
2. Komunikasi dengan pengguna.
3. Visualisasi (alternatif presentasi informasi).
4. Pengurangan kompleksitas.

OMT telah mengusulkan tiga jenis utama dari model, yaitu:

1. Model Obyek

   Model obyek merupakan fenomena yang statis dan paling stabil dalam domain pemodelan. Konsep utamanya adalah kelas dan asosiasi, dengan atribut dan operasi. Agregasi dan generalisasi (dengan *multiple inheritance*) merupakan standar hubungan.

2. Model Dinamis

   Model dinamis merupakan kondisi/transisi pada model. Konsep utamanya adalah kondisi, transisi antara kodisi, dan peristiwa untuk memicu transisi. Tindakan dapat dimodelkan sebagai peristiwa dalam kondisi. Generalisasi dan agregasi (konkurensi) adalah standar hubungan.

3. Model Fungsional

   Model fungsional menangani perspektif proses model, kira-kira sesuai untuk diagram arus data. Konsep utamanya adalah proses, menyimpan data, aliran data, dan aktor.

OMT adalah pendahulu dari *Unified Modeling Language* (UML). Banyak elemen pemodelan OMT yang umum digunakan pada UML.

2.2.4.3 Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. *Unified Modeling Language* (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi obyek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat tool untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh Object Management Group, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat desain berorientasi obyek dari IBM.

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem.

Sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah metodologi *booch*, metodologi *coad*, metodologi OOSE, metodologi OMT, metodologi *shlaer-mellor*, metodologi *wirfs-brock*, dan sebagainya. Masa itu terkenal dengan masa perang

metodologi dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila bekerjasama dengan kelompok/perusahaan lain yang menggunakan metodologi yang berlainan.

Dimulai pada bulan Oktober 1994, Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang metodologinya banyak digunakan, mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 dirilis *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG).

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi obyek seperti pada Gambar 2.3, yaitu:

1. *Use Case Diagram*

   Use case diagram digunakan untuk memodelkan bisnis proses berdasarkan perspektif pengguna sistem. *Use case diagram* terdiri atas diagram untuk *use case* dan *actor*. *Actor* merepresentasikan orang yang akan mengoperasikan atau orang yang berinteraksi dengan sistem aplikasi. *Use case* merepresentasikan operasi-operasi yang dilakukan oleh *actor*. *Use case* digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnya. *Actor* yang melakukan operasi dihubungkan dengan garis lurus ke *use case*.
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.

3. *Sequence Diagram*

   Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*, yaitu interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

4. *Collaboration Diagram*

   Collaboration diagram dipakai untuk memodelkan interaksi antar object di dalam sistem. Berbeda dengan *sequence diagram* yang lebih menonjolkan kronologis dari operasi-operasi yang dilakukan, *collaboration diagram* lebih fokus pada pemahaman atas keseluruhan operasi yang dilakukan oleh obyek.

5. *State Diagram* untuk memodelkan perilaku obyek di dalam sistem.

6. *Activity Diagram* untuk memodelkan perilaku *Use Cases* dan obyek di dalam sistem.

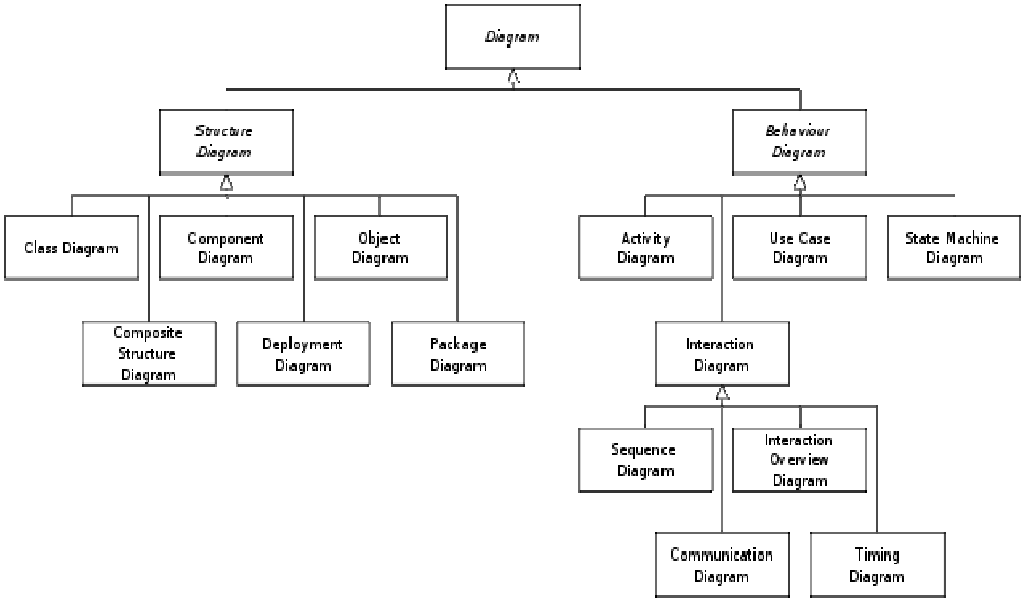
7. *Class Diagram*

   Class diagram merupakan diagram yang selalu ada pada permodelan sistem berorientasi obyek. *Class diagram* menunjukkan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.

8. *Object Diagram* untuk memodelkan struktur object.

9. *Component Diagram* untuk memodelkan komponen object.

10. *Deployment Diagram* untuk memodelkan distribusi aplikasi.



Gambar 2.3 Diagram UML [12]

2.2.4.4 Metode Enkripsi dengan Algoritma *Rijndael*

Rijndael merupakan *block cipher* yang didesain oleh Joan Daemen dan Vincent Rijmen. Desain dari *Rijndael* sangat dipengaruhi oleh desain dari blok *cipher square* yang juga merupakan hasil desain Daemen dan Rijmen. Nama algoritma ini merupakan kombinasi dari kedua penemunya tersebut [4].

Rijndael termasuk dalam jenis algoritma kriptografi yang sifatnya simetri dan *cipher block*. Dengan demikian algoritma ini mempergunakan kunci yang sama saat enkripsi dan dekripsi serta masukan dan keluarannya berupa blok dengan jumlah *bit* tertentu. *Rijndael* mendukung berbagai variasi ukuran blok dan kunci yang akan digunakan. Namun *Rijndael* mempunyai ukuran blok dan kunci yang tetap sebesar 128, 192, 256 *bit*. Pemilihan ukuran blok data dan kunci akan menentukan jumlah proses yang harus dilalui untuk proses enkripsi dan dekripsi.

Berikut adalah perbandingan jumlah proses yang harus dilalui untuk masing-masing masukan yang dapat dilihat pada Tabel 2.3 .

Tabel 2.3 Perbandingan Jumlah Proses

Panjang Kunci (Nk) Dalam words	Ukuran Blok Data (Nb) Dalam words	Jumlah Proses (Nr)
4	4	10
6	4	12
8	4	14

Blok-blok data masukan dan kunci dioperasikan dalam bentuk *array*. Setiap anggota *array* sebelum menghasilkan keluaran *ciphertext* dinamakan dengan *state*. Secara umum, *state* algoritma *Rijndael* terdiri dari:

1. *Initial Round Key Addition*:
   a. *AddRoundKey*
2. *Nr-1 Rounds*:
   a. *ByteSubb*.
   b. *ShiftRow*.
   c. *MixColumn*.
   d. *AddRoundKey*.
3. *Final Round*:
   a. *ByteSubb.*
   b. *ShiftRow*.
   c. *AddRoundKey.*

Setiap *state* akan mengalami proses yang secara garis besar terdiri dari empat tahap, yaitu *AddRoundKey*, *SubBytes*, *ShiftRows*, dan *MixColumns*. Kecuali tahap *MixColumns*, ketiga tahap lainnya akan diulang pada setiap proses, sedangkan tahap *MixColumns* tidak akan dilakukan pada tahap terakhir. Proses dekripsi adalah kebalikkan dari enkripsi.

Karena terjadi beberapa tahap dalam proses enkripsi, maka diperlukan *subkey-subkey* yang akan dipakai pada tiap tahap. Pengembangan jumlah kunci yang akan dipakai diperlukan karena kebutuhan *subkey-subkey* yang akan dipakai dapat mencapai ribuan bit, sedangkan kunci yang disediakan secara *default* hanya 128-256 bit. Jumlah total kunci yang diperlukan sebagai *subkey* adalah sebanyak Nb(Nr+1), dimana *Nb* adalah besarnya blok data dalam satuan *word*.

Sedangkan *Nr* adalah jumlah tahapan yang harus dilalui dalam satuan *word*. Sebagai contoh, bilamana digunakan 128 *bit* (4 *word*) blok data dan 128 *bit* (4 *word*) kunci maka akan dilakukan 10 kali proses. Dengan demikian dari rumus didapatkan 4(10+1) = 44 word = 1408 *bit* kunci. Untuk melakukan pengembangan jumlah kunci yang akan dipakai dari kunci utama maka dilakukan *key schedule*.

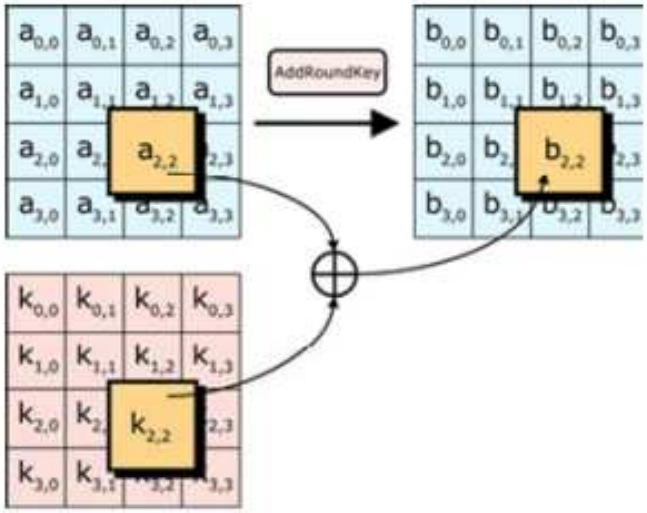
1. *Key Schedule*

Proses *key schedule* diperlukan untuk mendapatkan *subkey-subkey* dari kunci utama agar cukup untuk melakukan enkripsi dan dekripsi. Proses ini terdiri dari beberapa operasi, yaitu:

1. Operasi *Rotate*, yaitu operasi perputaran 8 *bit* pada 32 *bit* dari kunci.
2. Operasi *SubBytes*, pada operasi ini 8 *bit* dari *subkey* disubstitusikan dengan nilai dari S-*Box*.
3. Operasi *Rcon*, operasi ini dapat diterjemahkan sebagai operasi pangkat 2 nilai tertentu dari *user*. Operasi ini menggunakan nilai-nilai dalam *Galois field*. Nilai-nilai dari *Rcon* kemudian akan di-XOR dengan hasil operasi *SubBytes*.
4. Operasi XOR dengan *w[i-Nk]* yaitu *word* yang berada pada *Nk* sebelumnya.

2. *AddRoundKey*

Pada proses ini *subkey* digabungkan dengan *state*. Proses penggabungan ini menggunakan operasi XOR untuk setiap *byte* dari *subkey* dengan *byte* yang bersangkutan dari *state*. Untuk setiap tahap, *subkey* dibangkitkan dari kunci utama dengan menggunakan proses *key schedule*. Setiap *subkey* berukuran sama dengan *state* yang bersangkutan. Proses *AddRoundKey* dapat dilihat pada Gambar 2.4.



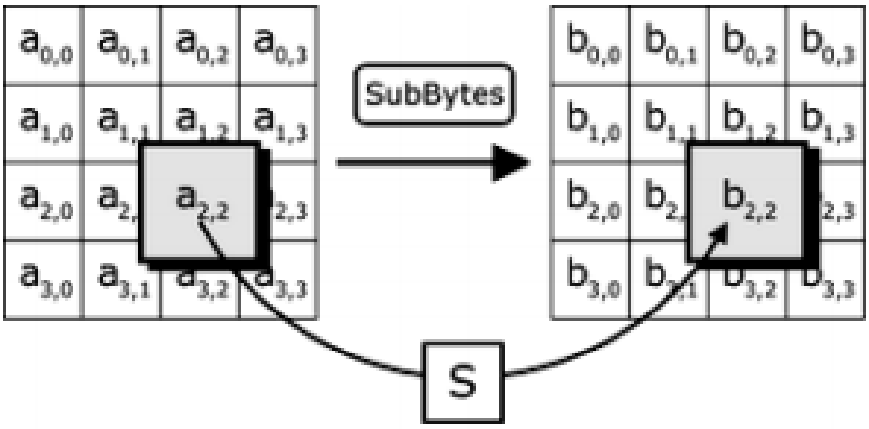
Gambar 2.4 Proses *AddRoundKey*

3. *SubBytes*

Proses *SubBytes* adalah operasi yang akan melakukan substitusi tidak linear dengan cara mengganti setiap *byte state* dengan *byte* pada sebuah tabel yang dinamakan tabel S-*Box*. Sebuah tabel S-*Box* terdiri dari 16x16 baris dan kolom dengan masing-masing berukuran 1 *byte*. Tabel S-*Box* dapat dilihat pada Tabel 2.4, sedangkan proses *SubBytes* dapat dilihat pada Gambar 2.5.

Tabel 2.4 S-*Box*

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

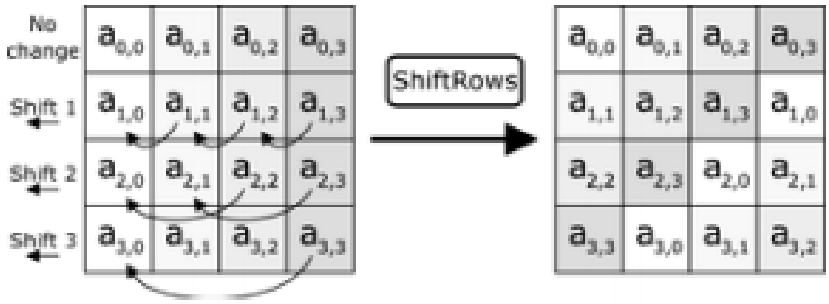


Gambar 2.5 Proses *SubBytes*

4. *ShiftRows*

Proses *ShiftRows* akan beroperasi pada tiap baris dari tabel *state*. Proses ini akan bekerja dengan cara memutar *byte-byte* pada 3 baris terakhir (baris 1, 2, dan 3) dengan jumlah perputaran yang berbeda-beda. Baris 1 akan diputar sebanyak 1 kali, baris 2 akan diputar sebanyak 2 kali, dan baris 3 akan diputar sebanyak 3

kali. Sedangkan baris 0 tidak akan diputar. Proses *ShiftRows* dapat dilihat pada Gambar 2.6.



Gambar 2.6 Proses *ShiftRows*

5. *MixColumns*

Proses *MixColumns* akan beroperasi pada tiap kolom dari tabel *state*. Operasi ini menggabungkan 4 *bytes* dari setiap kolom tabel *state* dan menggunakan transformasi linier.

Operasi *MixColumns* memperlakukan setiap kolom sebagai polinomial 4 suku dalam *Galois field* dan kemudian dikalikan dengan c(x) modulo (x4+1), dimana c(x)=3x3+x2+x+2. Kebalikkan dari polinomial ini adalah c(x)=11x3+13x2+9x+14. Operasi *MixColumns* juga dapat dipandang sebagai perkalian matrix. Langkah *MixColumns* dapat ditunjukkan dengan mengalikan 4 bilangan di dalam Galois field oleh matrix berikut ini:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Atau bila dijabarkan:

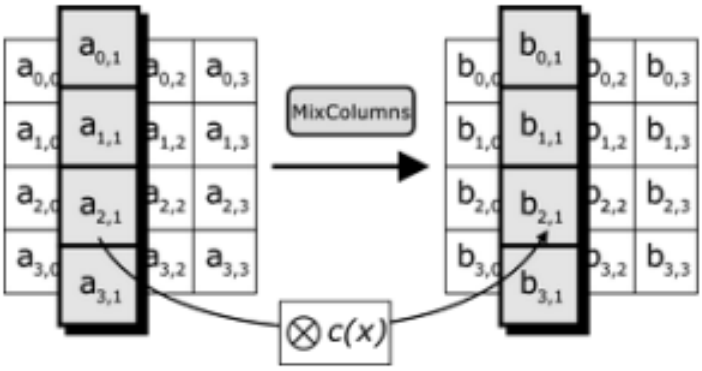
r0 = 2a0 + a3 + a2 + 3a1

r1 = 2a1 + a0 + a3 + 3a2

r2 = 2a2 + a1 + a0 + 3a3

a3 = 2a3 + a2 + a1 + 3a0

Operasi penjumlahan di atas dilakukan dengan operasi XOR, sedangkan operasi perkalian dilakukan dalam *Galois field* seperti pada Gambar 2.7.



Gambar 2.7 Proses *MixColumns*

2.2.4.5 Metode Kompresi dengan Algoritma *Lempel-Ziv-Welch* (LZW)

Algoritma *Lempel-Ziv-Welch* (LZW) merupakan algoritma kompresi yang bersifat *lossless* dan menggunakan metode *dictionary*. Algoritma ini ditemukan oleh Lemple, Ziv, dan Welch pada tahun 1984. Secara umum, algoritma kompresi *Lempel-Ziv-Welch* (LZW) akan membentuk *dictionary* selama proses kompresinya berlangsung kemudian setelah selesai maka *dictionary* tersebut tidak ikut disimpan dalam data yang telah terkompresi. Prisinp kompresi akan terjadi ketika besar *bit* untuk *dictionary* yang telah ditentukan menggantikan deretan karakter atau *string* yang terbentuk, sedangkan dalam proses dekompresinya

untuk memperoleh hasil yang sama dengan *file* sebelum dikompresi akan membuat kembali *dictionary* selama proses dekompresinya berlangsung [1].

Sehingga dapat dijelaskan bahwa prinsip umum kerja algoritma *Lempel-Ziv-Welch* (LZW) adalah mengecek setiap karakter yang muncul kemudian menggabungkan dengan karakter selanjutnya menjadi sebuah *string.* Jika *string* baru tersebut tidak berada dalam *dictionary* atau belum diindekkan, maka *string* baru tersebut akan diindekkan ke dalam *dictionary*. Berikut ini adalah *pseudocode* kompresi algoritma *Lempel-Ziv-Welch* (LZW):

```
a= NIL
while (ada inputan kemudian baca sebuah karakter x)
if (ax ada dalam kamus)
then a = ax
else
add ax to the dictionary
output indek untuk a
a = x
```

Dari *pseudocode* algoritma *Lempel-Ziv-Welch* (LZW) dapat dijelaskan bahwa:

1. Inisialisasi *input stream* yang mengandung karakter-karekter dasar.
2. Membaca karakter dari *input stream*, jika EOF maka menuju langkah 5.
3. Gabungkan karater awal dengan karakter yang dibaca selanjutnya menjadi sebuah *string*.
4. Jika string baru ini tidak ada dalam *dictionary*, maka:
    a. Buat indek baru untuk *string* baru tersebut.
    b. Menuju Langkah 2.
5. Jika *string* yang terbentuk telah mempunyai indek di dalam *dictionary*, maka:

a. Gabungkan karakter sebelumnya dengan karakter yang dibaca atau *string* yang telah terbentuk menjadi sebuah string baru.
b. Menuju langkah 2.

6. Tulis output kode.

Untuk contoh dari algoritma diatas dapat dibuat implementasi pada contoh dibawah ini seperti pada Tabel 2.5. dan Tabel 2.6.:

Input string : *TES_TEA_EAT_MEET_SLEEP*

bit dictionary : 9 *bit*

Total awal *bit* disimpan tanpa kompresi

= Total *input* * *bit dictionary* = 22 * 9 = 198 *bit*.

Besar data setelah dikompresi

= Total *output* * *bit dictionary* = 19 * 9 = 171 *bit*.

Input stream: *TES_<256>A_EAT_MEE<265>SL<268>P*.

Tabel 2.5 Proses Kompresi Algoritma Lempel-Ziv-Welch (LZW)

	C	w	entry	w+entry[0]	ouput	dictionary
1	T	T			T	
2	E	T	E	TE	E	256 = TE
3	S	E	S	ES	S	257 = ES
4	_	S	_	S_	_	258 = S_
5	<256>	_	TE	_T	TE	259 = _T
6	A	TE	A	TA	A	260 = TA
7	_	A	_	A_	_	261 = A_
8	E	_	E	_E	E	262 = _E
9	A	E	A	EA	A	263 = EA
10	T	A	T	AT	T	264 = AT
11	_	T	_	T_	_	265 = T_
12	M	_	M	_M	M	266 = _M
13	E	M	E	ME	E	267 = ME
14	E	E	E	EE	E	268 = EE
15	<265>	E	T_	ET	T_	269 = ET
16	S	T_	S	T_S	S	270 = T_S
17	L	S	L	SL	L	271 = SL
18	<268>	L	EE	LE	EE	272 = LE
19	P	EE	P	EP	P	273 = EEP
20		P				S

Tabel 2.6 Proses Dekompresi Algoritma Lempel-Ziv-Welch (LZW)

	Input	String + char	in *Dictionary* ?	Temporary	New *Code Word* (sub String)	Output
1	T	T	-	T	-	none
2	E	TE	N	E	256 = TE	T
3	S	ES	N	S	257 = ES	E
4	_	S_	N	_	258 = S_	S
5	T	_T	N	T	259 = _T	_
6	E	TE	Y (256)	TE	-	none
7	A	TEA	N	A	260 = TEA	256
8	_	A_	N	_	261 = A_	A
9	E	_E	N	E	262 = _E	_
10	A	EA	N	A	263 = EA	E
11	T	AT	N	T	264 = AT	A
12	_	T_	N	_	265 = T_	T
13	M	_M	N	M	266 = _M	_
14	E	ME	N	E	267 = ME	M
15	E	EE	N	E	268 = EE	E
16	T	ET	N	T	269 = ET	E
17	_	T_	Y (265)	T_	-	none
18	S	T_S	N	S	270 = T_S	265
19	L	SL	N	L	271 = SL	S
20	E	LE	N	E	272 = LE	L
21	E	EE	Y (268)	EE	-	none
22	P	EEP	N	P	273 = EEP	268
23						P

2.2.5 Perangkat Lunak Pendukung

Pada bagian ini dijelaskan tentang perangkat lunak dan bahasa pemrograman yang digunakan untuk mendukung dalam pengembangan aplikasi yang dibuat. Bahasa pemrograman yang digunakan adalah Visual Basic .NET dan menggunakan lingkungan pembangunan terintegrasi dari Microsoft Visual Basic 2008 Express Edition.

2.2.5.1 Pengertian *Visual Basic* .NET

Visual Basic .NET adalah bahasa pemrograman komputer berorientasi obyek yang dapat dilihat sebagai sebuah evolusi dari *Visual Basic* (VB) klasik,

yang diimplementasikan pada .NET *Framework*. Microsoft saat ini memasok dua implementasi utama *Visual Basic*, yaitu Microsoft Visual Studio yang berbayar dan Microsoft Visual Studio Express yang gratis. Implementasi Microsoft Visual Basic .NET dinamakan *Microsoft Visual Basic.*

Terdapat empat versi dan lima rilis Visual Basic .NET yang dilakukan oleh *Visual Basic Team*, yaitu:

1. Visual Basic .NET 2002 (VB 7.0)

   Versi pertama dari Visual Basic .NET adalah Visual Basic .NET 2002 yang dirilis pertama kali pada bulan Februari 2002. Visual Basic .NET 2002 merupakan sebuah bahasa pemrograman visual yang berbasis bahasa BASIC (sama seperti halnya Visual Basic 6.0, tetapi lebih disempurnakan dan lebih berorientasi objek), dan didesain untuk berjalan di atas Microsoft .NET Framework versi 1.0.

   Versi 7.0 ini dirilis bersamaan dengan *Visual* C# dan ASP.NET. Bahasa C#, yang dianggap sebagai jawaban terhadap *Java*, mendapatkan perhatian yang lebih banyak dibandingkan dengan VB.NET yang kurang begitu banyak diulas. Hasilnya, sedikit orang di luar komunitas *Visual Basic* yang memperhatikan VB.NET. Versi pertama ini kurang mendapat sambutan yang bagus dari para *programmer*, dan pada saat itu, program berbasis *Visual Basic* 6.0 sedang marak-maraknya dibuat. Para *programmer* yang mencoba *Visual Basic* .NET untuk pertama kali akan merasakan bahwa *Visual Basic* .NET sangatlah berbeda dibandingkan dengan *Visual Basic* sebelumnya. Contoh

yang paling mudah adalah *runtime engine* yang lebih besar 10 kali lipat dibandingkan *Visual Basic* 6.0, dan juga meningkatkan beban di memori.

2. Visual Basic .NET 2003 (VB 7.1)

   Visual Basic .NET 2003 dirilis bersama .NET *Framework* 1.1. Terdapat fitur baru yang mendukung .NET *Compact Framework* dan VB *upgrade wizard* yang lebih baik. Peningkatan performa dan relibilitas .NET IDE (khususnya *background compiler*) dan *runtime*. Tambahan, Visual Basic .NET 2003 juga terdapat versi Visual Studio .NET Academic Edition (VS03AE). VS03AE didistribukan kepada beberapa sekolah di setiap negara secara gratis.

3. Visual Basic 2005 (VB 8.0)

   Setelah itu, Microsoft pun berkonsentrasi dalam mengembangkan *Microsoft* .NET *Framework* 2.0, dan tentunya alat bantu untuk membangun program di atasnya. Hingga pada tahun 2005, mereka pun merilis versi terbaru dari Visual Basic .NET, yang kali ini disebut dengan Visual Basic 2005 (dengan membuang kata ".NET"), bersama-sama dengan beberapa aplikasi pengembangan lainnya.

   Untuk rilis 2005 ini, *Microsoft* menambahkan beberapa fitur baru, di antaranya adalah:

   a. *Edit and Continue*

      Fitur ini sebelumnya terdapat di dalam Visual Basic, akan tetapi dihapus di dalam Visual Basic .NET. Dengan keberadaan fitur ini, para programmer dapat memodifikasi kode pada saat program dieksekusi dan melanjutkan proses eksekusi dengan kode yang telah dimodifikasi tersebut.

b. Evaluasi ekspresi pada saat waktu desain.

c. Munculnya *Pseudo-Namespace* "`My`", yang menyediakan:

   - Akses yang mudah terhadap beberapa area tertentu dari dalam .NET *Framework* yang tanpanya membutuhkan kode yang sangat signifikan.
   - Kelas-kelas yang dibuat secara dinamis (khususnya `My.Forms`).

d. Peningkatan yang dilakukan terhadap konverter kode sumber dari *Visual Basic* ke *Visual Basic* .NET.

e. Penggunaan kata kunci (*keyword*) *Using*, yang menyederhanakan penggunaan objek-objek yang membutuhkan pola *Dispose* untuk membebaskan sumber daya yang sudah tidak terpakai.

f. *Just My Code*, yang menyembunyikan kode *reusable* yang ditulis oleh alat bantu *Integrated Development Environment* (IDE) *Visual Studio* .NET.

g. Pengikatan sumber data (Data Source binding), yang mampu mempermudah pengembangan aplikasi basis data berbasis klien/server.

Fungsi-fungsi yang tersebut di atas (khususnya `My`) ditujukan untuk memfokuskan Visual Basic .NET sebagai sebuah *platform* pengembangan aplikasi secara cepat dan menjauhkannya dari bahasa C#.

Bahasa Visual Basic 2005 memperkenalkan fitur-fitur baru, yakni:

a. Bawaan .NET *Framework* 2.0:

   - *Generics*
   - *Partial class*, sebuah metode yang dapat digunakan untuk mendefinisikan beberapa bagian dari sebuah kelas di dalam sebuah berkas, lalu menambahkan definisinya di lain waktu; sangat berguna

khususnya ketika mengintegrasikan kode pengguna dengan kode yang dibuat secara otomatis.

- *Nullable Type*

b. Komentar XML yang dapat diproses dengan menggunakan beberapa alat bantu seperti NDoc untuk membuat dokumentasi secara otomatis.

c. *Operator overloading.*

d. Dukungan terhadap tipe data bilangan bulat tak bertanda (unsigned integer) yang umumnya digunakan di dalam bahasa lainnya.

4. Visual Basic 2008 (VB 9.0)

Versi ini merupakan versi terbaru yang dirilis oleh *Microsoft* pada tanggal 19 November 2007, bersamaan dengan dirilisnya *Microsoft Visual* C# 2008, *Microsoft Visual* C++ 2008, dan *Microsoft* .NET *Framework* 3.5.

Dalam versi ini, *Microsoft* menambahkan banyak fitur baru, termasuk di antaranya adalah:

a. Operator If sekarang merupakan operator ternary (membutuhkan tiga operand), dengan sintaksis `If` (`boolean`, `nilai`, `nilai`). Ini dimaksudkan untuk mengganti fungsi IIF.

b. Dukungan *anonymous types*

c. Dukungan terhadap *Language Integrated Query* (LINQ)

d. Dukungan terhadap ekspresi Lambda

e. Dukungan terhadap literal XML

f. Dukungan terhadap inferensi tipe data.

5. Visual Basic 2010 (VB 10.0)

   Pada April 2010, *Microsoft* merilis *Visual Basic* 2010. *Microsoft* merencanakan untuk menggunakan *Dynamic Language Rontime* (DLR) pada rilis tersebut tetapi bergeser kepada strategi evolusi gabungan antara *Visual Basic* dan C# untuk membawa kedua bahasa tersebut ke paritas yang lebih dekat satu sama lainnya. Kemampuan bawaan *Visual Basic* untuk berinteraksi secara dinamis dengan CLR dan obyek COM telah ditingkatkan untuk bekerja dengan bahasa dinamis yang dibangun pada DLR seperti *IronPython* dan *IronRuby*. *Compiler Visual Basic* telah ditingkatkan untuk menyimpulkan kelanjutan baris dalam satu set konteks umum, dalam banyak kasus menghilangkan kebutuhan untuk karakter baris kelanjutan "_". Juga, dukungan yang ada sudah dilengkapi *inline Functions* dengan dukungan untuk *inline Subs* serta versi *multi-line* baik *Sub* dan *Function* lambda.

Berikut ini adalah contoh program VB.NET, yang menggunakan *MsgBox* atau *MessageBox class*:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
        Msgbox("Hello, World")
    End Sub
End Class

Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
        MessageBox.Show("Hello, World")
    End Sub
End Class
```

2.2.5.2 Pengertian Microsoft Visual Basic 2008 Express Edition

Terlepas dari kenyataan bahwa *Visual Basic Express* adalah versi ringan *Visual Studio*, beberapa perbaikan tetap dibuat pada *Visual Basic* 2008 untuk memperbaiki kekurangan pada *Visual Basic* 2005. *Visual Basic* 2008 *Express* mencakup perbaikan sebagai berikut:

1. Visual desainer *Windows Presentation Foundation* dengan nama kode "Cider"
2. *Debugs* saat *runtime*.
3. Dukungan *IntelliSense* yang lebih baik, seperti:
    a. Perbaikan kesalahan ejaan umum.
    b. Memperbaiki sintaks yang tidak valid.
    c. Menyediakan saran untuk nama kelas ketika kelas-kelas tertentu tidak ditemukan

Express Editions (2005 dan 2008) kebanyakan memiliki keterbatasan yang sama sebagai berikut:

1. Tidak ada IDE dukungan untuk *database* selain SQL *Server Express* dan *Microsoft Access*.
2. Tidak ada dukungan untuk Aplikasi *Web* dengan ASP.NET (ini justru bisa dilakukan dengan *Visual Web Developer Express*, meskipun non-*Express* versi *Visual Studio* memungkinkan baik *web* dan jendela-jendela aplikasi dari IDE yang sama).
3. Tidak ada dukungan untuk mengembangkan perangkat *mobile* (tidak ada *template* atau *emulator*).

4. Tidak ada *Crystal Reports*
5. Lebih sedikit proyek *template* (misalnya *Windows* jasa *template*, *template Excel Workbook*).
6. Terbatas pilihan untuk *debugging* dan *breakpoints*.
7. Tidak ada dukungan untuk membuat *Windows Services* (Dapat diperoleh melalui proyek *download template*).
8. Tidak ada dukungan untuk OpenMP.