BAB II

LANDASAN TEORI

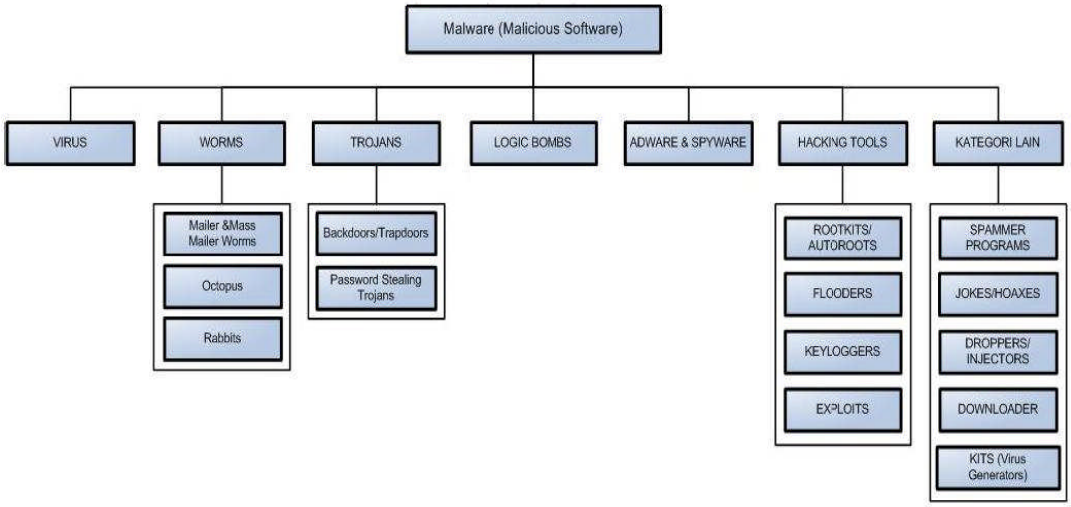
2.1 Sekilas Tentang Analisis Virus

Analisis Virus merupakan tahapan awal dalam memecahkan permasalahan yang dihadapi, karena tahap inilah kita dapat mengetahui gambaran riil dalam perancangan aplikasi dalam penanggulangan Virus Worm.win32/Amburadul

2.2 Tinjauan Umum *Malware*

Seiring pesatnya kemajuan teknologi informasi khususnya di bidang teknologi komputer dan jaringan, ternyata tidak hanya membawa manfaat yang besar bagi penggunanya tetapi juga minimbulkan dampak negatif dengan dibuatnya kode program yang berbahaya, mulai dari ancaman langsung para *Craker* atau *Hacker* jahat hingga acaman yang dilakukan melalui program yang disebut *Malware (malicious code)*. Suatu program atau *script* apapun yang bersifat merusak atau merugikan dapat katagorikan sebagai *Malware* termasuk Virus komputer, *Worm* atau *Trojan horse* dan *Malware* lainnya.

Pada dasarnya *Malware* didefinisikan dengan sebuah perangkat lunak yang didesain untuk melakukan *infiltrasi* atau merusak suatu sistem komputer, tanpa seizin pemiliknya, terjadinya tumpang tindih antara spesifikasi teknis tiap kelas dan *sub* kelasnya selalu ditemukan pada klasifikasi *Malware*. Namun secara umum, menurut Peter Szor *Malware* dapat diklasifikasikan seperti gambar berikut ini, yaitu :



Gambar 2.1 Klasifikasi *Malware*

Dari klasifikasi di atas, karena memiliki aksi yang hampir sama, terdapat dua tipe *Malware* yang agak sulit dibedakan yaitu, Virus dan *Worm*. Untuk mempermudah melihat perbedaan kedua *Malware* tersebut, dapat ditinjau dari sejarah perkembangannya, definisi dan teknik penyebarannya.

2.2.1 Sejarah dan Definisi

1. Sejarah dan Definisi Virus

Virus komputer tercipta bersamaan dengan komputer. Dimulai tahun 1949, ketika salah seorang pencipta komputer, John von Neumann, yang menciptakan *Electronic Discrete Variable Automatic Computer (EDVAC),* memaparkan suatu makalah yang berisi program yang dapat menyebar dengan sendirinya dengan berjudul *"Theory and Organization of Complicated Automata"* [http://www.azmifauzan.web.id] atau *" teori self altering automata"* [http://www.ilmukomputer.com] yang merupakan hasil riset dari para ahli matematika.

Perkembangan selanjutnya, terjadi di Amerika Serikat pada tahun 1984 ketika Fred Cohen mempublikasikan teori "*self replicating program"*. Yang menjadi landasan penamaan Virus. Penamaan Virus Komputer dinamakan *"Viruses"* karena memiliki persamaan mendasar dengan Virus pada istilah kedokteran *(Biological Viruses*) [http://www.e-dukasi.net], hingga pada tahun berikutnya Fred Cohen menyerahkan desertasinya *"Computer Viruses – Theory and Experiments"* yang menyebabkan Virus berkembang dengan cepat dikalangan pengguna komputer [http://www.azmifauzan.web.id].

Fred Cohen [http://www.ilmukomputer.com] mendefiniskan Virus sebagai berikut, yaitu: " *Viruses A program that can infect other programs by modifying them to include a slighty altered copy of it self. A Virus can spread throughout a computer system or network using the authorization of every user using it to infect their programs. Every programs that gets infected can also act as a Virus that infection grows"* artinya Virus merupakan program komputer yang dapat menggandakan dirinya sendiri kemudian menyebar dengan cara menyisipkan salinan dirinya ke dalam program maupun dokumen lain. Virus komputer dapat menyebar pada komputer atau jaringan dengan cara membuat duplikat dari dirinya sendiri tanpa sepengetahuan dari pengguna komputer tersebut.

Virus memiliki efek yang berbahaya dari pada program lainnya, mulai dari menampilkan pesan, mencuri data atau bahkan dapat mengontrol komputer dari jarak jauh. Virus komputer pun dapat merusak (misalnya dengan merusak data pada dokumen), membuat komputer menjadi lambat hingga memformat seluruh isi *harddisk*.

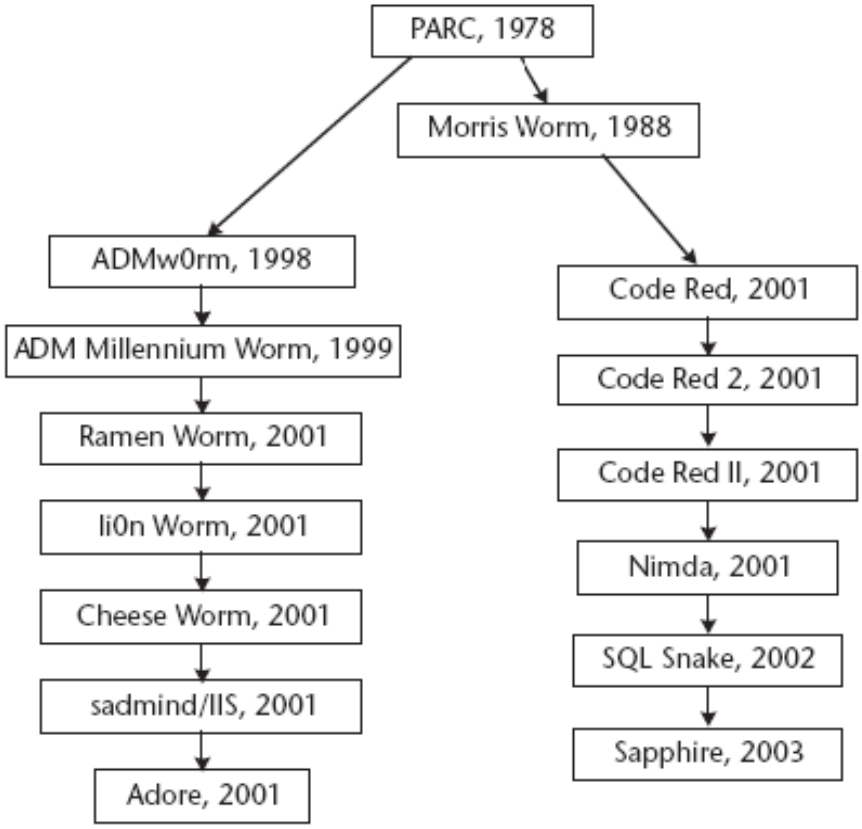
Seiring dengan perkembangan teknik pemrogramannya, terdapat beberapa bentuk Virus yang tidak sesuai dengan definisi tersebut. Sebagai contoh, suatu Virus yang sering disebut *Companion Virus* yang memiliki kemampuan menggandakan diri tanpa mengubah program yang diinfeksinya.

Sehingga menurut Peter Szor, definisi yang lebih akurat untuk Virus pada saat ini adalah, suatu program yang secara berulang *(recursively)* dan dengan tegas *(explicitly)* menggandakan suatu *versi* dirinya sebagai kemungkinan untuk *berevolusi*.

2. Sejarah dan Definisi *Worm*

Sejarah *Worm* mulai ada dan dikenal sejak awal internet mulai dipublikasikan. *Worm* yang mula-mula diciptakan oleh Robert T. Morris tahun 1984, dikategorikan sebagai program yang berbahaya (*Malicious software*/*Malware*) yang mirip dengan Virus dan menyebar melalui jaringan komputer. Keunggulan *Worm* ini jika dibandingkan dengan Virus adalah kecepatannya yang tinggi dalam menginfeksi komputer dalam jaringan (*internet*).

Untuk mengenal sejarah awal keberadaan *Worm* ini, secara umum dapat dilihat pada gambar kronologis kemunculan *Worm* di bawah ini, yaitu :



Gambar 2.2 Kronologis *Worm*

Secara singkat dapat diketahui bahwa sejak awal penciptaan *Worm* di *Palo Alto Research Center (PARC)* oleh Robert T. Morris, berdasarkan lingkungan sistem operasinya, berkembang menjadi dua kategori, yaitu : *Worm* yang berjalan dengan target sistem operasi *UNIX* dan *Worm* yang pada sistem operasi *Microsoft Windows*. *Worm* pada sistem *UNIX* dan variannya terdiri dari *ADMW0rm, ADM Millenium, Ramen, li0n, Cheese, sadmind/IIS, dan Adore. 7*, sedangkan *Worm* yang berjalan pada sistem

Microsoft Windows terdiri dari *Code Red 1, Code Red 2, Code Red II, Nimda, SQL Snake dan Sapphire.*

Sedangkan definisi formal untuk *Worm* menurut Robert T. Morris adalah suatu program yang berpindah dari satu komputer ke komputer yang lain tanpa mengikatkan dirinya *(attach it selft)* pada sistem operasi komputer yang diinfeksinya. Sejalan dengan perkembangannya, definsi *Worm* tersebut sudah tidak begitu tepat. Beberapa *Worm* sering menggunakan teknik untuk menyembunyikan kehadirannya dengan melakukan *instalasi* atau *modifikasi* sistem. Sehingga definisi yang lebih tepat menurut Jose Nazario adalah suatu agen penginfeksi yang otonom dan *independen* dalam bereplikasi, serta memiliki kemampuan dalam menginfeksi sistem *host* baru melalui fasilitas jaringan.

2.2.2 Teknik Penyebaran Virus dan *Worm*

Banyak teknik yang dipergunakan untuk menyebarkan masing-masing klasifikasi *Malware* tersebut, namun secara umum antara Virus dan *Worm* memiliki persamaan yang mendasar, yaitu : Virus memerlukan campur tangan pengguna dalam penyebarannya, misalnya dalam melakukan klik ganda pada *file* yang terinfeksi, Sedangkan *Worm* dapat secara otomatis menyebar dengan tanpa atau sedikit campur tangan dari penggunannya. Misalnya dengan membuka *file* lampiran *e-mail* yang terinfeksi *Worm*, membuka *drive* media penyimpanan seperti *flashdisk, disket* atupun *DVD/CD Rom* yang terinfeksi *Worm* maka satu sistem yang terhubung akan segera terinfeksi.

2.2.3 Penyebaran Virus dan *Worm* Di Indonesia

Perkembangan penyebaran *Worm* di Indonesia, pada awalnya lebih banyak didominasi oleh *Worm* yang berasal dari luar negeri. Namun pada bulan Oktober 2005, dominasi ini mulai runtuh dengan menyebarnya *Worm* Rontokbro/Brontok (Email-Worm.Win32.Brontok.a). Menurut vaksin.com [http://www.vaksin.com], *Worm* tersebut selama bulan Agustus 2005 sampai dengan November 2005, menempati urutan pertama sebagai *Worm* yang paling banyak menyebar di Indonesia.

Tidak hanya itu, Rontokbro tersebut juga telah menyebar pada beberapa negara seperti Amerika, Polandia, Spanyol, Jepang, Vietnam, Belanda, Hungaria, Swedia, Peru, Rusia dan Israel. Selain itu Aksi *Worm* ini juga telah melumpuhkan situs www.kaskus.com dan menyerang situs 17tahun.com, http://israel.gov.il dan sebuah situs pribadi *(blog)* www.fajarweb.com seperti yang diberitakan. [http://www.compactbyte.com].

Seiring dengan perkembangan teknik pemrograman dan cakupan lingkungan pemrogramannya, maka perkembangan Virus dan *Worm* di Indonesia semakin berkembang pesat, terbukti menurut virusindonesia.com [http://virusindonesia.com], ada kurang lebih 3 Virus baru yang bermunculan yang memiliki beragam teknik pemrograman dan teknik penyebarannya sehingga sampai sekarang Virus lokal *made in* Indonesia terkenal sulit untuk dibasmi karena kebanyakan menggabungkan teknik antara Virus dan *Worm* seperti Virus.Worm.Win32/Amburadul.

2.3 Tinjauan Umum *Registry Windows*

2.3.1 Sejarah dan Definisi *Registry Windows*

Registry adalah suatu *set file* data yang dipakai *Windows* untuk mengontrol perangkat keras, perangkat lunak, lingkungan *user* dan *interface*. Dulunya pekerjaan itu dilakukan oleh *file -file* yang berjenis *INI*. Sebelum menggunakan *file INI*, pada awalnya *Windows* memakai dua *file system configurasi* yaitu *config.sys* dan *autoexec.bat*, karena kemampuannya sangat terbatas kemudian *Microsoft* membuat dua *file* lagi yaitu *win.ini* dan *system.ini*. *system.ini* dipakai untuk mengendalikan *hardware* sedangkan *win.ini* mengendalikan desktop dan aplikasi. Namun karena kemampuan *win.ini* dan *system.ini* yang serba terbatas, maka muncullah konsep *database* yang menangani permasalahan tersebut yang dikenal dengan nama *registry* [http://id.wikipedia.org].

Registry merupakan informasi *database* yang berada pada sistem operasi *Windows 95* ke atas. *Database* ini menyimpan pengaturan yang kita lakukan pada komputer dan setiap kali *Windows* beroperasi, *registry* akan mengisinya dengan beberapa *variabel* atau *value* yang mengatur sistem komputer. Jadi Secara konsepsional *registry* adalah *database* terpusat yang dipakai *Windows* dan aplikasi yang berjalan di atasnya untuk menyimpan informasi konfigurasinya. Informasi tersebut berisi *system hardware, software* dan komunikasi yang dapat dirubah sesuai aplikasi yang dijalankan.

Untuk menampilkan *registry* dalam *database* tergabung, maka diperlukan *regedit* yang merupakan *editor* dari *registry* yang digunakan untuk mengamati, mengubah, atau menentukan *setting* sistem, yang berisi informasi tentang bagaimana komputer berjalan. *Database* tersebut disimpan dalam dua *file* yang terpisah yaitu *system.dat* dan *user.dat*. *system.dat* digunakan untuk menyimpan data mesin komputer sedangkan *user.dat* digunakan untuk menyimpan data pemakai.

Pada saat komputer *booting*, *Windows* pertama kali akan mengakses *system.dat* yang akan menyimpan data-data khusus yang berhubungan dengan sistem komputer. Sedangkan *user.dat* ini akan menspesifikkan informasi pemakai seperti nama *user, password, control panel, setting aplikasi, shutdown, log of,* ataupun *switch user* dan lain-lain.

2.3.2 Struktur *Registry Windows*

Dalam *Windows XP, registry* terdiri dari lima kategori informasi yang disebut sebagai *root keys*, kunci induk seperti ini berisi data yang berisi sistem komputer yang berhubungan erat dengan perangkat keras komputer. *Root keys* tersebut berisi *value, key* dan juga *subkey* yang memudahkan untuk dikelola, diatur dan dimodifikasi.

Root keys biasa disebut *handle keys (HKEY)* yang merupakan sekelompok masukan yang digunakan untuk melihat dan mengubah informasi dengan mudah [http://www.id.wikipedia.org], berikut keterangan dari lima *handle keys*:

HKEY_CURRENT_USER

Menangani satu informasi *user* yang aktif pada sistem operasi *Windows*, ada beberapa pengaturan yang disimpan di sini. Informasi ini dikenal sebagai suatu profil pemakai.

HKEY_USERS

Menangani semua informasi *user* yang aktif serta menangani *setting control panel* pada sistem operasi *Windows*. *HKEY_CURRENT_USER* adalah suatu *subkey HKEY_USERS*. Informasi yang ada pada *HKEY_USER* didapatkan dari *file user.dat*

HKEY_LOCAL_MACHINE

Memuat informasi tentang *hardware, software,* dan komunikasi yang dipakai pada komputer tersebut. Informasi yang ditampilkan diperoleh dari *file system.dat*

HKEY_CLASSES_ROOT

Adalah suatu *subkey HKEY_LOCAL_MACHINE\SOFTWARE* yang menangani semua jenis *file* yang ada di *Windows* dan menyimpan *setting software.*

HKEY_CURRENT_CONFIG

Berisi informasi tentang profil perangkat keras yang digunakan oleh komputer lokal pada *system startup*, dan mencatat konfigurasi sistem yang ada. Isinya sama dengan *HKEY_LOCAL_MACHINE\CONFIG*. Dari sini tampak bahwa *HKEY_CLASSES_ROOT* dan *HKEY_CURREN_CONFIG* adalah salinan dari sebagian data di *HKEY_LOCAL_MACHINE* dan *HKEY_CURRENT_USER* adalah salinan dari sebagian data di *HKEY_USER*.

Hal tersebut digunakan untuk kemudahan dalam proses *editing* dan penanganan data. Pada waktu komputer *booting* maka informasi untuk *HKEY_CLASS_ROOT* dan *HKEY_CURRENT_CONFIG* akan dipetakan dari *HKEY_LOCAL_MACHINE* sehingga siap untuk dilihat dan dioperasikan.

2.3.3 Perubahan Pada *Registry Windows*

Perubahan pada *registry* dapat terjadi secara *otomatis* dan secara *manual*, perubahan *registry* secara *otomatis* dapat disebabkan karena *instal software* baru atau masuknya Virus komputer yang melakukan perubahan pada *registry Windows*. Bila perubahan dilakukan secara *manual*, maka bisa langsung masuk ke menu *registry* dengan mengklik *Run – regedit* atau dengan menggunakan *visual basic script* dengan menuliskan perintah *RegDeleteKey*, *RegCreateKey* dan *ModifiBinariData*.

2.4 Konsep Penanggulangan Virus

Aplikasi atau program penanggulangan Virus (*Anti Virus*) adalah program yang dapat menangkap, mengidentifikasi dan menghancurkan Virus. Program *Anti Virus* terbagi 3 [http://www.virologi.info], yaitu:

1. *Fix*

   Fix adalah sebuah program yang dapat mendeteksi dan menghancurkan hanya satu Virus. Harus dijalankan terlebih dahulu kemudian program akan mencari dan menghapus Virus tertentu.

2. *Antidot*

   Antidot adalah sebuah program yang dapat menangkap, mendeteksi beberapa jenis Virus dan menghapusnya. Sama seperti program *fix,* harus dijalankan terlebih dahulu kemudian akan mencari *file* yang terinfeksi maupun *file* Virus tertentu.

3. Anti Virus

   Anti Virus adalah sebuah program yang dapat menangkap, mendeteksi dan menghapus banyak jenis Virus. Dapat otomatis menangkap *file* yang terinfeksi dan menghapusnya. Contohnya *Kaspersky Anti Virus* dari kaspersky.com [http://www.kaspersky-indonesia.com].

2.4.1 Cara kerja *Anti Virus*

Cara kerja program – program penanggulangn Virus ini kebalikan dari program Virus itu sendiri, yaitu mendeteksi apakah *file* terkena Virus atau tidak, jika ya maka *Anti Virus* akan menindaklanjuti *file* tersebut yaitu dengan mengkarantina, menghapus, atau memperbaikinya [http://www.arcle.net].

Secara umum cara kerja *Anti Virus* harus memiliki 3 komponen penting, yaitu :

1. List Virus atau Daftar Virus (*database* Virus)

   Daftar Virus yang dimaksudkan disini adalah *database* Virus, yang berisi nama Virus, sidik jari Virus dan jenis Virus.

2. Aplikasi proses (aplikasi *search enggine*)

   Maksudnya disini adalah aplikasi pencari sidik jari atau pemusnah Virus.

3. *User*

   Maksudnya disini adalah *Brainware* atau orang yang mengoperasikan program *Anti Virus*.

2.5 Konsep *Algoritma CRC32 [ftp:\\student.if.itb.ac.id]*

Cyclic Redundancy Check (CRC) adalah salah satu fungsi hash yang dikembangkan untuk mendeteksi kerusakan data dalam proses *transmisi* ataupun penyimpanan. *CRC* menghasilkan suatu *checksum* yaitu suatu nilai dihasilkan dari fungsi hash-nya, dimana nilai inilah yang nantinya digunakan untuk mendeteksi *error* pada *transmisi* ataupun penyimpanan data. Nilai *CRC* dihitung dan digabungkan sebelum dilakukan transmisi data atau penyimpanan, dan kemudian penerima akan melakukan *verifikas*i apakah data yang diterima tidak mengalami perubahan ataupun kerusakan. *CRC* cukup terkenal karena mudah diterapkan dalam *hardware*, dan mudah dilakukan analisis secara matematika. Prinsip utama yang digunakan adalah dengan melakukan pembagian *polynomialial* dengan mengabaikan *bit-bit carry*. Cara yang biasa digunakan adalah dengan menggunakan tabel *CRC* yang nilainya telah dihitung sebelumnya.

2.5.1 Cara Kerja (Prinsip *CRC*)

Pada intinya dalam proses penghitungan *CRC*, kita menganggap suatu *file* yang kita proses sebagai suatu *string* yang besar, yang terdiri dari *bit-bit*, dan kita operasikan sebagai suatu bilangan *polynomial* yang sangat besar. Untuk menghitung nilai *CRC*, kita membagi bilangan *polynomial*, sebagai *representasi* dari *file*, dengan suatu bilangan *polynomial* kecil yang sudah terdefinisi untuk jenis *varian CRC* tertentu.

Nilai *CRC* adalah sisa hasil bagi tersebut, yang biasa disebut dengan *cheksum*. Setiap operasi pembagian pasti menghasilkan suatu sisa hasil bagi (meskipun bernilai 0), tetapi ada perbedaan dalam melakukan pembagian pada penghitungan *CRC* ini. Secara umum (prinsip *Aljabar* biasa), pembagian dapat kita lakukan dengan mengurangi suatu bilangan dengan pembaginya secara terus-menerus sampai menghasilkan suatu sisa hasil bagi (yang lebih kecil dari bilangan pembagi). Dari nilai hasil bagi, sisa hasil bagi, dan bilangan pembagi kita bisa mendapat bilangan yang dibagi dengan mengalikan bilangan pembagi dengan hasil bagi dan menambah dengan sisa hasil bagi. Dalam penghitungan *CRC*, operasi pengurangan dan penjumlahan dilakukan dengan mengabaikan setiap nilai *carry* yang didapat. Tentu saja hal ini juga akan berpengaruh pada proses pembagian yang menjadi dasar utama dalam melakukan penghitungan nilai *CRC*. Operasi dalam *CRC* juga hanya melibatkan nilai 0 dan 1, karena secara umum kita beropersi dalam *level bit*. Contoh penghitungan dalam *CRC* adalah sebagai berikut:

```
(1)   1101          (2)   1010          1010
      1010 -              1111 +        1111 -
      --------            ---------     ---------
      0011                0101          0101
```

Pada contoh tersebut, operasi pertama (1) adalah operasi yang umum digunakan dalam operasi *Aljabar*, yaitu dengan menghitung nilai *carry* yang dihasilkan, sedangkan operasi kedua (2) adalah operasi dasar yang akan kita gunakan dalam proses penghitungan nilai *CRC*. Nilai *carry* diabaikan, sehingga operasi pengurangan dan penambahan akan menghasilkan suatu nilai

yang sama. Kedua operasi ini bisa dilakukan dengan melakukan penjumlahan dan *dimodulo 2*, atau dalam dunia pemrograman lebih dikenal dengan operasi *XOR* (istilah ini yang akan lebih sering kita gunakan untuk menyebut penjumlahan pada operasi penghitungan *CRC*). Pada proses pembagian yang dilakukan, akan tampak sekali bedanya karena pengurangan yang dilakukan dilakukan seperti melakukan penambahan. Nilai hasil bagi diabaikan, karena kita tidak menggunakannya, jadi hanya sisa hasil bagi *(remainder)* yang kita perhatikan. Dan *remainder* inilah yang akan menjadi dasar bagi nilai *CRC* yang dihasilkan.

2.5.2 Penghitungan *CRC* secara *Aljabar*

Untuk melakukan penghitungan *CRC* terhadap suatu data, maka yang pertama kita perlukan adalah suatu bilangan *polynomial* yang akan menjadi pembagi dari data yang akan kita olah (kita sebut sebagai '*poly*'). Kita juga menghitung lebar suatu *poly*, yang merupakan posisi dari *bit* tertinggi. Misalkan kita sebut lebar dari *poly* ini adalah W, maka jika kita mempunyai *poly* 1001, maka W *poly* tersebut adalah 3, bukan 4. *Bit* tertinggi ini harus kita pastikan bernilai 1. mengetahui nilai W secara tepat sangat penting karena akan berpengaruh pada jenis *CRC* yang kita gunakan (*CRC*16,*CRC*32,dll). Data yang kita olah mungkin saja hanya beberapa *bit* saja, lebih kecil dari nilai *poly* yang kita gunakan. Hal ini akan menyebabkan kita tidak mengolah semua nilai *poly* yang telah ditentukan.

Untuk mengatasi hal tersebut, dalam penghitungan dasar secara *Aljabar*, kita menambah suatu *string bit* sepanjang W pada data yang akan kita olah, untuk menjamin keseluruhan data kita proses dengan benar.

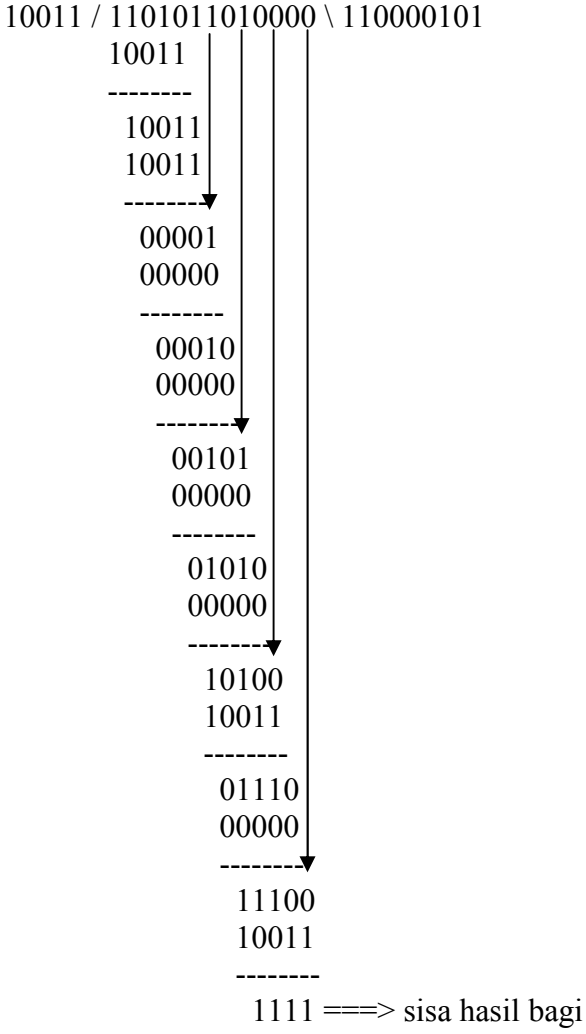
Contoh penghitungan kita menjadi sebagai berikut:

Diketahui : *Poly* = 10011
(width W=4)

Ditanyakan : Ceksum *CRC*32

Jawab : *Bitstring* + W zeros = 110101101 + 0000

Contoh pembagian yang dilakukan:



Nilai *remainder* inilah yang menjadi nilai *CRC*. Pada proses pembagian tersebut, kita mendapat hal penting yang perlu kita perhatikan dalam penghitungan secara *Aljabar* ini adalah kita tidak perlu melakukan operasi *XOR* ketika *bit* tertinggi bernilai 0, tapi kita hanya melakukan penggeseran (shift) sampai didapat *bit* tertinggi yang bernilai 1. Hal ini akan sedikit mempermudah dan mempercepat operasi *Aljabar* kita. Secara notasi *Aljabar* bisa kita tuliskan sebagai berikut:

$$a(x).xN = b(x).p(x)+r(x)$$

keterangan :

a(x) = Bilangan *polynomial* yang me*representasi*kan data.

xN = Nilai 0 sebanyak W

b(x) = hasil bagi yang didapat

p(x) = *poly*

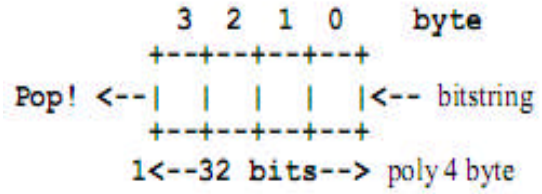
r(x) = sisa hasil bagi, nilai *CRC*

karena nilai *CRC* adalah sisa hasil bagi, maka untuk mengecek integritas data dapat dilakukan dengan beberapa cara, diantaranya:

a. Kita hitung nilai *CRC* dari data yang asli, lalu kita cocokkan dengan nilai *CRC* yang disimpan (di *append* dengan data). Data yang asli mudah kita dapatkan karena nilai *CRC* sepanjang N-1.
b. Data yang asli kita tambah dengan nilai *CRC*, lalu kita bagi dengan nilai *poly*, maka sisa hasil bagi adalah 0 jika data benar.

2.5.3 Pendekatan Tabel *CRC*

Penghitungan nilai *CRC* yang berbasis *bit* seperti pada pendekatan *Aljabar* diatas akan sangat lama dan tidak efficient. Kita bisa memperbaiki cara yang kita gunakan jika kita dapat melakukan operasi dengan basis *byte*, bukannya *bit*. *Poly* yang kita gunakan pun akan kita operasikan dalam bentuk *byte*, sehingga harus mempunyai panjang kelipatan 8 *bit* (*byte*). Dalam *CRC*32, berarti kita gunakan *poly* 32 *bit* (4 *byte*), akan tampak sebagai berikut:



Gambar 2.3 Pendekatan Tabel *CRC*

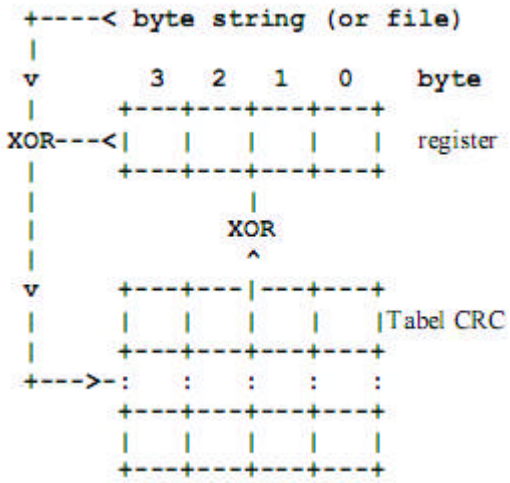
4 (empat) ruang kosong pada *ilustrasi* diatas menggambarkan *register* yang akan kita gunakan untuk menampung hasil *CRC* sementara (pada proses pembagian yang melibatkan operasi *XOR*). Proses yang dilakukan pada *register* ini adalah :

a. Kita masukkan *bitstring* (data) ke dalam *register* dari arah ke kanan (shift per *byte*) setiap saat *register* tidak terisi penuh.
b. Jika *register* sudah penuh (berisi 4 *byte*), maka kita geser satu *byte* ke arah kiri (keluar *register*), dan kita isi *register* dari arah kanan dengan 1 *byte* dari *bitstring*
c. Jika *register* yang digeser keluar punya nilai 1, maka kita melakukan operasi *XOR*

2.5.4 *Algoritma Direct Table*

Algoritma yang kita gunakan pada pendekatan diatas, dapat kita *optimasi* lagi jika kita bisa mencegah satu *byte* dari *bytestring* melewati setiap bagian dari *register* (konsekuensi dari penggeseran 1 *byte*) sebelumnya akhirnya diproses untuk mengacu nilai pada tabel *CRC*. Pada *Algoritma direct table*, kita melakukan operasi *XOR* secara langsung sebuah *byte* dari *bytestring* dengan *byte* yang dishift out dari *register* (*register* tidak diisi dengan *byte* dari *bytestring*). Hasil *XOR* ini akan menjadi *index* pada tabel, yang nilainya akan dilakukan *XOR* dengan isi dari *register* .

Untuk melakukan *Algoritma* ini, kita menginisiasi *register* dengan suatu nilai, kita sebut *INITXOR* (biasanya berupa bilangan dalam *hexa FFFFFFFF*). Kita juga tidak melakukan penggeseran terhadap nilai *bytestring* ke dalam *register* . Nilai *register* hanya didapat pada setiap operasi *XOR* yang dilakukan terhadap isi awal *register* dengan isi tabel *CRC*. Dengan adanya *INITXOR* yang memberikan nilai awal dari *register* , kita tidak membutuhkan bilangan 0 sejumlah W yang harus kita append pada *bytestring*, dan sebagai konsekuensinya pada akhir perhitungan nilai *CRC (checksum)* yang kita dapatkan juga akan kita *XOR* dengan suatu bilangan hexa, yang kita kita sebut *FINALXOR* dan biasanya berupa bilangan *FFFFFFFF*.



Gambar 2.4 Pendekatan *Algoritma* dengan Tabel *CRC*

keterangan:

- Ambil satu *byte* dari *register* , misal *b1*
- *XOR b1* dengan satu *byte* dari data, misalkan hasilnya *b2*.
- Gunakan *b2* sebagai index untuk mendapatkan nilai pada tabel *CRC*, misalkan nilai yang didapat *b3*.
- Lakukan *XOR b3* dengan isi *register* , dan hasilnya dimasukkan dalam *register* menggantikan isi *register* awal.

Untuk menerapkan konsep dan *Algoritma* diatas yang menggunakan tabel *CRC*, kita memerlukan Standar *CRC*32 sebagai berikut:

Nama : *"CRC-32"*

Panjang : *32*

Poly : *04C11DB7* ➔ *Hexa* Atau $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ ➔ *Polynomialial*

Initial (INITXOR) : *FFFFFFFF* *FINALXOR* : *FFFFFFFF*

Reflected : *True*