IMPLEMENTASI PROTOTIPE PERMINTAAN SERTIFIKAT X.509 CARDHOLDER PADA PROTOKOL SECURE ELECTRONIC TRANSACTION

HARIS

1293000132

13 Agustus 1999

Daftar Isi

Daftar Gambar

Daftar Tabel

Bab 1

PENDAHULUAN

1.1 LATAR BELAKANG MASALAH

Perkembangan internet sangat cepat. Masyarakat menggunakan internet untuk berdagang. Dengan adanya perdagangan di internet, muncul pula kemungkinan adanya pencurian. Karena itu dikembangkan metode-metode untuk melangsungkan perdagangan di internet secara aman dan cepat. Metode yang ada sekarang ini di antaranya adalah Secure Socket Layer (SSL), dan yang paling baru Secure Electronic Transaction (SET).

Di lingkungan Fakultas Ilmu Komputer Universitas Indonesia (Fasilkom UI), telah dijalankan implementasi SET yang dilakukan oleh *Special Interest Group Digital Security* (SIG-DS). Karena protokol SET ini sangat kompleks maka implementasi dibagi ke beberapa orang. Penulis melakukan implementasi di bagian komunikasi antara *Cardholder* dengan *Certificate Authority*.

Penelitian diawali dengan membaca dasar pengetahuan tentang SET. Referensi yang dipakai berasal dari dokumentasi standar SET yang dibuat oleh Visa dan Mastercard. Kemudian ternyata implementasi harus dilakukan mengikuti berbagai macam standar seperti ASN.1, BER, dan DER. Barulah implementasi dilakukan, sambil merancang metode pengujiannya.

Hasil penelitian berupa suatu aplikasi yang melakukan komunikasi dalam format yang ditentukan SET antara *Cardholder* dengan *Certificate Authority*. Pemakaian Internet semakin berkembang dari penggunaannya sebagai sarana komunikasi global menjadi sarana perdagangan. Suatu perusahaan penjual barang atau jasa tidak cukup hanya menempatkan informasi mengenai profil perusahaan saja, tetapi juga harus bisa menyediakan layanan pembelian produknya melalui Internet.

Perkembangan perdagangan elektronik sudah mencapai titik kritis. Konsumen menginginkan akses yang aman ke toko elektronik. Pedagang menginginkan metode yang sederhana dan murah untuk bertransaksi secara elektronik. Institusi keuangan menginginkan tingkat tertentu dalam perdagangan bagi penyedia perangkat lunak untuk menjamin produk berkualitas dalam harga kompetitif. Pe-

megang merk kartu pembayaran harus bisa membedakan berbagai perdagangan elektronik tanpa akibat yang signifikan pada infrastruktur yang ada.

Seiring dengan perkembangan perdagangan di Internet tersebut, diperlukan protokol perdagangan yang aman di Internet. Seperti sudah diketahui, Internet adalah jalur komunikasi publik yang terbuka dan sangat tidak aman. Ada dua protokol yang saat ini dianggap paling aman dan cukup luas pemakaiannya, yaitu *Secure Socket Layer* (SSL) dan *Secure Electronic Transaction* (SET).

Kedua protokol tersebut menggunakan konsep kriptografi, yaitu penyandian pesan sehingga hanya pihak penerima pesan yang berhak yang dapat membaca pesan yang dikirimkan. Kelebihan SET dibanding SSL adalah penggunaan sertifikat digital untuk mengidentifikasi pihak-pihak yang terlibat dalam proses perdagangan. Karena SET menggunakan sertifikat digital, pihak pembeli yakin bahwa pihak penjual adalah penjual yang sebenarnya dan tidak dipalsukan oleh orang lain. Penjual juga yakin bahwa pihak pembeli benar-benar ada dan mampu membeli barang yang dia ditawarkan.

1.2 TUJUAN PENELITIAN

Penelitian ini dilakukan untuk menghasilkan prototipe aplikasi permintaan sertifikat digital untuk *Cardholder* dan aplikasi *Certificate Authority* untuk menjawab permintaan sertifikat digital untuk *Cardholder*.

1.3 BATASAN MASALAH

Penelitian yang dilakukan penulis terbatas pada pembuatan prototipe aplikasi pemrosesan permintaan sertifikat konsumen (*Cardholder*/CH) oleh *Certificate Authority* (CA). Implementasi kriptografi tidak dilakukan karena memakai librari fungsi kriptografi yang disediakan pihak lain, dalam hal ini *library* yang dibuat oleh *Australian Business Assocation* (ABA).

Implementasi prototipe aplikasi ini meliputi pembuatan pesan-pesan yang berhubungan yang dispesifikasikan SET dalam [VIMA97a], [VIMA97b], dan [VIMA97c], dan obyek-obyek yang terlibat di dalamnya. Aplikasi ini tidak melibatkan antar muka yang sesuai dengan spesifikasi SET, karena hanya bertujuan untuk menguji pertukaran pesan.

Implementasi operasi kriptografi terbatas pada operasi yang diperlukan untuk membuat pesan SET itu, tidak meliputi validasi rantai sertifikat seperti disebutkan pada seksi 2.7, atau validasi BrandCRLIdentifier.

1.4 METODOLOGI PENELITIAN

Penelitian dimulai dengan mempelajari spesifikasi SET. Spesifikasi SET terdapat dalam buku yang dipublikasikan secara elektronik lewat URL http://www.setco.org, dan dibuat oleh Visa dan Mastercard.

Dari spesifikasi SET diketahui bahwa SET mendefinisikan pesan dalam format ASN.1 (*Abstract Syntax Notation One*) dan dikodekan dalam aturan DER (*Distinguished Encoding Rule*). Untuk itu perlu pengetahuan tentang spesifikasi format ASN.1 dan aturan DER. Format ASN.1 dan aturan DER ini dipelajari dari [KALI93] dan [LARM99].

Spesifikasi SET melibatkan teknik kriptografi. Di awal penelitian sudah ditekankan oleh koordinator penelitian, Arrianto Muktiwibowo, bahwa masalah kriptografi bukanlah hal yang utama dalam penelitian ini, karena itu untuk kriptografi sebaiknya menggunakan *library* kriptografi yang sudah diimplementasikan pihak lain. Selain itu bahasa pemrograman menggunakan Java, karena program Java bisa dijalankan pada berbagai macam *platform*, selama *platform* itu memiliki *Java Virtual Machine* (JVM). Dalam pencarian *library* yang cocok ditemukan *library* kriptografi yang cocok untuk dipakai dalam penelitian, yaitu *library* kriptografi dari *Australian Business Association* (ABA). Lisensinya gratis, memudahkan untuk pemakaian dalam penelitian.

Penelitian kemudian dilanjutkan dengan uji coba kriptografi dan pembuatan sertifikat. Setelah itu baru melakukan implementasi prototipe protokol komunikasi permintaan pembuatan sertifikat antara *Cardholder* dengan *Certificate Authority*.

1.5 SISTEMATIKA PENULISAN

Laporan tugas akhir ini disusun dengan sistematika sebagai berikut:

- Bab 1, berisi latar belakang tugas akhir, rumusan permasalahan dan ruang lingkup, tujuan penelitian, metode penelitian, serta penjelasan sistematika laporan tugas akhir.
- Bab 2, berisi tentang dasar-dasar yang diperlukan untuk memahami aturan yang digunakan dalam protokol SET.
- Bab 3, berisi spesifikasi protokol SET mengenai proses permintaan sertifikat *Cardholder*.
- Bab 4, berisi analisa spesifikasi dan rancangan implementasi penelitian.
- Bab 6, berisi implementasi yang dilakukan.
- Bab 6, menganalisa dan evaluasi hasil penelitian yang telah dilakukan.
- Bab 7, berisi kesimpulan tugas akhir serta saran-saran untuk pengembangan selanjutnya.

Bab 2

LANDASAN SET

Bab ini menjelaskan berbagai hal yang perlu diketahui untuk memahami spesifikasi SET.

2.1 KRIPTOGRAFI

Kriptografi (*cryptography*) adalah ilmu dan seni menjaga suatu pesan tetap aman [SCHN96]. Pesan yang asli (tidak disandikan) dinamakan *plaintext* atau *cleartext*. Proses untuk menyembunyikan isi suatu pesan disebut enkripsi (*encryption*). Pesan yang dienkrip disebut *ciphertext*. Proses untuk mengembalikan bentuk *ciphertext* ke *plaintext* disebut dekripsi (*decryption*).

Cryptanalysis adalah ilmu dan seni untuk memecahkan ciphertext, yaitu melihat apa yang disembunyikan. Orang yang melakukan *cryptanalysis* ini disebut *cryptanalysts*, sedangkan pelaku kriptografi disebut *cryptographers*. Ilmu matematika yang mendalami *cryptography* dan *cryptanalysis* adalah *cryptology* dan orang yang mendalaminya disebut *cryptologists*.

2.1.1 Kunci Simetrik

Kunci simetrik (*symmetric key*) adalah suatu kunci yang digunakan untuk menyandikan *plaintext* menjadi *ciphertext*. Kunci simetrik ini berupa suatu bilangan bulat yang kekuatannya didefinisikan dalam jumlah representasi bit-nya. Semakin banyak jumlah representasi bit, semakin kuat kunci itu. Untuk menghasilkan *ciphertext*, seseorang mengenkrip suatu pesan dengan kunci simetrik dalam fungsi $c = E(p, k)$. Untuk mengembalikan *ciphertext* menjadi bentuk *plaintext*-nya, fungsi yang dipakai adalah $p = D(c, k)$. Fungsi yang dipakai sama. Bila kunci yang dipakai berbeda maka *plaintext* yang didapat akan berbeda dengan *plaintext* yang asli. Kriptografi yang menggunakan kunci simetrik ini disebut kriptografi simetrik (*symmetric cryptography*).

2.1.2 Kunci Asimetrik

Kunci asimetrik (*asymmetric key*) adalah suatu pasangan kunci yang digunakan dalam kriptografi kunci publik (*public-key cryptography*). Pasangan kunci ini terdiri atas kunci publik (*public-key*) dan kunci privat (*private-key*). Dalam kriptografi ini enkripsi dilakukan menggunakan salah satu dari pasangan kunci itu, sedangkan dekripsi harus menggunakan pasangannya. Jadi bila enkripsi dilakukan menggunakan kunci publik, maka dekripsi harus menggunakan kunci privat, dengan rumus $c = E(p, Kpr)$ dan $p = D(c, Kpb)$, demikian juga sebaliknya.

Fungsi-fungsi yang dipakai dalam kriptografi ini merupakan fungsi searah (one-way function). Fungsi searah ini maksudnya komputasi dapat dilakukan dengan relatif mudah, tapi bila fungsi dibalik untuk mendapat nilai semula maka komputasi yang dilakukan sangat sulit sehingga membutuhkan waktu yang sangat lama untuk menyelesaikannya.

2.1.3 Fungsi *Hash*

Fungsi *hash* juga merupakan fungsi searah. Fungsi ini adalah sebuah fungsi, matematika atau lainnya, yang membutuhkan sebuah string masukan dengan panjang bervariasi dan mengubahnya ke sebuah string keluaran dengan panjang yang tetap. Setiap keluaran merupakan suatu string yang unik mewakili masukan yang juga unik.

2.2 PENGKODEAN (*ENCODING*)

Pesan-pesan (messages) dalam SET didefinisikan dalam standar ASN.1 (*Abstract Syntax Notation One*) yang dikeluarkan ISO/IEC dan ITU-T. Hal ini untuk mencegah pengkodean yang ambigu agar standar SET dapat diterima luas dan mudah dimengerti. Standar-standar yang mendefinisikan ASN.1 adalah:

- ITU-T Recommendation X.680 (1997) |ISO/IEC 8824-1:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation.
- ITU-T Recommendation X.681 (1997) |ISO/IEC 8824-2:1998, *Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification.*
- ITU-T Recommendation X.682 (1997) |ISO/IEC 8824-3:1998, *Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification.*
- ITU-T Recommendation X.683 (1997) |ISO/IEC 8824-4:1998, *Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specification.*

- ITU-T Recommendation X.690 (1997) |ISO/IEC 8825-1:1998, *Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*
- ITU-T Recommendation X.691 (1997) |ISO/IEC 8825-2:1998, *Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER).*

ASN.1 adalah sebuah bahasa formal untuk mendeskripsikan suatu pesan secara abstrak yang akan dipertukarkan di antara sistem komputer terdistribusi. Sebelumnya ASN.1 digunakan untuk menulis aplikasi, standar nasional dan internasional. Dengan munculnya berbagai perangkat lunak bantu, ASN.1 kini digunakan untuk menghasilkan kode bahasa pemrograman yang membentuk inti berbagai sistem pertukaran pesan.

Aturan pengkodean ASN.1 (*ASN.1 encoding rules*) adalah kumpulan aturan yang digunakan untuk mengubah data yang dispesifikasikan dalam bahasa ASN.1 ke dalam format standar yang bisa diterjemahkan oleh sistem yang memiliki dekoder yang memiliki kumpulan aturan yang sama. Berbagai aturan pengkodean dapat diterapkan pada ASN.1, hal ini tergantung pada perancangan protokol yang menggunakan ASN.1.

Aturan pengkodean ASN.1 yang standar saat ini adalah: BER (*Basic Encoding Rules*), DER (*Distinguished Encoding Rules*), CER (*Canonical Encoding Rules*), dan PER (*Packed Encoding Rules*).

BER dibuat pada awal 1980-an dan digunakan oleh banyak aplikasi, seperti SNMP (*Simple Network Management Protocol*) untuk manajemen internet, MHS (*Message Handling Service*) untuk pertukaran surat elektronik (*electronic mail/e-mail*) dan TSAPI untuk pengaturan interaksi telepon/komputer.

DER adalah bentuk khusus BER untuk aplikasi yang mementingkan segi keamanan. Aplikasi yang menggunakan DER, seperti *electronic commerce*, biasanya melibatkan kriptografi, dan mengharuskan hanya boleh ada satu cara untuk mengkodekan dan menerjemahkan sebuah pesan.

CER adalah bentuk khusus BER yang lain yang mirip dengan DER, tapi diperuntukkan dengan pesan yang sangat besar sedemikian sehingga lebih mudah untuk memulai mengkodekan pesan sebelum seluruh isinya tersedia. CER jarang dipakai, dengan kenyataan dunia industri lebih menyukai memakai DER dalam mengkodekan data untuk pertukaran pesan yang aman.

PER adalah kumpulan aturan pengkodean yang paling baru dan diakui karena penggunaan algoritma yang efisien menghasilkan pengkodean yang lebih cepat dan ringkas dibanding BER. PER digunakan dalam aplikasi yang kekurangan *bandwidth* atau CPU, seperti *air traffic control* dan telekomunikasi audio-visual.

SET menggunakan pengkodean DER. ASN.1 menyediakan definisi yang jelas dan unik tentang isi pesan, sedangkan DER menyediakan pengkodean yang tepat dan menjamin format yang unik untuk masing-masing data terkodekan (*encoded*). Hal ini penting untuk mendukung operasi yang

melibatkan *hash* dan tanda tangan digital. ASN.1 menyertakan sekelompok koleksi tipe data intrinsic yang digunakan untuk mendefinisikan *data field* dan pesan SET tapi bergantung pada batasan dan larangan tambahan. Hal ini akan diperiksa oleh aplikasi perangkat lunak. Sebagai contoh, tipe data IA5String digunakan sebagai tipe data intrinsic ASN.1untuk mendefinisikan beberapa data *field* yang berisi string karakter (contohnya MerOrderNum). Karakter yang diperbolehkan dan didukung IA5String kadang disebut sebagai karakter ASCII. Sebagai tambahan, batasan ukuran dalam *field* ditetapkan (contohnya MerOrderNum tidak boleh lebih dari 25 karakter) dan harus diperiksa oleh semua aplikasi SET.

2.2.1 ASN.1

Dalam ASN.1, suatu tipe data adalah sekumpulan nilai. Untuk tipe tertentu, jumlah nilainya tentu, dan tipe lainnya tidak pasti jumlahnya. Sebuah nilai dalam tipe ASN.1 adalah elemen dari kumpulan nilai yang disimpan tipe itu. ASN.1 memiliki empat macam tipe:

- tipe sederhana (*simple types*), sifatnya "atom" dan tidak mempunyai komponen,
- tipe terstruktur (*structured types*), memiliki komponen di dalamnya,
- tipe dengan tag (*tagged types*), tipe yang diambil (*derived*) dari tipe lain,
- tipe lain yang menyertakan tipe `CHOICE` dan tipe `ANY`.

Tipe dan nilai bisa dinamakan dengan operator penetapan (*assignment operator*) "::=", dan nama tadi bisa digunakan untuk mendefinisikan tipe dan nilai lain.

Setiap tipe ASN.1 selain `CHOICE` dan `ANY` memiliki tag, yang terdiri dari suatu kelas dan tag bilangan nonnegatif. Tipe ASN.1 sama secara abstrak jika dan hanya jika bilangan tag-nya sama. Dengan kata lain, nama tipe ASN.1 tidak mempengaruhi arti, hanya tag yang berpengaruh.

Ada empat kelas tag:

- *Universal*, untuk tipe yang artinya sama untuk semua aplikasi. Tipe ini hanya didefinisikan dalam standar X.208.
- *Application*, untuk tipe yang artinya spesifik untuk suatu aplikasi, seperti layanan direktori X.500. Tipe dalam dua aplikasi berbeda bisa memiliki tag *application-specific* yang sama dan arti berbeda.
- *Private*, untuk tipe yang artinya spesifik untuk perusahaan yang disebutkan.
- *Context-specific*, untuk tipe yang artinya spesifik pada tipe terstruktur yang disebutkan. Tag *context-specific* digunakan untuk membedakan antara tipe komponen yang memiliki tag dasar

Tipe	Bilangan tag (decimal)	Bilangan tag (hexadecimal)
`BOOLEAN`	1	01
`INTEGER`	2	02
`BIT STRING`	3	03
`OCTET STRING`	4	04
`NULL`	5	05
`OBJECT IDENTIFIER`	6	06
`Object Descriptor`	7	07
`EXTERNAL`	8	08
`SEQUENCE` dan `SEQUENCE OF`	16	10
`SET` dan `SET OF`	17	11
`NumericString`	18	12
`PrintableString`	19	13
`T61String` (`TeletexString`)	20	14
`VideotexString`	21	15
`IA5String`	22	16
`GraphicString`	25	19
`VisibleString` (`ISO64String`)	26	1A
`GeneralString`	27	1B
`UTCTime`	23	17
`GeneralizedTime`	24	18

Tabel 2.1: Tipe ASN.1 dan tag kelas universal-nya.

yang sama di dalam konteks tipe terstruktur yang diberikan, dan tipe komponen dalam dua tipe terstruktur yang berbeda bisa memiliki tag sama dengan arti berbeda.

Tipe dengan tag selain *universal* selalu diperoleh dengan tag eksplisit atau implisit.

Tipe dan nilai ASN.1 ditulis dalam notasi yang fleksibel dan mirip seperti bahasa pemrograman, dengan aturan khusus:

- Layout tidak signifikan, multi spasi dan baris baru bisa dianggap sebagai satu spasi.
- Komentar dibatasi dengan pasangan hipenasi (–), atau sepasang hipenasi dan baris baru.
- *Identifier* (untuk nama nilai dan *field*) dan referensi tipe (nama tipe) terdiri dari huruf besar dan huruf kecil, angka, hipenasi, dan spasi. *Identifier* diawali dengan huruf kecil, tipe referensi diawali dengan huruf besar.

Tabel 2.1 menampilkan tag untuk tipe kelas ASN.1.

Tipe sederhana

Tipe sederhana adalah tipe yang tidak berisi komponen lain, sifatnya "*atomic*". Tipe sederhana yang digunakan dalam SET adalah:

`BOOLEAN`, nilainya hanya benar atau salah.

`INTEGER`, bilangan bulat.

`BIT STRING`, string yang terdiri dari bit (0 dan 1).

`OCTET STRING`, string yang terdiri dari oktet (nilai 8-bit).

`NULL` , nilai null.

`OBJECT IDENTIFIER`, pengenal suatu obyek, terdiri dari kumpulan bilangan bulat berurutan yang mengidientifikasi suatu obyek seperti algoritma atau tipe atribut.

`PrintableString`, string karakter yang bisa dicetak.

`T61String`, string yang terdari dari karakter T.61 (8-bit).

`IA5String`, string yang terdiri dari karakter IA5 (ASCII).

`NumericString`, string yang terdiri dari karakter bilangan (0 sampai 9).

`UTCTime`, waktu Greenwich Mean Time (GMT).

`GeneralizedTime`, sama seperti `UTCTime`, tapi menyimpan format tahun dalam format empat digit.

Tipe terstruktur

Tipe terstruktur memiliki komponen di dalamnya. ASN.1 mendefinisikan empat tipe terstruktur:

`SEQUENCE`, koleksi berurutan tipe berjumlah satu atau lebih.

`SEQUENCE OF`, koleksi berurutan sebuah tipe yang diberikan dengan jumlah kemunculan nol atau lebih.

`SET`, koleksi tidak berurutan tipe berjumlah satu atau lebih.

`SET OF`, koleksi tidak berurutan sebuah tipe yang diberikan dengan jumlah kemunculan nol atau lebih.

Tipe terstruktur bisa memiliki komponen *optional*, kemungkinan nilai default.

Tipe dengan tag implisit dan eksplisit

Pemakaian tag berguna untuk membedakan berbagai tipe dalam suatu aplikasi, juga sudah dipakai luas untuk membedakan tipe komponen dalam tipe terstruktur. Sebagai contoh, komponen *optional* dengan tipe SET atau SEQUENCE biasanya diberikan tag *context-specific* yang berbeda untuk mencegah artinya mendua.

Ada dua cara untuk memberi tag suatu tipe, implisit dan eksplisit. Tipe dengan tag explisit diambil dari tipe lain dengan menambahkan satu tag luar pada tipe yang dimaksud. Hasilnya, tipe dengan tag eksplisit adalah tipe terstruktur yang terdiri atas satu komponen, tipe di dalamnya. Notasi tag eksplisit adalah kata kunci ASN.1 `[` ***nomor kelas*** `]` `EXPLICIT`.

Kata kunci `[` ***nomor kelas*** `]` `EXPLICIT` sendiri sama dengan tag explisit, kecuali jika "modul" tempat tipe didefinisikan mempunyai *default* tag implisit.

Tipe dengan tag implisit dianggap sama dengan tipe yang disimpan, tapi tag-nya berbeda.

Tipe lain

Tipe lain dalam ASN.1 mencakup tipe `CHOICE` dan `ANY`. Tipe `CHOICE` menyatakan perpaduan satu atau lebih alternatif, tipe `ANY` menyatakan nilai tidak tentu dari tipe tidak tentu, dengan tipenya kemungkinan didefinisikan pada registrasi suatu *object identifier* atau nilai *integer*.

2.2.2 DER

DER adalah subset dari BER, yang menjamin pengkodean yang unik untuk setiap nilai ASN.1. Semua aturan BER berlaku pada DER, ditambah dengan beberapa aturan spesifik.

Dalam BER ada tiga metode pengkodean nilai ASN.1, masing-masing tergantung pada tipe nilainya dan panjang nilai itu diketahui atau tidak. Tiga metode itu adalah:

- *primitive* (primitif) dan pengkodean dengan panjang diketahui (*definite-length*),
- *constructed* (terbangun) dan pengkodean dengan panjang diketahui,
- *constructed* dan panjang tidak diketahui (*indefinite-length*).

Tipe non-string sederhana memakai metode yang primitif dengan panjang diketahui. Tipe terstruktur memakai salah satu metode *constructed*, dan tipe string sederhana memakai salah satu dari tiga metode itu, tergantung apakah panjang nilainya diketahui. Tipe dengan tag implisit memakai metode tipe yang disimpam sedangkan tipe dengan tag eksplisit memakai salah satu dari metode *constructed*.

Dalam tiap metode, pengkodean BER memiliki tiga atau empat bagian berikut:

- Oktet pengenal (*identifier octets*). Oktet ini mengidentifikasikan kelas dan bilangan tag nilai ASN.1-nya, dan mengindikasikan primitif atau terbangun.
- Oktet panjang (*length octets*). Untuk metode dengan panjang definit, oktet ini menyimpan jumlah oktet muatan. Untuk metode *constructed* dan panjang idefinit, oktet ini menunjukan bahwa panjang indefinit.
- Oktet muatan (*contents octets*). Untuk metode primitif panjang definit, oktet ini menyimpan representasi nyata nilainya. Untuk metode *constructed*, oktet ini menyimpan penggabungan pengkodean BER dari nilai komponennya.
- Oktet akhir-muatan (*end-of-contents octets*). Untuk metode *constructed* panjang indefinit, oktet ini menunjukkan akhir muatan. Untuk metode lain oktet ini tidak ada.

Metode primitif panjang definit

Metode ini diterapkan untuk tipe sederhana dan tipe yang diambil dari tipe sederhana dengan pemberian tag implisit. Panjang nilai harus sudah lebih dulu diketahui. Bagian-bagian pengkodean BER-nya adalah sebagai berikut:

Oktet pengenal. Ada dua bentuk: bilangan tag kecil (*low tag number*) untuk bilangan tag antara 0 dan 30, dan bilangan tag besar (*high tag number*) untuk bilangan tag lebih dari 30. Bentuk bilangan tag kecil terdiri dari satu oktet. Bit ke-8 dan ke-7 menspesifikasikan kelas. Bit ke-6 bernilai "0", mengindikasikan pengkodean merupakan primitif, dan bit ke-5 sampai dengan bit ke-1 adalah bilangan tag. Bentuk bilangan tag besar terdiri dari dua oktet atau lebih. Oktet pertama seperti bentuk bilangan tag kecil, kecuali bit ke-5 sampa dengan ke-1 semua bernilai "1". Oktet kedua dan seterusnya adalah bilangan tag, basis 128, diawali bilangan MSD (*most significant digit*), dengan sesedikit mungkin digit, dan bit ke-8 setiap oktet kecuali oktet terakhir diset ke nilai "1".

Oktet panjang. Ada dua bentuk: pendek untuk panjang nilai antara 0 dan 127, dan panjang tidak tentu untuk panjang nilai antara $2^{1008} - 1$. Bentuk pendek terdiri dari satu oktet, dengan bit ke-8 bernilai "0" dan bit ke-7 sampai dengan ke-1 berisi panjang yang disimpan. Bentuk panjang terdiri atas 2 sampai 127 oktet. Bit ke-8 oktet pertama bernilai "1" dan bit ke-7 sampai dengan ke-1 jumlah oktet tambahan. Oktet kedua dan seterusnya berisi panjang, basis 256, diawali dengan bilangan MSD.

Oktet muatan. Oktet ini berisi representasi nyata nilai yang disimpan (atau nilai dari tipe yang disimpan, jika diambil dari pemberian tag implisit).

Metode *constructed* panjang definit

Metode ini diterapkan pada tipe string sederhana, tipe terstruktur, tipe yang diambil dari tipe string sederhana dan tipe terstruktur dengan pemberian tag implisit, dan tipe yang diambil dari tipe lain selain dengan pemberian tag eksplisit. Panjang nilai harus sudah lebih dulu diketahui. Bagian-bagian pengkodean BER-nya adalah sebagai berikut:

Oktet pengenal. Sama dengan metode primitif, tapi bit ke-6 bernilai "1" menandakan pengkodean adalah *constructed*.

Oktet panjang. Sama dengan metode primitif.

Oktet muatan. Oktet ini berisi penggabungan komponen pengkodean BER dengan nilai:

- Untuk Tipe string sederhana dan tipe yang diambil dari tipe string sederhana dengan pemberian tag implisit, nilainya adalah penggabungan pengkodean BER dari substring nilai yang bersambung (untuk pemberian tag implist adalah nilai yang dikandung).
- Untuk tipe terstruktur dan tipe yang diambil dari tipe terstruktur dengan pemberian tag implisit, nilainya adalah penggabungan pengkodean BER dari komponen nilai (untuk pemberian tag implisit adalah nilai yang dikandung).
- Untuk tipe yang diambil dari tipe lain selain tag eksplisit, nilainya adalah pengkodean BER nilai yang dikandung.

Metode *constructed* panjang indefinit

Metode ini berlaku untuk tipe string sederhana, tipe terstruktur, tipe yang diambil dari tipe string sederana dan tipe terstruktur dengan pemberian tag implisit, dan tipe yang diambil dari tipe lain selain pemberian tag eksplisit. Panjang nilai tidak harus diketahui. Bagian pengkodean BER-nya adalah sebagai berikut:

Oktet pengenal. Sama dengan metode *constructed* panjang definit.

Oktet panjang. Satu oktet, `80`.

Oktet muatan. Sama dengan metode *constructed* panjang definit.

Oktet akhir-muatan. Dua oktet, `00 00`.

Aturan spesifik DER

DER ditujukan untuk aplikasi yang membutuhkan pengkodean string oktet secara unik, seperti dalam kasus komputasi tandatangan digital dalam suatu nilai ASN.1. DER menambahkan batasan kepada aturan BER yang sudah diterangkan sebelumnya:

1. Bila panjang bernilai antara 0 dan 127, oktet panjang harus menggunakan bentuk pendek.
2. Bila panjang bernilai 128 atau lebih, oktet panjang harus menggunakan bentuk panjang, dan panjang harus dikodekan dalam jumlah oktet minimum.
3. Untuk tipe string sederhana dan tipe dengan pemberian tag implisit yang diambil dari tipe string sederhana, metode yang dipakai haruslah metode primitif panjang definit.
4. Untuk tipe terstruktur, tipe dengan pemberian tag implisit diambil dari tipe terstruktur, dan tipe dengan pemberian tag eksplisit diambil dari tipe lain, harus menggunakan metode *constructed* panjang definit.

Beberapa batasan lain didefinisikan untuk tipe tertentu seperti `BIT STRING`, `SEQUENCE`, `SET`, dan `SET OF`.

2.3 SERTIFIKAT DIGITAL

Sertifikat digital (*digital certificate*) adalah sertifikat yang menerangkan identitas pemilik sertifikat yang disimpan dalam bentuk elektronik. Sertifikat ini menyimpan informasi nama pemilik sertifikat, badan sertifikasi yang mengeluarkan, masa validasi sertifikat, kunci publik yang dimiliki, dan ekstensi (*extension*) lain. Ekstensi ini digunakan SET dalam transaksi yang menerapkan SET.

Sertifikat digital memiliki karakteristik sebagai berikut:

- setiap orang yang memiliki akses ke kunci publik certificate authority [lihat: 2.5] bisa mendapatkan kunci publik yang sudah disertifikasi.
- tidak ada pihak lain selain certificate authority bisa memodifikasi sertifikat tanpa terdeteksi (sertifikat tidak dapat dipalsukan).

Format sertifikat digital yang dipakai dalam SET adalah format yang ditentukan ITU (International Telecommunication Union) dalam standar X.509 [ITUT97].

Tipe data ASN.1 yang merepresentasikan sertifikat X.509 adalah sebagai berikut:

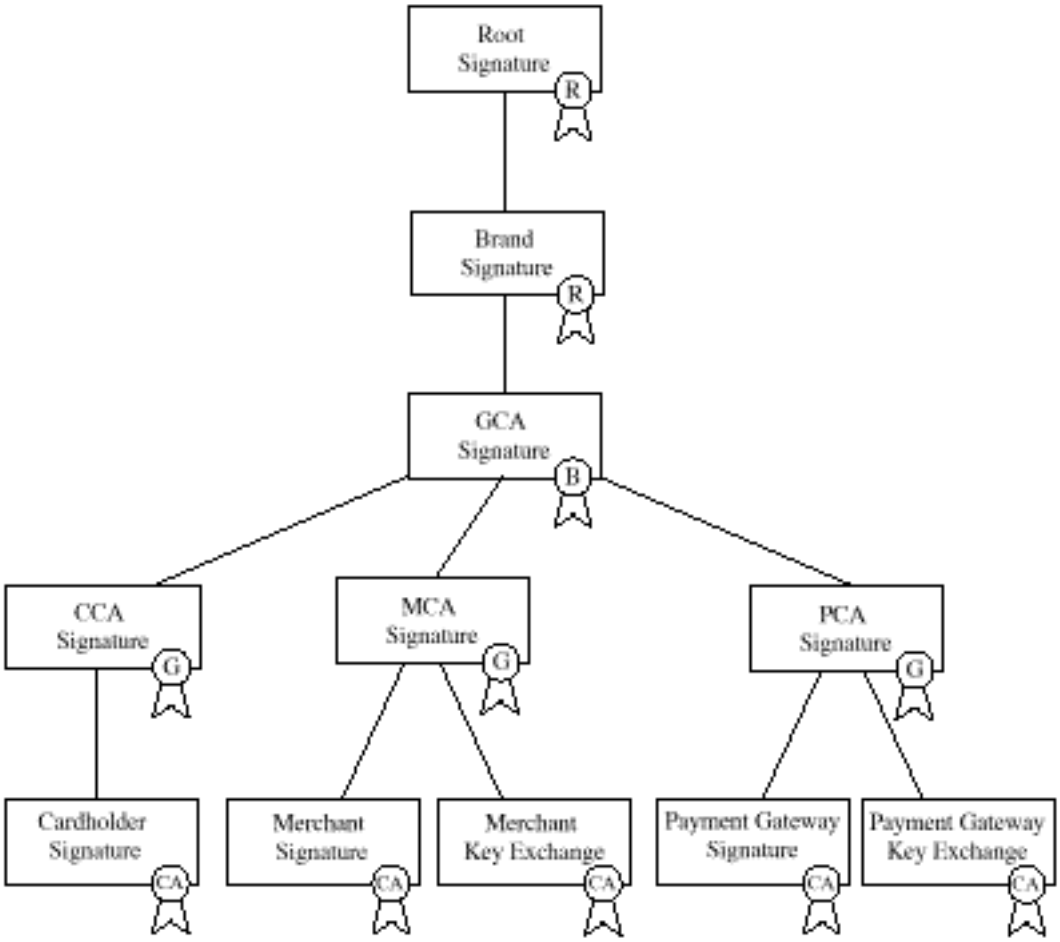
```
Certificate     ::= SIGNED { SEQUENCE {
    version                 [0] Version DEFAULT v1,
    serialNumber             CertificateSerialNumber,
    signature                AlgorithmIdentifier,
    issuer                   Name,
    validity                 Validity,
    subject                  Name,
    subjectPublicKeyInfo     SubjectPublicKeyInfo,
    issuerUniqueIdentifier   [1] IMPLICIT UniqueIdentifier OPTINAL,
                                  -- if present, version must be v2 or v3
    subjectUniqueIdentifier  [2] IMPLICIT UniqueIdentifier OPTINAL
                                  -- if present, version must be v2 or v3
    extensions               [3] Extensions OPTIONAL
                                  -- if present, version must be v3 -- } }
Version                 ::= INTEGER{ v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER
AlgorithmIdentifier     ::= SEQUENCE {
    algorithm             ALGORITHM.&id({SupportedAlgorithms}),

    parameters            ALGORITHM.&Type({SupportedAlgorithms}{@algorithm} OPTIONAL }
```

SET menggunakan X.509 versi 3 karena SET memakai tipe **extensions** di dalam X.509. Ekstensi ini digunakan untuk menyimpan berbagai nilai yang diperlukan dalam transaksi SET.

2.4 *CERTIFICATE REVOCATION LIST* (CRL)

Certificate Revocation List (Daftar Pencabutan Sertifikat) adalah suatu mekanisme untuk mempublikasikan dan distribusi daftar sertifikat yang dicabut dan kadaluarsa, didefinisikan dalam X.509.



Gambar 2.1: Hirarki Kepercayaan

Setiap CA (kecuali MCA dan CCA) akan memelihara suatu CRL. Setiap CA akan mendistribusikan CRL. BrandCRLIdentifier (BCI) adalah suatu struktur yang didefinisikan oleh SET dan berisi daftar CRL terbaru untuk CA dalam domain Brand yang disebutkan. Setiap kali CA mengeluarkan CRL baru, BCI yang berhubungan diperbarui. BCI didistribusikan dalam semua pesan *downstream*. Kepemilikan BCI dan CRL yang diidentifikasikan menjamin EE menyaring sertifikat terhadap informasi pencabutan terakhir.

2.5 *CERTIFICATE AUTHORITY*

Certificate Authority (CA) adalah suatu entitas yang dipercaya orang atau kelompok lain untuk membuat dan memberikan sertifikat digital. CA juga berfungsi sebagai sumber terpercaya untuk mendapatkan sertifikat entitas lain.

CA membuat sertifikat suatu pemohon sertifikat dengan cara menandatatangani (secara digital) sekumpulan informasi, termasuk nama pemohon (*user distinguished name*) dan kunci publik, termasuk juga sebuah *unique identifier optional* yang berisi informasi tambahan tentang si pemohon.

Sertifikat diverifikasi melalui suatu hirarki kepercayaan seperti yang digambarkan pada gambar 2.1. Setiap sertifikat dihubungkan pada tanda tangan sertifikat milik entitas yang mengeluarkan sertifikat. Sertifikat divalidasi dengan mengikuti hirarki kepercayaan sampai pada *Root CA*. Jalur yang dilalui sertifikat pada saat validasi disebut "rantai tanda tangan" (*signature chain*).

Root CA (RCA) adalah CA paling atas. Sertifikat Root CA ditandatangani sendiri (self-signed) oleh si Root CA. Root CA ini dioperasikan dengan kontrol fisik yang sangat ketat, dan senantiasa off-line. Sekali dalam beberapa waktu Root CA diakses Otoritas untuk mengeluarkan sertifikat baru untuk Brand CA.

Brand CA (BCA) adalah CA untuk suatu merk kartu pembayaran tertentu. Di bawah BCA ada suatu Brand-Geopolitical CA (GCA) yang merupakan *optional*. GCA ini hanya ada untuk suatu wilayah yang memiliki kesamaan geografis atau politis. Kemudian di bawah GCA atau langsung di bawah BCA ada Cardholder CA (CCA), Merchant CA (MCA), dan Payment Gateway CA (PCA). Masing-masing CA menangani entitas di bawahnya, yaitu Cardholder, Merchant, dan Payment Gateway. Cardholder hanya memiliki satu sertifikat yang berfungsi untuk identifikasi diri/membuat tanda tangan digital. Merchant dan Payment Gateway masing-masing memiliki dua sertifikat, yaitu untuk membuat tanda tangan digital, dan untuk enkripsi data.

Validasi melalui rantai tanda tangan harus memenuhi syarat:

- Setiap sertifikat dalam jalur - mulai dari sertifikat End Entity (EE) sampai ke sertifikat Root - harus divalidasi, dan
- Setiap sertifikat harus dipetakan dengan benar pada CA yang mengeluarkan sertifikat.

Validasi rantai sertifikat SET dilakukan sesuai dengan kebutuhan pemrosesan yang dispesifikasikan pada Seksi 12.4.3 Amandemen 1 spesifikasi X.509 dan sesuai dengan spesifikasi kebutuhan SET. Aturan SET tentang validasi rantai merupakan tambahan dari yang sudah dispesifikasikan X.509. Rantai sertifikat SET terdiri dari sekumpulan sertifikat mulai dari EE sampai sertifikat Root, ditambah semua turunan Root sampai kembali ke sertifikat Root awal.

2.6 PKCS

PKCS (*Public Key Cryptography Standard*) adalah standar yang ditetapkan RSA Laboratories. Standar ini mengatur berbagai protokol dan format yang berhubungan dengan kriptografi yang melibatkan penyandian asimetrik.

PKCS terbagi atas beberapa standar, yaitu:

- PKCS #1 mendefinisikan mekanisme untuk enkripsi dan penandatanganan data menggunakan algoritma kunci publik RSA.

- PKCS #3 mendefinisikan protokol pertukaran kunci menggunakan algoritma Diffie-Hellman.
- PKCS #5 mendefinisikan metode untuk mengenkrip suatu string dengan suatu kunci rahasia yang diambil dari kata kunci.
- PKCS #6 mendeskripsikan format untuk sertifikat yang diperluas. Sertifikat yang diperluas terdiri dari sertifikat X.509 bersama dengan sekumpulan atribut yang ditandatangani pemberi sertifikat. Standar ini sudah tidak berlaku lagi dengan adanya X.509 versi 3.
- PKCS #7 mendefinisikan sintaks umum pesan yang melibatkan penambahan kriptografi seperti tanda tangan digital dan enkripsi.
- PKCS #8 mendeskripsikan format informasi kunci privat. Informasi ini melibatkan kunci privat untuk algoritma kunci publik tertentu, dan sekumpulan atribut.
- PKCS #9 mendefinisikan tipe atribut terpilih untuk digunakan pada standar PKCS lain.
- PKCS #10 mendeskripsikan sintaks untuk permintaan sertifikat.
- PKCS #11 mendefinisikan antar muka pemrograman yang tidak tergantung teknologi, yang dinamakan Cryptoki, untuk alat kriptografik seperti *smart card* dan kartu PCMCIA.

2.7 PROSEDUR PEMROSESAN JALUR SERTIFIKAT

Pemrosesan jalur sertifikat dilakukan dalam sebuah sistem yang mengharuskan penggunakan kunci publik suatu entitas yang terpisah (*remote*), misalnya sistem yang pemeriksaan tanda tangan digitalnya dilakukan oleh entitas terpisah. Peraturan sertifikat (*certificate policies*), batasan dasar (*basic constraints*), batasan nama (*name constraints*), dan ekstensi batasan peraturan (*policy constraints extensions*) telah dirancang untuk menyediakan implementasi yang otomatis dan *self-contained* untuk logika pemrosesan jalur sertiikasi.

Masukan untuk prosedur pemrosesan jalur sertifikat adalah:

1. Sekumpulan sertifikat yang membentuk jalur sertifikat.
2. Suatu nilai kunci publik atau pengenal kunci (jika kunci disimpan secara internal pada modul pemrosesan jalur sertifikat) yang terpercaya, untuk pemeriksaan sertifikat pertama dalam jalur sertifikat.
3. Sebuah *initial-policy-set* (kumpulan peraturan awal) terdiri atas satu atau lebih pengenal peraturan sertifikat, menandakan masing-masing peraturan dapat diterima oleh pemakai sertifikat untuk tujuan pemrosesan jalur sertifikat. Masukan ini juga bisa mengambil nilai khusus *any-policy*.

4. Suatu nilai indikator *initial-explicit-policy* (peraturan eksplisit awal), yang menandakan bila suatu pengenal peraturan yang dapat diterima harus muncul secara eksplisit dalam *field* ekstensi peraturan sertifikat pada semua sertifikat dalam jalur.

5. Suatu nilai indikator *initial-policy-mapping-inhibit* (pencegahan pemetaan peraturan inisial), yang mengindikasikan jika pemetaan peraturan dilarang dalam jalur sertifikat.

6. Nilai waktu (tanggal/jam) sekarang (jika tidak tersedia secara internal dalam modul pemrosesan jalur sertifikat).

Nilai butir 3, 4, dan 5 akan tergantung pada kebutuhan peraturan dalam kombinasi aplikasi dengan pengguna yangmengharuskan penggunaan kunci publik entitas akhir yang sudah disertifikasi.

Keluaran prosedur ini adalah:

1. Indikasi sukses atau gagal validasi jalur sertifikat.

2. Jika validasi gagal, suatu kode diagnosa menandakan alasan kegagalan.

3. Jika validasi berhasil, sekumpulan aturan dibatasi oleh CA dalam jalur sertifikat, bersama dengan seluruh *qualifier* untuk peraturan yang ditemukan dalam jalur sertifikat, atau nilai khusus *any-policy*. Selain jika *any-policy* dikembalikan, pengguna sertifikat harus menggunakan sertifikat sesuai dengan salah satu aturan yang teridentifikasi dan memproses semua *qualifier* untuk peraturan yang ada dalam jalur sertifikat.

4. Jika validasi berhasil dan butir 3 mengembalikan nilai *any-policy*, kumpuluan sema *qualifier* elemen peraturan yang ditemukan dalam jalur sertifikat.

5. Rincian setiap pemetaan aturan yang muncul dalam pemrosesan jalur sertifikat.[1]

Sebagai tambahan untuk langkah pemrosesan pada X.509, batasan-batasan berikut berlaku pada rantai sertifikat:

1. *Field* **certIssuer** dalam ekstensi **AuthorityKeyIdentifier** di sertifikat EE harus cocok dengan **IssuerName** pada sertifikat CA-nya.

2. *Field* **certSerialNumber** dalam ekstensi **AuthorityKeyIdentifier** di sertifikat EE harus cocok dengan **SerialNumber** pada sertifikat CA-nya.

3. Tanggal **Validity** (dalam sertifikat dan dalam ekstensi **PrivateKeyUsagePeriod**) dalam sertifikat EE harus di dalam rentang tanggal **Validity** sertifikat CA.

[1] Jika validasi berhasil, sistem yang menggunakan sertifikat masih boleh menolak menggunakan sertifikat hasil nilai *qualifier* peraturan atau informasi lain dalam sertifikat.

4. Tanggal **Validity notBefore** dalam sertifikat EE harus dalam rentang tanggal **Validity** pada ekstensi **PrivateKeyUsagePeriod** dalam sertifikat CA.

5. Untuk setiap sertifikat di bawah sertifikat Root, **BrandNames** dalam struktur Name pada **SubjectDistinguishedName** tiap sertifikat harus cocok. Jika ada, tipe produk dalam struktur Name tiap sertifikat juga harus cocok.

6. Bila memeriksa status penarikan sertifikat, pemeriksa harus yakin bahwa dia mempunyai BrandCRLIdentifier (BCI) yang terkini dan BCI ini memiliki semua CRL di dalamnya.

Hal-hal berikut harus divalidasi pada sertifikat EE, sebagai tambahan pemrosesan rantai sertifikat X.509:

1. *Field* **KeyUsage** pada ekstensi **KeyUsage** valid untuk tujuan yang akan dilakukan.

2. *Field* **subjectType** pada ekstensi **BasicConstraints** menunjuk ke EE.

3. Ekstensi privat **CertificateType** berhubungan dengan konteks penggunaan sertifikat.

4. Tanda tangan valid.

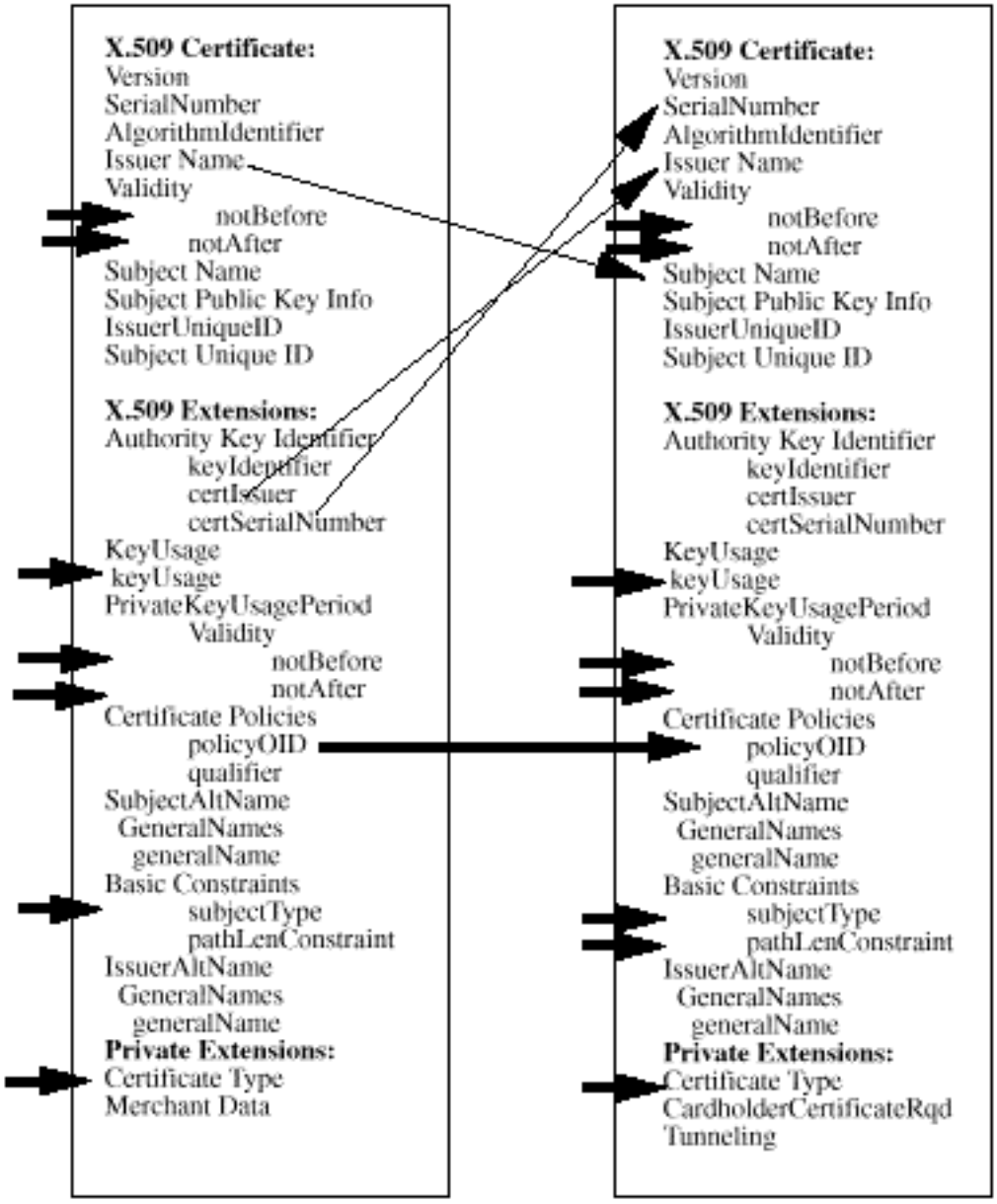
Hal-hal berikut harus divalidasi untuk setiap sertifikat CA, sebagai tambahan pemrosesan rantai sertifikat X.509:

1. *Field* **KeyUsage** pada ekstensi **KeyUsage** valid untuk tujuan yang akan dilakukan.

2. Ekstensi private **CertificateType** berhubungan dengan konteks penggunaan sertifikat.

Gambar 2.2 menjelaskan logika elemen data sertifikat, dengan penekanan pada elemen data yang digunakan untuk validasi rantai tanda tangan. Panah tebal menandakan *field* yang harus divalidasi dan panah tipis menandakan *field* yang harus bernilai sama.

Validasi tanggal kadaluarsa sertifikat merupakan bagian terpadu proses validasi rantai tanda tangan. Validasi suatu sertifikat EE mengharuskan semua rantai yang dipercaya valid dan tidak ada sertifikat dalam rantai yang kadaluarsa.



Gambar 2.2: Validasi Sertifikat

Bab 3

PROTOKOL PERMINTAAN SERTIFIKAT CARDHOLDER PADA SET

Bab ini berisi penjelasan landasan teori kebutuhan SET dan spesifikasi protokol permintaan pembuatan sertifikat Cardholder pada SET.

3.1 DEFINISI SET

Secure Electronic Transaction (selanjutnya disebut sebagai SET) adalah sebuah protokol transaksi elektronik yang dikembangkan Visa dan Mastercard, sebagai suatu metode untuk mengamankan transaksi kartu pembayaran yang dilangsungkan melalui jaringan elektronik terbuka. SET dipublikasikan sebagai spesifikasi terbuka bagi industri elektronik. Spesifikasi ini terbuka untuk diaplikasikan ke semua pelayanan kartu pembayaran dan boleh digunakan digunakan oleh pengembang perangkat lunak untuk membuat aplikasi. Saran dan bantuan dalam pengembangan spesifikasi ini disediakan oleh GTE, IBM, Microsoft, Netscape, RSA, SAIC, Terisa, dan Verisign.

SET bertujuan memenuhi kebutuhan institusi keuangan dan sistem pembayaran yaitu:

- transmisi rahasia
- autentikasi pihak yang terlibat
- jaminan integritas data instruksi pembayaran atas barang dan jasa
- autentikasi identitas pembeli terhadap pedagang dan sebaliknya

Ciri jaringan komunikasi adalah anonim, karena itu harus dibuat prosedur baru untuk menggantikan transaksi yang ada sekarang (tatap muka, pemasanan lewat surat/telepon), termasuk autentikasi si pembeli oleh pedagang. Pembeli juga membutuhkan autentikasi apakah pedagang menerima transaksi SET dan dibolehkan menerima kartu pembayaran.

3.2 SPESIFIKASI PROTOKOL SET

Rancangan SET menggunakan sertifikat X.509 versi 3 agar bisa menggunakan kunci publik untuk pembuatan tanda tangan digital dan enkripsi. Sertifikat ini berisi kunci publik beserta bukti otentikasi kunci tersebut.

Tanda tangan pada sertifikat cardholder memberikan otentikasi dan integritas informasi yang dikirimkan ke merchant dan ke Payment Gateway. SET mendukung sistem yang mengharuskan cardholder memiliki sertifikat, dan juga sistem yang tidak mengharuskan cardholder memiliki sertifikat. Suatu merk kartu pembayaran harus menentukan aplikasi SET yang dipakai itu membutuhkan sertifikat atau tidak.

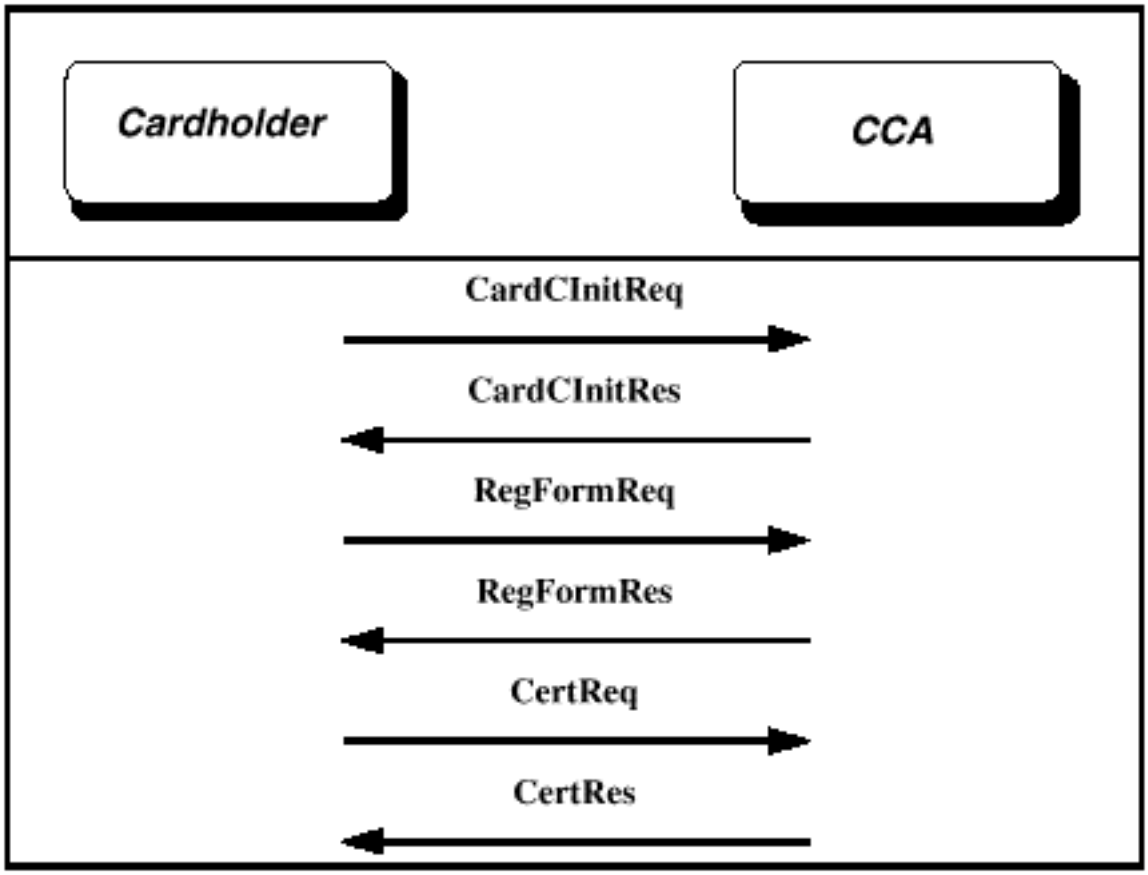
3.3 PROTOKOL PERMINTAAN SERTIFIKAT

Dalam protokol ini, pihak yang terlibat adalah Cardholder (CH) sebagai EE, dan Cardholder CA (CCA) sebagai CA. Untuk meminta sertifikat Cardholder harus lebih dulu memiliki beberapa hal berikut:

- Sebuah rekening yang valid.
- Kemampuan untuk membuat pasangan kunci publik/privat dan menyimpan kunci privat secara aman.
- Pengetahuan tentang informasi tertetntu untuk digunakan dalam proses identifikasi dan otentikasi CH sebagaimana disyaratkan oleh Issuer kartu pembayaran (Issuer mempunya syarat lain untuk informasi ini).
- *Universal Resource Locator* (URL) atau alamat mail Internet milik CCA.
- Browser yang *SET-compliant* atau aplikasi SET.

Tahapan pemrosesan permintaan sertifikat oleh CH ke CCA adalah sebagai berikut:

1. Aplikasi CH mengirimim pesan CardCInitReq ke CA, menggunakan Brand ID yang sudah tersimpan atau yang diambil dari pesan inisiasi sertifikat.
2. CCA mengirim pesan CardCInitRes ke aplikasi SET di CH.



Gambar 3.1: Pertukaran pesan permintaan sertifikat

3. Aplikasi CH mengirim pesan RegFormReq ke CCA.

4. CCA mengirim RegFormRes berisi formulir registrasi dan kebijakan aturan (*policy statement*).

5. Aplikasi CH menampilkan formulir registrasi dan kebijakan aturan. User mengisi informasi yang diminta dan menyetujui aturan yang ada.

6. Aplikasi CH menyertakan formulir registrasi yang sudah diisi, kunci publik yang baru, dan sertifikat yang sedang diperbarui (jika ada), ke dalam pesan CertReq, dan mengirimkannya ke CCA.

7. CCA membuat sertifikat.

8. CCA menyertakan sertifikat ke dalam pesan CertRes dan mengirimkannya ke CH.

Gambar 3.1 menjelaskan pertukaran informasi dasar antara CH dan CCA.

Jika CertRes memberi tahu bahwa sertifikat belum siap, EE (dalam hal ini CH) bisa mengirimkan pesan CertInqReq untuk mendapatkan status permintaan sertifikat. Pesan CertInqRes menjawab apakah sertifikat sudah siap, menyediakan status jika ada masalah dengan permintaan sertifikat, atau memberi tahu jika sertifikat sudah siap untuk diambil.

CardCInitReq	{RRPID, LID-EE, Chall-EE, BrandID, [Thumbs]}
RRPID	ID pasangan permintaan/jawaban.
LID-EE	ID lokal; dibuat oleh dan untuk sistem CH.
Chall-EE	Challenge CH untuk usia tanda tangan CCA.
BrandID	BrandID dari sertifikat yang diminta.
Thumbs	Daftar Sertifikat (termasuk Root), CRL, dan *thumbprint* BCI yang disimpan CH.

Tabel 3.1: CardCInitReq

3.4 INISIASI PEMROSESAN PERMINTAAN/JAWABAN SERTIFIKAT CARDHOLDER

Bagian ini menjelasakan tahap inisiasi pemrosesan permintaan sertifikat Cardholder. Sesudah aplikasi SET dijalankan, CH mengirim pesan CardCInitReq ke CCA, ditandai melalui Thumbprints sertifikat, CRL, dan BCI yang berada di dalam *cache* sertifikat. CCA merespon dengan pesan CardCInitRes berisi semua serrtifikat, CRL, dan BCI yang dibutuhkan CH untuk verifikasi tanda tangan, termasuk sertifikat enkripsi untuk pesan berikutnya. Tabel 3.1 menerangkan struktur CardCInitReq.

Untuk membuat pesan CardCInitReq, aplikasi SET harus melakukan langkah-langkah berikut:

1. Buat RRPID.
2. Buat LID-EE.
3. Buat Chall-EE yang baru dan acak.
4. *Copy* BrandID yang sudah tersimpan atau diterima pada pesan inisiasi.
5. Isi Thumbs (*optional*), yang berisi *thumbprint* untuk setiap CRL, sertifikat SET, BCI, dan sertifikat Root yang berada di *cache* terpercaya CH, jika masing-masing ada.
6. Jalankan *Compose Message Wrapper* (dijelaskan dalam [VIMA97b] Bagian I halaman 76)

Saat CCA menerima CardCInitReq, CCA memprosesnya dengan langkah-langkah sebagai berikut:

1. Terima CardCInitReq dari *Receive Message* (dijelaskan dalam [VIMA97b] bagian I halaman 77).
2. Periksa bahwa RRPID dalam CardCInitReq cocok dengan RRPID dalam *Message Wrapper*. Jika tidak cocok, kembalikan suatu *Error Message* dengan ErrorCode diset ke nilai unknownRRPID.
3. Simpan Thumbs, LID-EE, Chall-EE, dan RRPID untuk digunakan dalam CardCInitRes.

Setelah CCA memproses CardCInitReq, CCAakan membuat CardCInitRes. Seperti dengan tipe SignedData, sertifikat dan CRL yang dibutuhkan untuk memeriksa tanda tangan disertakan dalam CardCInitRes, di luar data "To Be Signed".

Langkah-langkah yang dilakukan CCA untuk membuat CardCInitRes adalah:

CardCInitRes	S(CA, CardCInitResTBS).
CardCInitResTBS	{RRPID, LID-EE, Chall-EE, [LID-CA], CAEThumb, [BrandCRLIdentifier], [Thumbs]}
RRPID	ID pasangan permintaan/jawaban.
LID-EE	Di-*copy* dari CardCInitReq.
Chall-EE	Di-*copy* dari CardCInitReq.
LID-CA	ID lokal, dibuat oleh dan untuk sistem CCA.
CAEThumb	Thumbprint sertifikat CCA untuk pertukaran kunci yang akan digunakan CH untuk mengenkripsi RegFormReq.
BrandCRLIdentifier	BCI
Thumbs	Di-*copy* dari CardCInitReq.

Tabel 3.2: CardCInitRes

1. Buat "CardCInitResTBS" data dengan cara:
   (a) *Copy* RRPID, LID-EE, dan Chall-EE, dari nilai yang diterima dalam CardCInitReq.
   (b) Buat LID-CA (*optional*).
   (c) Isi caeThumb dengan *thumbprint* sertifikat enkripsi data milik CCA.
   (d) Jika BrandCRLIdentifier tidak disebutkan dalam Thumbs yang diterima dalam CardCInitReq, isi BrandCRLIdentifier.
   (e) *Copy* Thumbs dari CardCInitReq.
2. Tandatangani CardCInitResTBS dalam bentuk pengkodean DER-nya. Set tipe muatan SignedData ke nilai id-set-content-CardCInitResTBS. Sertakan dalam SignedData semua sertifikat, CRL, atau BrandCRLIdentifier yang tidak dinyatakan oleh Thumbs dan yang mungkin akan digunakan CH untuk memeriksa tanda tangan CCA atau untuk enkripsi RegFormReq dan CertReq.
3. Jalankan *Compose Message Wrapper* untuk mengirimkan CardCInitRes ke CH.

Aplikasi SET di CH memproses CardCInitRes dengan langkah-langkah:

1. Terima CardCInitRes dari *Receive Message*
2. Periksa bahwa RRPID cocok dengan yang dikirim dalam CardCInitReq dan yang diterima dalam pembungkus CardCInitRes. Jika tidak cocok, kembalikan *Error Message* dengan ErrorCode diset ke nilai unknownRRPID.
3. Periksa bahwa Thumbs yang diterima cocok dengan yang dikirim dalam pesan CardCInitReq. Jika tidak cocok, kembalikan *Error Message* dengan ErrorCode diset ke thumbsMismatch.
4. Periksa bahwa Chall-EE yang diterima sama dengan yang dikirim dalam CardCInitReq. Jika tidak sama, kembalikan *Error Message* dengan ErrorCode diset ke challengeMismatch.
5. Jika disertakan, simpan LID-CA yang diterima untuk dipakai dalam RegFormReq. Periksa bahwa Chall-EE yang diterima sama dengan yang dikirim dalam CardCInitReq.

6. Periksa bahwa aplikasi CH mendukung algoritma yang disebutkan dalan ekstensi Tunneling dalam sertifikat enkripsi CA. Jika aplikasi CH tidak mendukung algoritma enkripsi yang dipakai CA, beritahu user dan hentikan pemrosesan pesan CA selanjutnya.

3.5 PEMROSESAN PERMINTAAN/JAWABAN FORMULIR REGISTRASI CARDHOLDER

Sesudah menerima sertifikat, CRL, dan BCI yang sesuai, CH dapatmeminta dengan aman formulir registrasi sertifikat lewat RegFormReq. Jika CCA berhasil memvalidasi permintaan formulir registrasi itu, CCA memberikan formulir dalam RegFormRes. Jika CCA tidak memiliki formulir registrasi untuk permintaan CH dan/atau memiliki informasi tambahan mengenai penolakan permintaan layanan untuk disampaikan ke CH, juga disebutkan dalam RegFormRes.

RegFormReq dienkrip oleh aplikasi CH menggunakan sertifikat yang diterima dari CCA dalam CardCInitRes. Aplikasi CH kemudian membuat RegFormReq, bila pemrosesan CardCInitRes berhasil, dengan langkah-langkah:

1. Buat "RegFormReqData" dengan cara:
   (a) Buat RRPID baru.
   (b) *Copy* LID-EE yang dikirimkan dalam CardCInitReq.
   (c) Buat Chall-EE2 baru.
   (d) Jika ada LID-CA dalam CardCInitRes, *copy* LID-CA dari CardCInitRes itu.
   (e) Isi RequestType sesuai dengan tabel 3.4.
   (f) Isi Language.
   (g) Sertakan Thumbs (*optional*), yang menyimpan *thumbprint* untuk tiap CRL, sertifikat SET, BrandCRLIdentifier, dan sertifikat Root yang tersimpan dalam *cache* terpercaya CH, jika masing-masing ada.
2. Buat "RegFormReqTBE" dengan cara:
   (a) Masukkan RegFormReqData.
   (b) Isi PANOnly menggunakan PAN dan ExNonce. PAN tidak dimodifikasi.
   (c) Buat *hash* dari PANOnly yang dalam bentuk terkode DER, dengan algoritma SHA-1. Tentukan tipe muatan digestedData ke id-set-content-PANOnly.
3. Buat "To Be Extra Encrypted" Data dengan cara:

RegFormReq	EXH(CA, RegFormReqData, PANOnly)
RegFormReqData	{RRPID, LID-EE, Chall-EE2, [LID-CA], RequestType, Language, [Thumbs]}
PANOnly	lihat di bawah.
RRPID	ID pasangan permintaan/jawaban.
LID-EE	Di-*copy* dari CardCInitRes.
Chall-EE2	Challenge EE untuk memeriksa usia tanda tangan CA.
LID-CA	Di-*copy* dari CardCInitRes.
RequestType	
Language	Bahasa yang ingin digunakan selanjutnya dalam pemrosesan ini.
Thumbs	Daftar Sertifikat (termasuk Root), CRL, dan BCI yang saat ini disimpan CH.
PANOnly	{PAN, EXNonce}
PAN	Nomor Kartu Pembayaran CH
EXNonce	bilangan acak yang digunakan untuk menyamarkan PAN.

Tabel 3.3: RegFormReq

RequestType	Hanya Sertifikat Tanda Tangan	Hanya Sertifikat Enkripsi	Keduanya
Cardholder Initial (sertifikat baru)	1	2*	3*
Cardholder Renewal (pembaruan sertifikat)	10	11*	12*

Request Type	Batasan
2*	Harus memiliki sertifikat tanda tangan yang valid dan menggunakan kunci privat yang berhubungan untuk menandatangani permintaan untuk sertifikat Enkripsi.
10, 12*	Kedua kunci privat yang berhubungan dengan sertifikat yang sedang diperbarui dan kunci privat untuk sertifikat tanda tangan yang baru harus digunakan untuk menanda tangani permintaan pembaruan.
11*	Pembaruan sertifikat Enkripsi: Subject Distinguished Name dalam sertifikat Tanda Tangan (yang digunakan untuk menandatangani permintaan) dan sertifikat Enkripsi harus cocok.

Tabel 3.4: Nilai RequestType dalam Formulir Registrasi Cardholder.

(a) Isi PAN. Jika PAN kurang dari 19 byte, tambahkan dengan nilai "00" sampai berjumlah 19 byte.

(b) Buat suatu *nonce* baru, EXNonce, untuk menyamarkan PAN.

4. Enkrip data menggunakan pemrosesan EXH dengan

(a) RegFormReqTBE sebagai data "To Be Ordinarily Encrypted" dan contentType EnvelopedData diset ke id-set-content-RegFormReqTBE, dan

(b) hasil dari langkah ketiga sebagai data "To Be Extra Encrypted".

5. Jalankan *Compose Message Wrapper* untuk mengirimkan RegFormReq ke CCA.

RequestType memiliki salah satu nilai yang tertera dalam tabel 3.4. Catatan: * menandakan pilihan yang disimpan untuk SET versi berikutnya.

Saat CCA menerima pesan RegFormReq, CCA memvalidasi pesan itu dan menentukan apakah formulir registrasi akan diberikan, dengan langkah-langkah:

1. Terima pesan RegFormReq dari *Receive Message*.
2. Simpan PAN dari data "To Be Extra Encrypted", sesudah menghapus tambahan byte yang ada.
3. Dari data "RegFormReqData", simpan RequestType, RRPID, LID-EE, Chall-EE2, LID-CA, Thumbs, dan Language.
4. Periksa bahwa RRPID yang diterima dalam pembungkus pesan cocok dengan yang ada dalam RegFormReqTBE. Jika tidak cocok, kembalikan suatu *Error Message* dengan ErrorCode diset ke unknownRRPID.

Sesudah memvalidasi RegFormReq, CCA akan membuat RegFormRes.

Jika validasi RegFormRes berhasil, formulir registrasi, pernyataan kebijakan, dan URL untuk merk dan logo kartu diberikan. Jika validasi gagal, alasan dan URL *optional* atau alamat e-mail *optional* untuk informasi lebih detail akan diberikan. Langkah-langkah untuk membuat RegFormRes adalah sebagai berikut:

1. Buat "RegFormTBS" dengan cara:
   (a) *Copy* RRPID, RequestType, LID-EE, Thumbs, dan Chall-EE2 dari RegFormReqData.
   (b) Jika LID-CA disediakan dalam CardCInitRes, *copy* LID-CA, kalau tidak, buat LID-CA untuk permintaan layanan ini.
   (c) Buat Chall-CA baru.
   (d) Jika BrandCRLIdentifier tidak dispesifikasikan dalam Thumbs yang diterima dalam RegFormReq, isi BrandCRLIdentifier.
   (e) Jika formulir registrasi CH tersedia untuk PAN, Language, dan RequestType yang dimaksud, buat RegFormData dengan cara:
      i. isi RegTemplate dan PolicyText yang sesuai dengan RequestType, PAN, dan Language,
         A. sertakan RegFormID dan RegFieldSeq. RegFieldSeq bisa dilalui dalam kasus pembaruan.
         B. sertakan (*optional*) URL untuk menampilkan Logo Merk atau Kartu.
      ii. CertReq akan dienkrip dengan kunci yang berbeda dengan yang digunakan untuk mengenkrip RegFormReq,
      iii. isi CAEThumb dengan *thumbprint* yang berbeda dengan yang dikirimkan dalam CardCInitRes.
   (f) Jika formulir registrasi CH yang sesuai tidak tersedia, isi ReferralData dengan cara:
      i. isi Reason dengan informasi penolakan layanan yang akan ditampilkan ke CH, dan

ii. isi (*optional*) ReferralLoc dengan alamat e-mail dan/atau URL untuk informasi lebih detail tentang penolakan layanan ini, bagi kepentingan user.

2. Tanda tangani hasil langkah pertama. Set contentType SignedData ke id-set-content-RegFormTBS.
3. Jalankan *Compose Message Wrapper* untuk mengirimkan pesan RegFormRes ke CH.

Aplikasi CH memproses RegFormRes dengan langkah-langkah:

1. Terima pesan RegFormRes dari *Receive Message*.
2. Periksa tanda tangan. Jika tidak valid, kembalikan suatu *Error Message* dengan ErrorCode diset ke signatureFailure.
3. Ambil RRPID, RequestType, LID-EE, Chall-EE2, CAEThumb dari "RegFormTBS".
4. Periksa bahwa RRPID sama dengan yang diterima dalam pembungkus pesan yang dikirim dalam RegFormReq. Jika tidak sama, kembalikan suatu *Error Message* dengan ErrorCode diset ke unknownRRPID.
5. Periksa bahwa RequestType, LID-EE, dan Chall-EE2 sama dengan yang dikirim dalam RegFormReq. Jika tidak sama, kembalikan suatu *Error Message* dengan ErrorCode diset ke challengeMismatch.
6. Jika CAEThumb disertakan, simpan sertifikat Enkripsi yang terkait untuk enkripsi CertReq.
7. Periksa bahwa Thumbs yang diterima cocok dengan yang dikirimkan dalam pesan CardCInitReq. Jika tidak cocok, kembalikan suatu *Error Message* dengan ErrorCode diset ke thumbsMismatch.
8. Jika RegFormData disertakan dalam data "RegFormTBS":
   (a) Simpan LID-CA.
   (b) Tampilkan teks kebijakan dan minta konfirmasi user sebelum aplikasi SET membuat CertReq.
   (c) Tampilkan *field* yang bisa dilihat dalam formulir registrasi dan minta user mengisinya.
   (d) Jika RegFormRes berisi URL, tampilkan Logo Merk dan/atau Kartu.
   (e) Isi semua *field* invisible dalam formulir registrasi. Jika suatu *field invisible* dan dibutuhkan dan aplikasi tidak bisa mengisi *field* itu, *field* harus dibiarkan kosong dan sisa formulir registrasi harus diisi dan dikirimkan dalam CertReq seperti dispesifikasikan.
   (f) Sesudah user menyelesaikan formulir registrasi buat suatu CertReq.
9. Jika ReferralData disertakan dalam data "RegFormResTBS":
   (a) Tampilkan Reason.

RegFormRes	S(CA, RegFormResTBS)
RegFormResTBS	{RRPID, LID-EE, Chall-EE2, [LID-CA], Chall-CA, [CAEThumb],
	RequestType, RegFormOrReferral, [BrandCRLIdentifier], [Thumbs]}
RRPID	ID pasangan permintaan/jawaban.
LID-EE	Di-*copy* dari RegFormReq.
Chall-EE2	Di-*copy* dari RegFormReq.
LID-CA	ID lokal, dibuat oleh dan untuk sistem CA (mungkin merupakan nilai baru).
Chall-CA	Challenge CA untuk mengetahui usia tanda tangan CH.
CAEThumb	Thumbprint dari sertifikat pertukaran kunci CAyang akan digunakan untuk mengenkrip CertReq; jika *field* ini tidak ada, sertifikat yang digunakan adalah yang diidentifikasikan dalam CardCInitRes.
RequestType	
RegFormOrReferal	Lihat di bawah.
BrandCRLIdentifier	BCI.
Thumbs	Di-*copy* dari RegFormReq.

RegFormOrReferral	< RegFormData, ReferralData >
RegFormData	{[RegTemplate], PolicyText}
ReferralData	{[Reason], [ReferralURLSeq]}
RegTemplate	{regFormID, [BrandLogoURL], [CardLogoURL], RegFieldSeq}
PolicyText	Pernyataan yang akan ditampilkan bersama dengan RegTemplate pada sistem CH.
Reason	Pernyataan mengenai permintaan yang akan ditampilakan pada sistem CH.
ReferralURLSeq	{ReferralURL +} URL *optional* menunjuk ke informasi referensi, terdaftar di dalam urutan relevansi.
RegFormID	*identifier* yang ditentukan CA.
BrandLogoURL	URL untuk logo merk kartu pembayaran.
CardLogoURL	URL untuk logo institusi keuangan.
RegFieldSeq	{RegField +}
ReferralURL	URL alternatif untuk CA yang sama untuk pemrosesan permintaan sertifikat untuk CH ini.
RegField	{[FieldId], FieldName, [FieldDesc], [FieldLen], FieldRequired, FieldInvisible}
FieldID	Object Identifier
FieldName	Satu atau lebih nama *field* untuk ditampilkan sebagai label untuk formulir pengisian pada sistem CH. teks dalam bahasa yang dispesifikasikan dalam RegFormReq.
FieldDesc	Deskripsi isi *field* dalam bahasa yang dispesifikasikan pada RegFormReq, berisi informasi tambahan yang digunakan bila user meminta penjelasan tentang mengisi formulir.
FieldLen	Panajang maksimum *field*.
FieldRequired	Nilai boolean menandakan data harus ada (diisi CH atau diisi aplikasi).
FieldInvisible	Nilai booleanaa menandakan *field* akan ditampilkan untuk user, aplikasi akan mengisi dalam FieldValue atau membiarkannya kosong.

Tabel 3.5: RegFormRes dan RegFormOrReferral

(b) Jika ReferralLoc disertakan, tampilkan URL atau alamat e-mail dari ReferralLoc.

(c) Jangan buat CertReq. Protokol harus mengulang CardCInitReq dari awal.

3.6 PEMROSESAN PERMINTAAN DAN PEMBUATAN SERTIFIKAT

Dalam proses ini CH diminta untuk memberikan nomor kartu pembayarannya, tanggal kadaluwarsa, dan informasi lain yang diminta CCA (ada di dalam formulir registrasi). Permintaan Sertifikat (CertReq) akan berisi:

- kunci publik yang baru,
- sertifikat yang sedang diperbarui, jika benar,
- formulir registrasi yang sudah diisi,
- informasi rekening End-Entity (EE),
- kunci rahasia yang akan digunakan CA untuk mengenkrip Jawaban Sertifikat (CertRes),
- dan bilangan referensi dan *challenge* lain.

CertReq dibuat menggunakan pemrosesan EncX atau Enc tergantung pada kehadiran AcctInfo. Jika EE adalah CH, AcctInfo selalu berisi PAN dan EncX selalu digunakan. EncX hanya digunakan jika AcctInfo ada. Jika CertReq dikirim ulang dengan formulir registrasi yang sudah dikoreksi, nilai Chall-EE3 dan RRPID harus dibuat ulang untuk pengiriman ulang CertReq. Langkah-langkah pembuatan CertReq bagi CH adalah:

1. Jika RequestType untuk sertifikat tanda tangan, buat pasangan kunci privat/publik untuk sertifikat tanda tangan.
2. Buat suatu bilangan acak 160-bit, CardSecret.
3. Buat suatu bilangan acak 160-bit, EXNonce.
4. Buat CertReqTBS dengan cara:
   (a) Buat RRPID baru.
   (b) Jika EE menerima RegFormRes, *copy* RequestType dari pesan itu, kalau tidak isi RequestType.
   (c) Isi RequestDate dengan tanggal saat itu.
   (d) *Copy* LID-EE dari pesan sebelumnya. Jika tidak ada, buat yang baru.

(e) Buat Chall-EE3 baru.

(f) *Copy* LID-CA, jika disertakan, dan Chall-CA dari pesan sebelumnya, jika ada.

(g) Isi PAN, CardExpiry, dan CardSecret,

(h) Buat EXNonce.

(i) *Copy* ID RegForm yang dikirimkan dalam RegFormRes.

(j) Jika RegFieldSeq ada dalam RegFormRes, sertakan RegForm yang baru atau terkoreksi.

5. Untuk data ekstra yang dibutuhkan enkapsulasi EncX, ambil data-data berikut.

   (a) Pilih dari ekstensi privat "Tunneling" dalam sertifikat enkripsi kunci CA, suatu algoritma enkripsi yang diinginkan untuk dipakai CA mengenkrip CertRes. Isi ID algoritma dan sebuah kunci dalam CaBackKeyData. Jika tidak ada algoritma yang sesuai, hentikan pemrosesan selanjutnya dan beritahu user.

   (b) Isi kunci publik yang baru dibuat, PublicKeyS dan/atau PublicKeyE, untuk disertifikasi CA.

   (c) Sertakan (*optional*) Thumbs, yang berisi *thumbprint* untuk setiap CRL, sertifikat SET, BCI, dan sertifikat Root yang tersimpan dalam *cache* terpercaya milik CH, jika masing-masing ada.

6. Berikutnya, format data "To Be Extra Encrypted": Isi PAN, CardExpiry, CardSecretCardNonce, dan EXNonce dalam PANData0.

7. Bungkus data menggunakan enkapsulasi EncX, dengan cara:

   (a) Sertakan CertReqTBS sebagai "To Be Signed" data. Set contentType SignedData ke id-set-content-CertReqTBS.

   (b) Hasil langkah ke-5 sebagai data "To Be Extra Encrypted". Enkripsi "ekstra" menggunakan sertifikat CA ditandai CAEThumb dalam CardCInitRes atau RegFormRes, jika salah satu disertakan.

   (c) CertReqTBEX sebagai data "To Be Ordinarily Encrypted". Enkrip CertReqTBEX dan set contentType EnvelopedData ke id-set-content-CertReqTBEX.

8. Jalankan *Compose Message Wrapper* untuk mengirimkan CertReq ke CA.

CA akan memvalidasi CertReq dengan langkah-langkah:

1. Terima pesan CertReq dari *Receive Message*.

(a) Jika RequestType menandakan sertifikat Tanda Tangan baru atau sertifikat Tanda Tangan dan Enkripsi, akan ada satu tanda tangan pada CertReq. Periksa menggunakan kunci publik tanda tangan yang ada dalam PublickKeyS. Jika tidak valid, kembalikan suatu CertRes dengan CertStatusCode diset ke sigValidationFail.

(b) Jika RequestType menandakan pembaruan suatu sertifikat Tanda Tangan atau sertifikat Tanda Tangan dan Enkripsi, akan ada dua tanda tangan (SignerInfo) pada CertReq.

i. Untuk SignerInfo dengan Issuer DN bernilai Null, periksa tanda tangan menggunakan kunci publik tanda tangan yang baru yang terdapat dalam PublicKeyS. Jika tidak sesuai, kembalikan suatu CertRes dengan CertStatusCode diset ke sigValidationFail.

ii. Untuk SignerInfo dengan Issuer DN dan Serial Number sama dengan nilai dalam sertifikat Tanda Tangan yang diperbarui, periksa tanda tangan menggunakan kunci publik dalam sertifikat itu. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke signatureFailure.

(c) Jika RequestType menandakan sertifikat Enkripsi, akan ada satu tanda tangan pada CertReq. Periksa dengan menggunakan kunci publik dari sertifikat Tanda Tangan CH. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke signatureFailure.

2. Dari data "CertReqTBS", simpan RRPID, LID-EE, Chall-EE3, RequestType, LID-CA, Chall-CA, IDData, RegForm, CaBackKeyData, Thumbs, dan sertifikat Tanda Tangan dan/atau Enkripsi yang baru.

3. Periksa bahwa RRPID dan RequestDate cocok dengan yang diterima dalam pembungkus pesan. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke unknownRRPID.

4. Periksa bahwa Chall-CA yang diterima cocok dengan yang dikirim dalam RegFormRes. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke challengeMismatch.

5. Periksa PAN, jika disertakan, menurut kebijakan *Brand*; kalau tidak periksa AcctData. Jika gagal, kembalikan suatu CertRes dengan CertStatusCode diset ke rejectedByIssuer.

6. Jika RequestType menandakan pembaruan, periksa bahwa sertifikat yang sedang diperbarui belum dperbarui sebelumnya (untuk menjamin bahwa sertifikat yang dimaksud tidak diperbarui berulang kali). Jika gagal, kembalikan suatu CertRes dengan CertStatusCode diset ke rejectedByCA.

7. Periksa bahwa RegFormID valid untuk Language, RequestType dan BIN atau PAN yang dimaksud. Jika gagal, kembalikan suatu CertRes dengan CertStatusCode diset ke rejectedByCA.

8. Simpan algoritma dan kunci yang disertakan dalam CABackKeyData untuk digunakan mengenkrip CertRes yang akan diberikan ke CH.

CertReq	<EncX(EE, CA, CertReqData, AcctInfo), Enc(EE, CA, CertReqData)>
CertReqData	{RRPID, LID-EE, Chall-EE3, [LID-CA], [Chall-CA], RequestType, RequestDate, [IDData], RegFormID, [RegForm], [CABackKeyData], PublicKeySorE, [EEThumb], [Thumbs]}
AcctInfo	< PANData0, AcctData >
RRPID	ID pasangan permintaan/jawaban.
LID-EE	Di-*copy* dari RegFormRes.
Chall-EE3	Challenge CH untuk mengetahui usia tanda tangan CA.
LID-CA	Di-*copy* dari RegFormRes.
Chall-CA	Di-*copy* dari RegFormRes.
RequestType	Lihat tabel 3.4.
RequestDate	Tanggal permintaan sertifikat.
IDData	Dibiarkan karena EE adalah CH.
RegFormID	*Identifier* yang diset CA.
RegForm	{RegFormItems +} Semua nama *field* yang di-*copy* dari RegFormRes, sekarang dilengkapi dengan nilai yang sudah diisi.
CABackKeyData	{CAAlgId, CAKey}
PublicKeySorE	{[PublicKeyS], [PublicKeyE]} Kunci publik CH. Paling tidak satu kunci harus dispesifikasikan. User bisa meminta sertifikat tanda tangan, enkripsi, atau keduanya.
EEThumb	Thumbprint sertifikat enkripsi kunci yang sedang diperbarui.
Thumbs	Daftar sertifikat (termasuk Root), CRL, dan BCI yang ada.
PANData0	Lihat tabel 3.7.
AcctData	Lihat tabel 3.8.
RegFormItems	{FieldName, FieldValue}
CAAlgId	*Identifier* algoritma kunci simetrik.
CAKey	Kunci rahasia yang sesuai dengan *identifier* algoritma.
PublicKeyS	Kunci publik tanda tangan yang diminta untuk disertifikasi.
PublicKeyE	Kunci publik enkripsi yang diminta untuk disertifikasi.
FieldName	Satu atau lebih nama *field* yang akan ditampilkan sebagai formulir isian pada sistem peminta, sebagai *field* teks dalam bahasa yang dispesifikasian dalam RegFormReq.
FieldValue	Nilai yang diisi CH.

Tabel 3.6: CertReq

9. Periksa item formulir registrasi yang *invisible*. Jika ada *field invisible* yang dibutuhkan dan tidak diisi dengan benar, kembalikan suatu CertRes dengan CertStatusCode diset ke rejectedByIssuer.

10. Jika semua pengecekan di atas berhasil, periksa item formulir registrasi. Untuk tiap item dalam formulir registrasi, periksa panjang, format, dan tipe karakter semuanya benar. Periksa bahwa *field* yang dibutuhkan terisi. Jika ada kesalahan ditemukan, kembailakn nomor item dan pesan teks menandakan kesalahan dalam urutan FailedItems pada CertRes dengan CertStatusCode diset ke regFormAnswerMalformed.

CertRes berisi salah satu dari sertifikat yang diminta atau status permintaan sertifikat. CertRes akan ditandatangani dan dienkrip bila diinginkan, tergantung data yang disertakan dalam pesan. Jika CertRes berhasil dan ditujukan untkuk CH, pesan dienkrip menggunakan algoritma simetrik yang didukung aplikasi CH dan CA. Jika algoritma enkripsi tidak bisa dinegosiasikan antara aplikasi CH dan CA, permintaan akan ditolak dan status yang sesuai akan diberikan. Jika CertRes berisi

PANData0	{PAN, CardExpiry, CardSecret, EXNonce}
PAN	Primary Account Number (Nomor Rekening Primer), biasanya nomor pada kartu.
CardExpiry	Tanggal kadaluarsa kartu.
CardSecret	Usulan dari CH untuk setengah dari rahasia yang dibagi dalam PANSecret.
EXNonce	Nonce baru untuk menjaga serangan dictionary terhadap PANData0.

Tabel 3.7: PANData0

AcctData	{AcctIdentification, EXNonce}
AcctIdentification	Untuk Merchant dan Acquirer.
EXNonce	Nonce baru untuk menjaga serangan dictionary terhadap AcctIdentification.

Tabel 3.8: AcctData

status untuk CH, pesan ditandatangani tapi tidak di-enkrip.

Jika CertReq otentik, valid, dan CA telah membuat sertifikat menggunakan kunci yang dikirimkan, suatu CertRes dengan status yang lengakap akan diberikan. Jika CertRes ditujukan untuk CH dan menyertakan sebuah kunci (dalam CaBackKeyData) untuk mengenkrip CertRes, Ca akan membuat CertRes sebagai SignedData dalam EnvelopedData dengan melakukan langkah-langkah berikut:

1. Buat "CertResData" dengan cara:
   (a) *Copy* RRPID, LID-EE, Thumbs, dan Chall-EE3 dari CertReq.
   (b) Buat LID-CA, atau *copy* dari CertReq, jika ada.
   (c) Isi (*optional*) URL CardLogo, URL BrandLogo, CardCurrency, dan/atau CardholderMessage (CaMsg).
   (d) Set CertStatusCode ke RequestComplete.
   (e) Buat Nonce-CCA.
   (f) Hitung dan isi *thumbprint* sertifikat EE, CertThumbs.
   (g) Jika BrandCRLIdentifier tidak dispesifikasikan dalam Thumbs yang diterima dalam CertReq, isi BrandCRLIdentifier.
2. Tanda tangani dan bungkus data menggunakan enkapsulasi EncK menggunakan CertResData sebagai data "To Be Signed", dengan cara:
   (a) Tanda tangani data dengan sertifikat Tanda Tangan CA.
   (b) Set contentType SignedData ke id-set-content-CertResData.
   (c) Sertakan sertifikat Tanda Tangan dan/atau Enkripsi milik EE yang baru dan sudah disertifikasi ke dalam bagian sertifikat di SignedData.

(d) Enkrip data yang sudah ditandatangani, menggunakan vektor inisialisasi yang dibuat CA dan algoritma serta kunci yang disebutkan CaBackKeyData dalam CertReq.

(e) Set contentType EncryptedData ke id-set-content-CertResTBE.

3. Jalankan *Compose Message Wrapper* untuk mengirimkan CertRes ke EE.

Jika CA mengembalikan status dalam CertRes, EEMessage disertakan untuk menyampaikan informasi ke EE. Langkah-langkah berikut digunakan untuk membuat CertReq yang ditandatangani:

1. Jika CA telah membuat sertifikat yang akan disertakan dalam CertRes, buat CertResTBS.

2. Jika CA belum membuat sertifikat, contohnya mempunya status selain "*Request Complete*", buat CertResData dengan cara:

   (a) *Copy* LID-EE dan Chall-EE3 dari CertReq.

   (b) Isi (*optional*) EEMessage.

   (c) Isi CertStatusCode.

   (d) Jika CertStatusCode diset ke regFormAnswerMalformed, isi ItemNumbers dan ItemReasons untuk tiap FailedItem dalam formulir registrasi.

3. Tanda tangani data menggunakan opeator S, dengan CertResData sebagai data "To Be Signed". Set tipe muatan SignedData ke id-set-content-CertResData.

4. Jalankan *Compose Message Wrapper* untuk mengirimkan CertRes ke EE.

EE memvalidasi sertifikat baru dengan menjalankan:

1. Terima pesan CertRes dari *Receive Message*. Jika CertRes berisi data yang ditandatangani dalam data terenkripsi, dekrip dan periksa tanda tangan pada CertRes, dengan kebalikan langkah pada pemrosesan EncK.

2. Periksa, sebagaimana dispesifikasikan dalam langkah pemrosesan "Thumbprint", bahwa Thumbs yang diterima cocok dengan yang dikirim dalam pesan CardCInitReq. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke thumbsMismatch.

3. Periksa LID-EE dan Chall-EE cocok dengan yang dikirim dalam CertReq. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke challengeMismatch.

4. Jika CertStatusCode menandakan "*Certificate request complete*":

   (a) Ambil sertifikat baru dari bagian sertifikat di SignedData dan periksa tanda tangannya.

(b) Periksa bahwa CertThumbs yang diterima cocok dengan yang dikirimkan dalam CertReq. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke thumbsMismatch.

(c) Jika ada, ambil CaMsg, tampilkan logo dan CardholderMsg dari CaMsg, dan simpan CardCurrency.

(d) Periksa bahwa kunci publik dalam tiap sertifikat berhubungan dengan kunci privat masing-masing. Jika tidak sesuai, kembalikan suatu *Error Message* dengan ErrorCode diset ke invalidCertificate.

(e) Hitung (Nonce-CCA $\oplus$ CardSecret) untuk mendapatkan PANSecret.

(f) Hitung UniqueCardholderID, HMAC-SHA-1 {{PAN, cardExpiry}, PANSecret}, yaitu *Salted Hash*, dan periksa bahwa hasilnya cocok dengan nilai yang ada di sertifikat.

5. Jika CertStatusCode menandakan "*Malformed Registration Form Items*", ada kesalahan pada beberapa item dalam formulir registrasi. Untuk tiap item yang salah, aplikasi EE harus menampilkan nomor item dan pesan kesalahan yang berhubungan yang dikembalikan dalam CertRes. EE harus dibolehkan untuk mengisi ulang *field-field* itu dan aplikasi EE harus mengirim ulang CertReq sebagai permintaan sertifikat baru.

6. Jika CertStatusCode diset ke requestinProgress atau requestPended, sertifikat bisa diambil melalui permintaan lewat CertInqReq.

7. Jika CertStatusCode menandakan status lainnya, tampilkan EEMessage.

Jika validasi gagal, EE mengirim pesan kesalahan ke CA menandakan kegagalannya.

3.7 OPERATOR DALAM SET

Dalam SET didefinisikan beberapa operator kriptografi yang menerangkan operasi kriptografi yang dilakukan untuk membungkus pesan-pesan dalam SET.

Notasi **S{s, t}** merupakan operator tanda tangan untuk menghasilkan tipe PKCS #7 **SignedData** dengan tuple t ditandatangani oleh entity s. Algoritma default untuk tanda tangan dalam SET adalah RSA dengan *hash* SHA-1. Semua alat bantu dan aplikasi SET harus mengikuti kebijakan "tanda tangan sebelum enkripsi".

Notasi **SO{s, t}** mirip dengan **S{s, t}**, tapi hanya mengembalikan tanda tangan dari pesan yang ditandatangani.

Konstruksi OAEP (*Optimal Asymmetric Encryption Padding*) merupakan suatu teknik konstruksi data untuk menyimpan kunci simetrik DES, *hash* data yang akan dienkrip, ditambah dengan parameter tambahan. Konstruksi ini kemudian akan dienkrip menggunakan kunci publik penerima pesan.

CertRes	<S(CA, CertResData), EncK(CABackKeyData, CA, CertResData)> Versi Enck pesan ini hanya dibutuhkan jika komponen CAMsg disertakan dalam CertRes dan hanya digunakan jikaa CaBackKeyData disertakan dalam CertReq.
CertResData	{RRPID, LID-EE, Chall-EE3, LID-CA, CertStatus, [CertThumbs], [BrandCRLIdentifier], Thumbs]}
CABackKeyData	Di-*copy* dari CertReq.
RRPID	ID pasangan permintaan/jawaban.
LID-EE	Di-*copy* dari CertReq.
Chall-EE3	Di-*copy* dari CertReq.
LID-CA	Di-*copy* dari CertReq. Jika tidak ada dalam CertReq, buat nilai baru.
CertStatus	{CertStatusCode, [Nonce-CCA], [EEMessage], [CaMsg], [FailedItem-Seq]}
CertThumbs	Jika permintaan komplit, thumbprint dari sertifikat signature dan/atau enkripsi.
BrandCRLIdentifier	
Thumbs	Di-*copy* dari CertReq.
CertStatusCode	Kode yang menandakan status permintaan sertifikat.
Nonce-CCA	Jika permintaan komplit dan dari CH, merupakan setengah bagian dari rahasia yang dibagi bersama antara CH dan CA. Ada hanya jika EE adalah CH.
EEMessage	Pesan dalam bahasa natural yang akan ditampilkan pada sistem EE.
CAMsg	{[CardLogoURL], [BrandLogoURL], [CardCurrency], [CardholderMsg]} Jika permintaan komplit dan dari CH.
FailedItemSeq	{FailedItem+}
CardLogoURL	URL yang menunjuk grafik logo merk.
BrandLogoURL	URL yang menunjuk grafik logo merk kartu pembayaran.
CardCurrency	Mata uang penagihan CH.
CardholderMsg	Pesan dalam bahasa natural CH yang akan ditampilan perangkat lunak.
FailedItem	{ItemNumber, ItemReason}
ItemNumber	Menandakan posisi item yang gagal dalam daftar *field* registrasi. Nilai 0 menandakan *field* AcctData
ItemReason	Alasan kegagalan, sebagai *field* teks dalam bahasa yang dispesifikasikan.

Tabel 3.9: CertRes

Tujuan penyampaian kunci simetrik menggunakan konstruksi ini adalah untuk menyulitkan pendeteksian kunci simetrik DES oleh pihak ketiga dan menjaga integritas data.

SET menggunakan Bellare-Rogaway metode *Optimal Asymmetric Encryption Padding* (OAEP) sehubungan dengan operator enkapsulasi kriptografiknya. Sebagai tambahan SET menggunakan teknik *hash* yang dikembangkan Matyas dan Johnson untuk memperkuat konstruksi dasar OAEP Bellare-Rogaway. Walaupun OAEP tidak berhubungan langsung dengan prosesan pembungkusan digital, alat bantu dan aplikasi SET harus menerapkan OAEP pada enkripsi kunci DES dan data tambahan menggunakan kunci publik penerima.

Notasi **E{r, t}** adalah operator enkripsi asimetrik. Operasinya adalah enkrip t dengankunci simetrik baru k (Digital Envelope), kemudian masukan ke dalam PKCS #7 **EnvelopedData** untuk entitas r dengan OAEP ; yaitu dengan enkrip OAEP(k) menggunakan kunci publik entitas r, diambil dari sertifikat dalam tuple r. Algoritma standar untuk enripsi simetrik adalah DES.

Notasi **EH{r, t}** adalah operator enkripsi integritas. Operator ini seperti operator E, tapi PKCS #7 **EnvelopedData** berisi OAEP({k, H(t)}) untuk menjamin integritas bila tanda tangan tidak tersedia. Perangkat lunak yang memproses harus melakukan *hash* terhadap t dan mencocokkan terhadapap H(t) dalam PKCS #7 **EnvelopedData**.

Notasi **EX{r, t, p}** adalah operator enkripsi ekstra. Operator ini seperti E, kecuali t dan p merupakan bagian dari pesan yang terbagi dua. t adalah tuple yang dihubungkan ke p dan subjek untuk enkripsi simetrik biasa, dan p adalah parameter, atau bagian dari pemrosesan subjek "ekstra". OAEP({k,p}) dimasukan ke dalam PKCS #7 **EnvelopedData** untuk entitas r. Slot t dinamakan slot EX biasa, dan slot p dinamakan slot EX tambahan.

Notasi **EXH{r, t, p}** adalah operator enkripsi ekstra dengan integritas. Operator ini mirip dengan EX, perbedaannya yaitu OAEP({k, H(t), p}) berada dalam PKCS #7 **EnvelopedData** dan keharusan perangkat lunak memeriksa H(t) seperti pada EH.

Notasi **EK{k, t}** adalah operator enkripsi simetrik dengan kunci k. Operasi ini melakukan enkripsi simetrik terhadap tuple t dengan kunci rahasia k, hasilnya disimpan ke *instance* dari PKCS #7 EncryptedData.

Notasi **Enc{s, r, t}** adalah operator enkapsulasi sederhana dengan tanda tangan. Operasinya adalah tandatangani pesan, kemudian enkrip. **Enc{s, r, t}** mempunyai arti yang sama dengan **E{r, S{s,t}}**.

Notasi **EncK{k, s, t}** adalah operator enkapsulasi sederhana dengan tanda tangan dan kunci yang disediakan. Pesan t yang sudah ditandatangani s dienkrip menggunakan kunci rahasia k yang sudah diketahui. **Enck{k, s, t}** mempunyai arti yang sama dengan **EK{k, S{s, t}}**.

Notasi **EncX{s, r, t, p}** adalah operator enkapsulasi ekstra dengan tanda tangan dua bagian. Pesan dienkrip dengan bagian pertama menggunakan enkripsi biasa dalam slot E dan bagian kedua dalam

slot ekstra E (OAEP). **EncX{s, r, t, p}** mempunyai arti yang sama dengan **E{r,{t, SO{s,{t,p}}}}**.

3.8 PKCS #7

Untuk menyimpan berbagai hasil operasi kriptografi diperlukan format khusus yang mampu menangani informasi yang diperlukan untuk keperluan pembuatan dan pemeriksaannya. Obyek-obyek yang berfungsi untuk menyimpan hasil operasi kriptografi ini dijelaskan dalam standar PKCS #7 [PKCS7], dan yang dipakai dalam SET dari standar itu adalah obyek SignedData, EncryptedData, dan EnvelopedData.

SignedData merupakan obyek untuk menyimpan informasi mengenai suatu data yang ditandatangani suatu pihak penandatangan. Obyek ini menyimpan obyek yang ditandatangani, tandatangannya, serta informasi mengenai penandatangan yang diperlukan untuk memeriksa tanda tangan itu, yaitu sertifikat tanda tangan yang berisi kunci publik, daftar sertifikat untuk memeriksa rantai sertifikat, dan CRL yang diperlukan untuk memvalidasi sertifikat, termasuk BrandCRLIdentifier.

EncryptedData merupakan obyek untuk menyimpan suatu data yang dienkrip untuk dipertukarkan, tanpa informasi mengenai pengirim dan penerima. Kunci rahasia diasumsikan diatur lewat manajemen lain.

EnvelopedData merupakan obyek untuk menyimpan suatu data terenkrip yang akan dikirimkan ke pihak lain. Penerima yang dituju bisa lebih dari satu. Kombinasi isi terenkrip dan kunci enkripsi terenkrip untuk penerima merupakan "amplop" digital untuk penerima.

Bab 4

ANALISA DAN PERANCANGAN

Bab ini membahas analisa dan perancangan terhadap spesifikasi SET yang sudah dijelaskan pada bab III. Analisa dan perancangan di sini meliputi analisa pesan-pesan yang dipakai dalam permintaan pembuatan sertifikat CH ke CA, dan perancangan yang diperlukan untuk membuat pesan-pesan itu.

4.1 PERANGKAT LUNAK

SET mendefinisikan struktur pesan yang dipakai dalam format ASN.1. Format ini mirip sekali dengan bahasa pemrograman berorientasi obyek (C++ atau Java).

Setelah mempelajari *Java Software Development Kit* (SDK) atau yang lebih dikenal sebagai *Java Developer's Kit* (JDK) lebih lanjut, ternyata sudah ada ekstensi kriptografi yang disediakan Java. Kemudian penulis menemukan *library* kriptografi yang memenuhi standar *Java Cryptography Extensions* dari Sun, yaitu *library* JCE-ABA dari *Australian Business Association* (ABA), dan *library* Cryptix dari Cryptix. Ternyata implementasi JCE dari ABA sudah meliputi JDK versi 1.0, 1.1, dan 1.2, sedangkan implementasi dari Cryptix baru meliputi JDK versi 1.0 dan 1.1. Karena JDK 1.2 dianggap akan lebih digunakan di masa depan, maka *library* yang digunakan adalah *library* dari ABA.

Dalam JDK 1.2 sudah ada implementasi untuk pengkodean pesan yang formatnya menggunakan ASN.1 dan pengkodeannya dengan DER. Implementasi dari JDK 1.2 ini sudah meliputi pembuatan sertifikat digital dan ekstensi-nya, serta pemeriksaan sertifikat sesuai dengan standar X.509. Ekstensi yang dicakup dalam JDK 1.2 ini belum meliputi ekstensi yang didefinisikan sebagai ekstensi privat SET, sehingga harus membuat ekstensi privat SET ini untuk ditambahkan ke dalam sertifikat X.509. Ekstensi yang dibutuhkan dalam protokol ini hanyalah ekstensi CertificateType, sesuai dengan spesifikasi SET pada [VIMA97b] halaman 240.

4.2 RINGKASAN PROTOKOL PERMINTAAN SERTIFIKAT

Berdasarkan spesifikasi SET yang dijelaskan pada 3, protokol permintaan sertifikat Cardholder dapat digambarkan seperti pada gambar 4.1.

Dalam gambar 4.1 terdapat enam pesan yang dipertukarkan antara CH dengan CA. Pesan-pesan itu adalah Inisiasi Permintaan, Inisiasi Jawaban, Permintaan Formulir Registrasi, Permintaan Sertifikat Cardholder, dan Sertifikat Cardholder. Masing-masing disimpan dalam obyek CardCInitReq, CardCInitRes, RegFormReq, RegFormRes, CertReq, dan CertRes. Kesemua pesan dibungkus dalam obyek MessageWrapper.

4.3 PEMBUATAN *CardCInitReq*

Untuk membuat CardCInitReq, tipe ASN.1-nya harus diketahui dulu. Dari [VIMA97c] CardCInitReq mempunyai tipe ASN.1 sebagai berikut:

```
486 CardCInitReq ::= SEQUENCE {
487     rrpid       RRPID,
488     lid-EE      LocalID,
489     chall-EE    Challenge,
490     brandID     BrandID,
491     thumbs      [0] EXPLICIT Thumbs OPTIONAL
492 }
```

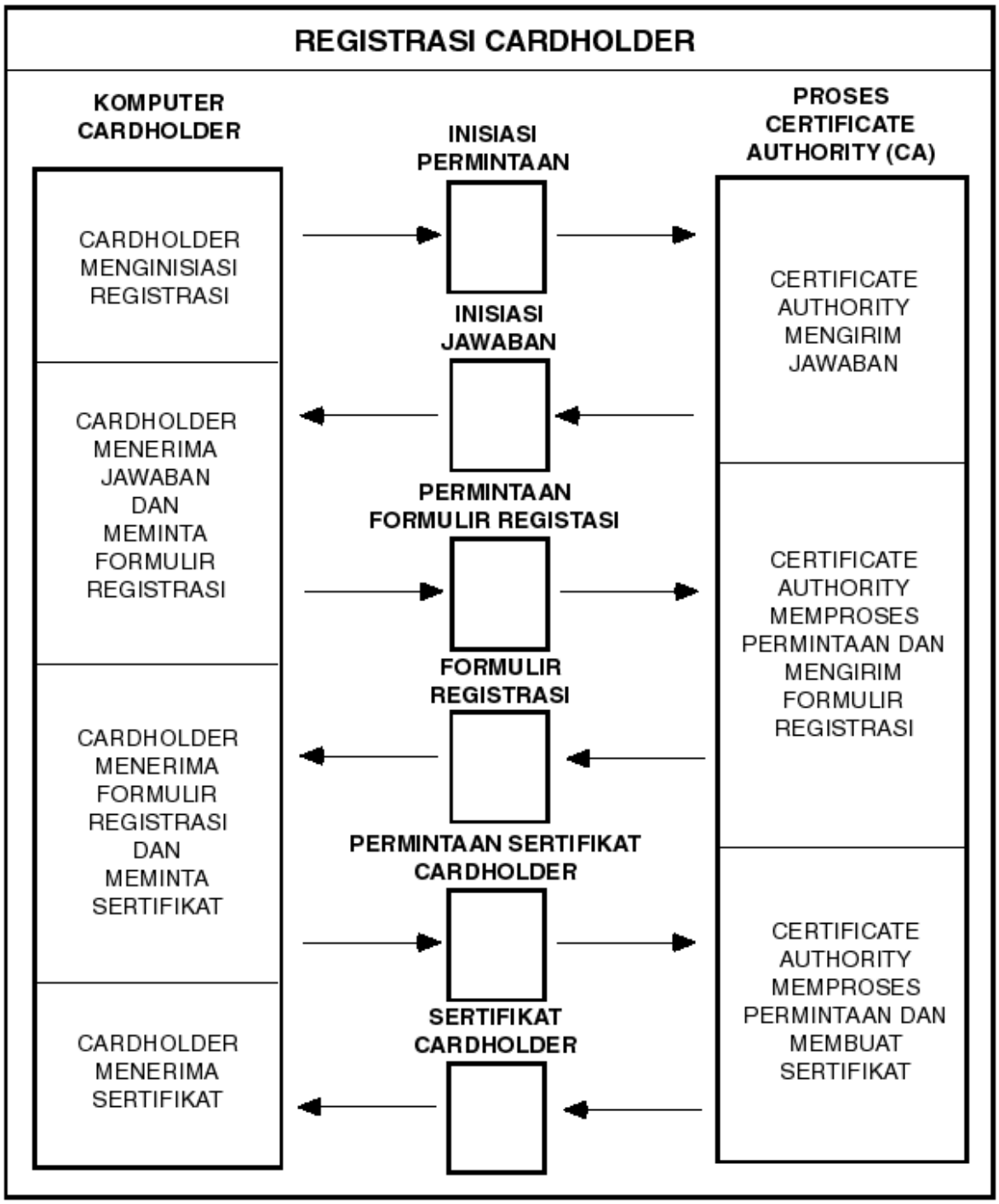
Dalam tipe ASN.1 terdapat angka pada tiap baris di sebelah kiri struktur. Angka itu menunjukkan nomor baris pada spesifikasi pesan SET dalam format ASN.1.

RRPID dibuat secara acak, digunakan sebagai pengenal/identitas untuk pencocokan dalam pesan-pesan selanjutnya. **LID-EE** dibuat secara acak atau berurutan, tapi tidak boleh diulang terlalu sering. **BrandID** merupakan pengenal merk kartu pembayaran yang dimiliki CH. **Thumbs** dibuat dari digest kumpulan sertifikat, CRL, dan BCI yang dimiliki CH (bila ada).

CH sebagai konsumen dalam transaksi SET harus sudah memiliki minimum satu kartu pembayaran sebelumnya. Merk kartu pembayaran itu diberikan CH kepada aplikasi, dan diterjemahkan aplikasi menjadi **BrandID** untuk membentuk kelas CardCInitReq. Bila aplikasi CH sudah pernah menyimpan sertifikat, CRL, dan BCI sebelumnya, maka **Thumbs** bisa diisi.

```
232 BrandID ::= SETString { ub-BrandID }


3130 SETString { INTEGER:maxSIZE } ::= CHOICE {
3131     visibleString VisibleString (SIZE(1..maxSIZE)),
```



Gambar 4.1: Registrasi Cardholder

```
3132     bmpString BMPString (SIZE(1..maxSIZE))
3133 }


352 ub-BrandID INTEGER ::= 40


259 Challenge ::= OCTET STRING (SIZE(20)) -- Signature freshness challenge
324 RRPID ::= OCTET STRING(SIZE(20)) -- Request response pair identification
284 LocalID ::= OCTET STRING (SIZE(1..20))


330 Thumbs ::= SEQUENCE {
331     digestAlgorithm   AlgorithmIdentifier {{DigestAlgorithms}},
332     certThumbs        [0] EXPLICIT Digests OPTIONAL,
333     crlThumbs         [1] EXPLICIT Digests OPTIONAL,
334     brandCRLIdThumbs  [2] EXPLICIT Digests OPTIONAL
335 }
```

Dari seksi 3.4 sudah dijelaskan bahwa pada pesan ini tidak ada kriptografi yang perlu dilibatkan, karena itu pesan ini tidak memakai operasi kriptografi. Hal ini disebabkan CH belum mempunyai modal untuk melakukan pertukaran pesan secara aman dengan CA. CH belum memiliki pasangan kunci privat/publik, dan belum memiliki sertifikat digital CA yang berisi kunci publik CA.

4.4 PEMBUATAN *CardCInitRes*

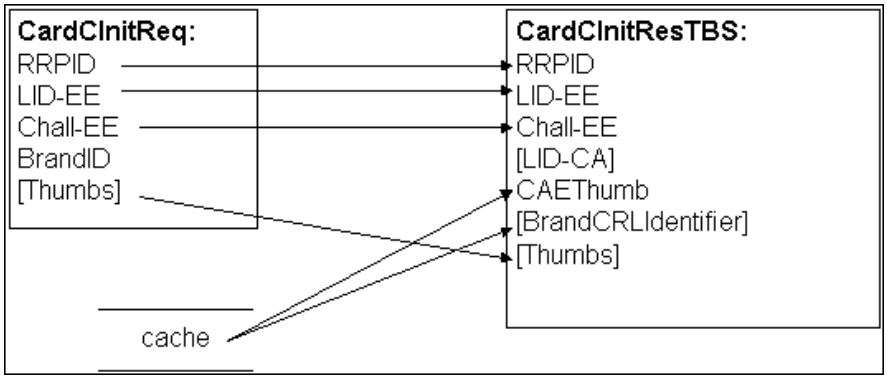
CardCInitRes mempunyai tipe ASN.1 sebagai berikut:

```
494 CardCInitRes ::= S { CA, CardCInitResTBS }
```

Dalam seksi 3.7 telah dijelaskan bahwa operator **S{s, t}**merupakan fungsi penandatanganan dengan nilai kembalian suatu PKCS #7 **SignedData**, dengan s sebagai entitas yang menandatangani pesan t. Dalam pesan **CardCInitRes** ini yang berlaku sebagai s adalah CCA, dan t adalah struktur **CardCInitResTBS**. Di sini, struktur **CardCInitResTBS** dibentuk dulu, kemudian dengan menggunakan operator **S{s, t}** pesan CardCInitResTBS ditandatangani menggunakan kunci privat CA untuk medapatkan pesan **CardCInitRes** yang merupakan salah satu pilihan dari tipe **Message**.

Di dalam CardCInitRes harus disertakan sertifikat, CRL, atau BCI yang tidak diindikasikan dalam Thumbs, yang dibutuhkan CH untuk memeriksa tanda tangan CA, atau mengenkrip RegFormReq dan CertReq.

```
2948 CA ::= ENTITY-IDENTIFIER -- Certifying Authority
```



Gambar 4.2: Pembuatan CardCInitResTBS

```
2942 ENTITY-IDENTIFIER ::= TYPE-IDENTIFIER -- Generic placeholder
```

CardCInitResTBS sendiri struktur ASN.1-nya sebagai berikut:

```
496 CardCInitResTBS ::= SEQUENCE {
497     rrpid               RRPID,
498     lid-EE              LocalID,
499     chall-EE            Challenge,
500     lid-CA              LocalID OPTIONAL,
501     caeThumb            [0] EXPLICIT CertThumb,
502     brandCRLIdentifier  [1] EXPLICIT BrandCRLIdentifier OPTIONAL,
503     thumbs              [2] EXPLICIT Thumbs OPTIONAL
504 }
```

Untuk membuat CardCInitResTBS CA harus mengisi rrpid, lid-EE, chall-EE dari nilai yang diterima pada CardCInitRes, dan membuat lid-CA (tidak harus). CAEThumb diisi dengan *thumbprint* dari sertifikat enkripsi data milik CA. brandCRLIdentifier diisi dengan data BrandCRLIdentifier yang dimiliki CA, jika CardCInitReq yang bersesuaian tidak menyertakan BrandCRLIdentifier. Jika CardCInitReq sudah menyertakan BrandCRLIdentifier, *copy* dari CardCInitReq. Thumbs di-*copy* dari CardCInitReq. Gambar 4.2 menerangkan pembuatan CardCInitResTBS.

```
254 CertThumb ::= SEQUENCE {
255     digestAlgorithm     AlgorithmIdentifier {{DigestAlgorithms}},
256     thumbprint          Digest
257 }


191 BrandCRLIdentifier ::= SIGNED {
192     EncodedBrandCRLID
```

```
193 } ( CONSTRAINED BY { -- Verify Or Sign UnsignedBrandCRLIdentifier -- } )

195 EncodedBrandCRLID ::= TYPE-IDENTIFIER.&Type (UnsignedBrandCRLIdentifier)

197 UnsignedBrandCRLIdentifier ::= SEQUENCE {
198     version             INTEGER { bVer1(0) } (bVer1),
199     sequenceNum         INTEGER (0..MAX),
200     brandID             BrandID,
201     notBefore           GeneralizedTime,
202     notAfter            GeneralizedTime,
203     crlIdentifierSeq    [0] CRLIdentifierSeq OPTIONAL,
204     bCRLExtensions      [1] Extensions OPTIONAL
205 }
```

Struktur CardCInitResTBS, CertThumb, dan UnsignedBrandCRLIdentifier harus memiliki kelas sendiri dalam implementasi, karena merupakan struktur dengan tipe SEQUENCE. BrandCRLIdentifier merupakan *instance* dari kelas SIGNED, sedangkan CardCInitRes merupakan kelas yang memiliki obyek PKCS #7 SignedData di dalamnya.

Sejak tahap ini, pertukaran pesan untuk pembuatan sertifikat CH mulai menggunakan teknik kriptografi untuk mengamankan pesan. Pesan CardCInitRes ini mempunyai tingkat keamanan paling rendah karena hanya melibatkan tanda tangan digital untuk menjamin keaslian pesan. Selain itu data yang dikandung dalam pesan ini tidak penting untuk dirahasiakan, tapi harus dijaga keaslian-nya. Ditambah dengan entitas yang dituju, CH, belum memliki pasangan kunci privat/publik untuk melakukan pengamanan yang lebih tinggi terhadap pesan.

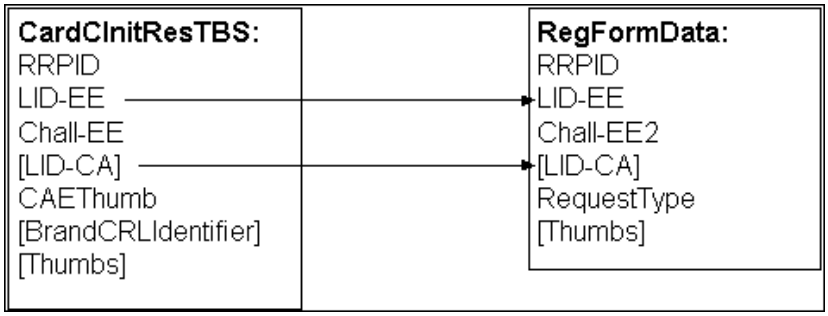
4.5 PEMBUATAN *RegFormReq*

RegFormReq memiliki tipe ASN.1 sebagai berikut:

```
537 RegFormReq ::= EXH { CA, RegFormReqData, PANOnly }
```

Operator **EXH{r, t, p}** sudah diterangkan dalam seksi 3.7. EXH adalah enkripsi ekstra dengan integritas. Enkripsi ini dilakukan karena CH belum memiliki sertifikat untuk menandatangani pesan yang akan dikirim ke CA. Enkripsi dilakukan menggunakan kunci publik enkripsi yang terdapat dalam sertifikat bersesuaian yang disertakan dalam CardCInitRes.

RegFormReqData adalah data yang akan dikirimkan ke CA, dengan data tambahan PANOnly. PANOnly ini berisi data *Personal Account Number* (PAN), dan *nonce*-nya. Gambar 4.3 menerangkan pembuatan RegFormData berdasarkan pesan CardCInitRes yang diterima sebelumnya.



Gambar 4.3: Pembuatan RegFormData.

Setelah CA menerima RegFormReq ini, dia harus mendekripsi pesan menggunakan kunci privatnya dan memeriksa integritas dengan mencocokkan *hash* dari RegFormData dengan *hash* yang disimpan dalam PKCS #7 EnvelopedData.

Struktur ASN.1 RegFormReqData adalah sebagai berikut:

```
542 RegFormReqData ::= SEQUENCE {
543     rrpid           RRPID,
544     lid-EE          LocalID,
545     chall-EE2       Challenge,
546     lid-CA          [0] LocalID OPTIONAL,
548     requestType     RequestType,
549     language        Language,
550     thumbs          [1] EXPLICIT Thumbs OPTIONAL
551 }
```

language memiliki tipe language yang didefinisikan dalam aturan ISO 639. **requestType** memiliki tipe RequestType, sebagaimana didefinisikan dalam tabel 3.4.

```
552 PANOnly ::= SEQUENCE {
553     pan       PAN,
554     exNonce   Nonce
555 }


296 Nonce ::= OCTET STRING (SIZE(20))


298 PAN ::= NumericString (SIZE(1..19))
```

Pesan RegFormReq ini menggunakan enkripsi dengan algoritma kunci publik, karena menyimpan data PAN yang dinilai vital. Data PAN ini berisi nomor kartu pembayaran milik CH, yang bila jatuh ke tangan orang yang tidak berhak dapat merugikan CH. Karena itu data PAN ini disertakan

dalam bagian yang akan dienkrip. Kemudian untuk menjamin integritas data maka dipilih operator EXH. Dengan menggunakan operator ini, data PAN dan data RegFormReq dijamin integritasnya karena menggunakan operasi penghubungan, enkripsi, OAEP, dan enkripsi kunci publik, sebagaimana dispesifikasikan dalam [VIMA97b].

4.6 PEMBUATAN *RegFormRes*

Tipe ASN.1 untuk RegFormRes adalah:

```
557 RegFormRes ::= S { CA, RegFormResTBS }
```

Mirip dengan CardCInitRes, RegFormRes menggunakan operator S untuk membentuk pesannya. RegFormRes juga berupa suatu PKCS #7 SignedData, dengan data RegFormResTBS dan penandatangan dilakukan oleh CA.

```
559 RegFormResTBS ::= SEQUENCE {
560     rrpid              RRPID,
561     lid-EE             LocalID,
562     chall-EE2          Challenge,
563     lid-CA             [0] LocalID OPTIONAL,
564     chall-CA           Challenge,
565     caeThumb           [1] EXPLICIT CertThumb OPTIONAL,
566     requestType        RequestType,
567     formOrReferal      RegFormOrReferral,
568     brandCRLIdentifier [2] EXPLICIT BrandCRLIdentifier OPTIONAL,
569     thumbs             [3] EXPLICIT Thumbs OPTIONAL
570 }
```

RegFormOrReferral akan berisi seperti berikut:

```
442 RegFormOrReferral ::= CHOICE {
443     regFormData        [0] RegFormData,
444     referralData       [1] ReferralData
445 }
```

RegFormOrReferral akan berisi regFormData bila formulir registrasi untuk CH tersedia untuk PAN, Language, dan RequestType yang dikirim CH lewat RegFormReq.

```
447 RegFormData ::= SEQUENCE {
448     regTemplate        RegTemplate OPTIONAL,
```

```
449     policy          PolicyText
450 }
```

RegFormReferral akan berisi referralData bila permintaan formulir untuk CH ditolak oleh CA. Alasan penolakan dimasukkan ke dalam tipe Reason, atau dimasukkan ke dalam suatu URL, dan URL itu harus dimasukkan ke dalam ReferralData.

```
470 ReferralData ::= SEQUENCE {
471     reason          Reason OPTIONAL, -- Displayed on requestor's system
472     referralURLSeq  ReferralURLSeq OPTIONAL
473 } ( WITH COMPONENTS { ..., reason PRESENT } |
474         WITH COMPONENTS { ..., referralURLSeq PRESENT } )
```

Operasi kriptografi yang dilakukan terhadap pesan ini hanyalah penandatanganan, karena karakteristik pesan ini sama dengan pesan CardCInitRes. Tidak ada data penting yang dikirimkan, tapi keaslian perlu dijaga.

4.7 PEMBUATAN *CertReq*

CertReq mempunyai tipe ASN.1 :

```
574 CertReq ::= CHOICE {
575     encx    [0] EXPLICIT EncX { EE, CA, CertReqData, AcctInfo },
576     enc     [1] EXPLICIT Enc { EE, CA, CertReqData }
577 }
```

Dalam seksi 3.6 disebutkan bahwa untuk CertReq pada CH selalu menggunakan EncX karena data AcctInfo ada. Karena itu CertReq dalam protokol ini selalu bernilai:

```
CertReq ::= encx [0] EXPLICIT EncX { EE, CA, CertReqData, AcctInfo }
```

Struktur di atas berarti buat tanda tangan terhadap deret CertReqData dengan AcctInfo, menggunakan kunci privat yang dimiliki EE (dalam hal ini CH). Kemudian CertReqData dideretkan dengan tanda tangan yang baru dibuat, menjadi data yang akan dienkrip menggunakan kunci publik yang didapat dari sertifikat milik CA. AcctInfo merupakan parameter dalam definisi EncX, oleh CH diisi dengan data Personal Account Number (PAN) yang berisi nomor rekening CH pada kartu pembayaran yang dimiliki.

```
392 AcctInfo ::= CHOICE {
393     panData0 [0] EXPLICIT PANData0,
```

```
394     acctData [1] EXPLICIT AcctData
395 }
```

CertReqData merupakan data permintaan sertifikat yang dibuat oleh CH. Sebelum membentuk CertReqData, CH harus membuat pasangan kunci privat/publik untuk sertifikat tanda tangan. Kunci privat yang dibuat harus disimpan secara aman mengikuti aturan yang dispesifikasikan dalam PKCS #5 dan PKCS #8. Kunci publik kemudian dimasukkan ke dalam CertReqData bagian publicKeySorE, seperti yang dispesifikasikan berikut:

```
592 CertReqData ::= SEQUENCE {
593     rrpid           RRPID,
594     lid-EE          LocalID,
595     chall-EE3       Challenge,
596     lid-CA          [0] LocalID OPTIONAL,
597     chall-CA        [1] Challenge OPTIONAL,
598     requestType     RequestType,
599     requestDate     Date,
600     idData          [2] EXPLICIT IDData OPTIONAL,
601     regFormID       INTEGER (0..MAX), -- CA assigned identifier
602     regForm         [3] RegForm OPTIONAL,
603     caBackKeyData   [4] EXPLICIT BackKeyData OPTIONAL,
604     publicKeySorE   PublicKeySorE,
605     eeThumb         [5] EXPLICIT CertThumb OPTIONAL,
606     thumbs          [6] EXPLICIT Thumbs OPTIONAL
607 }
```

caBackKeyData adalah kunci simetrik yang akan digunakan CA untuk mengenkrip CertReq.

```
609 RegForm ::= SEQUENCE SIZE(1..ub-FieldList) OF RegFormItems
610
611 RegFormItems ::= SEQUENCE {
612     fieldName       FieldName,
613     fieldValue      FieldValue
614 }
615
616 FieldName ::= SETString { ub-FieldName }
617
618 FieldValue ::= CHOICE {
```

```
619     setString       SETString { ub-FieldValue },
620     octetString     OCTET STRING (SIZE(1..ub-FieldValue))
621 }
```

Pesan CertReq ini menggunakan enkapsulasi EncX untuk pembentukannya. Enkapsulasi ini merupakan salah satu bentuk pengamanan paling baik untuk pengiriman pesan. Informasi yang dikirim ditandatangani dulu oleh pengirim, sebelum kemudian ditambahkan dengan tanda tangannya, dan dienkrip menggunakan kunci publik penerima. Hal ini dimungkinkan karena CH sudah memiliki pasangan kunci privat/publik untuk penandatanganan pesan. Kunci publik inilah yang nantinya akan disimpan dalam sertifikat dan disahkan oleh CA. Pengamanan data dengan enkripsi dilakukan karena data menyimpan caBackKeyData yang merupakan kunci rahasia untuk enkripsi simetrik yang akan digunakan oleh CA untuk membuat CertRes.

4.8 PEMBUATAN *CertRes*

Pesan CertRes memiliki tipe ASN.1 sebagai berikut:

```
635 CertRes ::= CHOICE {
636     certResTBS      [0] EXPLICIT S { CA, CertResData },
637     certResTBSK     [1] EXPLICIT EncK { CAKey, CA, CertResData }
638 }
```

Dalam seksi 3.6 dan [VIMA97b] diterangkan bahwa jika EE adalah CH, maka CertRes dienkapsulasi menggunakan EncK, sehingga pada protokol ini CertRes akan memiliki struktur:

```
CertRes ::= certResTBSK [1] EXPLICIT EncK { CAKey, CA, CertResData }
```

Pada seksi 3.7 telah diterangkan bahwa operator **EncK{k, s, t}** merupakan bentuk lain dari **EK{k, S{s,t}}**. Artinya di sini adalah CA menandatangani **CertResData** dengan kunci privatnya, dan menyimpan hasilnya ke dalam PKCS #7 **SignedData**. Kemudian hasil tanda tangan itu (*instance* dari PKCS #7 SignedData) dienkrip menggunakan kunci rahasia **CAKey**. Kunci **CAKey** ini diambil dari **caBackKeyData** yang didapat dari CertReq. Hasilnya disimpan ke dalam suatu *instance* dari PKCS #7 **EncryptedData**.

```
640 -- Intermediate results of EncK
641 CertResTBE ::= S { CA, CertResData }


643 CertResData ::= SEQUENCE {
644 rrpid RRPID,
```

```
645     lid-EE              LocalID,
646     chall-EE3           Challenge,
647     lid-CA              LocalID,
648     certStatus          CertStatus,
649     certThumbs          [0] EXPLICIT Thumbs OPTIONAL,
650     brandCRLIdentifier  [1] EXPLICIT BrandCRLIdentifier OPTIONAL,
651     thumbs              [2] EXPLICIT Thumbs OPTIONAL
652 }

654 CertStatus ::= SEQUENCE {
655     certStatusCode     CertStatusCode,
656     nonceCCA           [0] Nonce OPTIONAL,
657     eeMessage          SETString { ub-eeMessage } OPTIONAL,
658     caMsg              [1] CAMsg OPTIONAL,
659     failedItemSeq      [2] FailedItemSeq OPTIONAL
660 }
```

Pesan ini perlu dienkrip karena nonceCCA pada CertStatus berisi bagian rahasia yang dibagi antara CH dengan CA dalam PANData0. Karena itu CertResData ditandatangani dulu oleh CA baru dienkrip menggunakan kunci rahasia yang diberikan CH lewat caBackKeyData dalam CertReq.

4.9 PERBANDINGAN KEAMANAN DENGAN PROTOKOL SSL

Saat ini protokol untuk permintaan sertifikat digital untuk konsumen umumnya menggunakan SSL. Dalam protokol ini CH meminta sertifikat digital lewat protokol https. Protokol ini menggunakan teknik kriptografi SSL (Secure Socket Layer). SSL menggunakan enkripsi DES dengan kunci rahasia DES berkekuatan 56 bit untuk penggunaan server di Amerika, dan kunci rahasia DES berkekuatan 40 bit untuk penggunaan server di luar Amerika.

Dalam SSL, komunikasi dilakukan dengan enkrips simetrik DES menggunakan kunci yang dipertukarkan di awal komunikasi. Pertukaran kunci dapat dilakukan menggunakan enkripsi kunci publik RSA dengan kekuatan kunci sebesar 128 bit, atau teknik pertukaran kunci Diffie-Helman, atau teknik pertukaran kunci lainnya. Selama komunikasi dengan session id yang sama, kunci simetrik yang digunakan tetap, atau dapat menggunakan kunci simetrik yang dibuat baru, menggunakan session id yang sama dan parameter komunikasi yang sama.

Dalam SET, komunikasi dilakukan menggunakan pertukaran kunci simetrik DES dengan kekuatan

kunci 56 bit, tapi kunci rahasia ini selalu dibuat baru untuk setiap pesan. Selain itu kunci simetrik DES dienkrip dengan menggunakan teknik enkripsi kunci publik RSA berkekuatan 1024 bit, dan penyampaian kunci simetrik ditempatkan dalam konstruksi OAEP sehingga menjamin integritas data dan menambah keamanan penyampaian kunci simetrik.

Bab 5

IMPLEMENTASI

Bab ini menerangkan implementasi yang dilakukan sehubungan dengan penelitian. Seperti diterangkan dalam Bab 4, implementasi dilakukan dalam bahasa pemrograman Java. Metode implementasi mengikuti gaya implementasi yang dilakukan Sun dalam JDK 1.2. Implementasi kriptografi tidak dilakukan karena menggunakan *library* dari ABA.

5.1 STRUKTUR PAKET

Di dalam Java, bila dalam suatu proyek terdapat banyak kelas, maka kelas-kelas itu dikelompokkan berdasarkan aturan yang diinginkan. Dalam hal ini mengikuti standar JDK, semua kelas yang dibuat dikelompokkan ke dalam paket-paket:

- digsec.set.certificate.x509
- digsec.set.certificate.components
- digsec.set.core.crypto
- digsec.set.core.pkcs
- digsec.set.core.operators
- digsec.set.message

Paket-paket itu mempunyai nama dengan awalan digsec.set, artinya paket ini dibuat untuk penelitian SIG-Digital Security (asal nama digsec) terhadap SET (asal nama set). Kemudian terbagi ke dalam certificate, core, dan message.

Paket digsec.set.certificate berisi kelas-kelas yang berhubungan dengan sertifikat digital, atau pembuatannya. Semua obyek yang dispesifikasikan dalam standar X.509 dikelompokkan ke dalam paket

certificate.x509, sedangkan obyek lain tapi yang masih berhubungan dengan sertifikat dikelompokkan ke dalam paket components.

Paket digsec.set.core berisi kelas-kelas yang merupakan dasar dari pembentukan pesan-pesan dalam SET. Paket ini terbagi ke dalam crypto, pkcs, dan operators. Paket digsec.set.core.crypto berisi kelas-kelas yang digunakan untuk melakukan fungsi kriptografi. Paket digsec.set.core.pkcs berisi kelas-kelas untuk menampung hasil bentukan fungsi kriptografi. Paket digsec.set.core.operators berisi kelas-kelas yang menyimpan operator kriptografi dalam SET, seperti diterangkan pada seksi 3.7. Kelas representasi pesan dalam protokol permintaan sertifikat dimasukkan ke dalam paket digsec.set.message.

5.2 IMPLEMENTASI OBYEK

Obyek yang dibuat ke dalam kelas Java dibuat mengikuti aturan JDK. Di dalam JDK sudah ada berbagai kelas dan *interface* mengenai kriptografi yang tertuang di bawah paket java.security. Di dalam paket java.security dan java.security.cert sudah dibuat kelas untuk mengakses kunci private, kunci publik dalam sertifikat, kelas untuk mengakses fungsi yang disediakan *library* dari pihak lain, kelas untuk membuat sertifikat sesuai dengan standar X.509, dan kelas lainnya. Beberapa di antara kelas itu masih merupakan kelas abstrak, yaitu kelas yang tidak mempunyai implementasi dalam deklarasi metodenya.

Aturan Dasar

Aturan dasar yang harus diikuti dalam implementasi kelas:

- Setiap obyek yang diimplementasikan sebagai kelas mempunyai constructor dengan parameter suatu nilai DER.
- Setiap obyek yang diimplementasikan sebagai kelas mempunya constructor dengan parameter berisi nilai-nilai yang akan disimpan pada variabel kelas-nya.
- Setiap obyek kelas harus bisa dikodekan ke dalam format DER.
- Untuk obyek yang merupakan implementasi CHOICE dari ASN.1-nya, buat *interface* yang mendefinisikan *method* untuk menentukan pilihan yang dipakai.
- Untuk obyek yang merupakan hasil dari operasi kriptografi (seperti S, E, Enc), simpan hasil operasi itu ke dalam kelas variabel yang memiliki tipe bersesuaian.

5.3 IMPLEMENTASI OPERATOR KRIPTOGRAFI

Implementasi operator SET yang diterangkan pada seksi 3.7 dibagi atas beberapa kelas yang merepresentasikan masing-masing operator. Kelas-kelas ini diimplementasikan oleh Arif Priharsanta dan Dwinanda Prayudi. Kelas-kelas itu terdiri dari kelas S, SO, E, EX, EXH, dan EK. Kelas-kelas ini berada dalam paket digsec.set.core.

Hasil dari operator-operator tadi diterangkan dalam standar PKCS #7. Bentukan hasil operator itu berupa tipe SignedData, EncryptedData, dan EnvelopedData. Kelas-kelas yang merepresentasikan tipe tersebut dikumpulkan dalam paket digsec.set.pkcs. Semua kelas yang didefinisikan dalam standar PKCS dikumpulkan dalam paket digsec.set.pkcs.

5.4 MESSAGEWRAPPER

Setiap pesan dalam SET dibungkus dengan suatu obyek bernama **MessageWrapper**. **MessageWrapper** ini mempunyai struktur sebagai berikut:

```
43 MessageWrapper ::= SEQUENCE {
44     messageHeader  MessageHeader,
45     message        [0] EXPLICIT MESSAGE.&Type (Message),
46     mwExtensions   [1] MsgExtensions {{MWExtensionsIOS}} OPTIONAL
47 }
```

MessageHeader merupakan header yang menjelaskan keterangan mengenani pesan seperti versi, revisi, tanggal, pengenal perangkat lunak, pengenal pesan, dan pengenal pasangan pasangan. **MessageHeader** mempunyai struktur ASN.1 sebagai berikut:

```
58 MessageHeader ::= SEQUENCE {
59     version      INTEGER { setVer1(1) } (setVer1),
60     revision     INTEGER (0) DEFAULT 0, -- This is version 1.0
61     date         Date,
62     messageIDs   [0] MessageIDs OPTIONAL,
63     rrpid        [1] RRPID OPTIONAL,
64     swIdent      SWIdent
65 }
```

Tipe **Message** sendiri merupakan pilihan dari dari 32 macam pesan ditambah satu pesan kesalahan. Kelas Message direpresentasikan dalam kelas MessageInterface dan kelas Message. Kelas MessageInterface adalah *interface* yang mengidentifikasikan tipe pesan yang dikandung kelas Message. Kelas

Message berisi *method* untuk mengembalikan nilai DER dari pesan yang dikandung, serta *method* untuk mengembalikan nilai pengenal pesan yang dikandung. Constructor Message mempunyai parameter nilai DER yang akan diubah menjadi pesan yang sesuai dengan pengenalnya.

Kelas Message diterangkan dalam potongan *source-code* berikut:

```
package digsec.set.wrapper;

public class Message {
    private MessageInterface message;

    public Message(DerValue derval)
        throws IOException
    {
      // Buat pesan yang sesuai berdasarkan tag yang terdapat
      // dalam nilai DER yang diberikan.
    }

    public Message(MessageInterface mi)
    {
      // Buat instance dengan kelas variabel bernilai mi.
    }

    public void encode(DerOutputStream derout)
        throws IOException
    {
      // Berdasarkan method yang diimplementasikan berdasarkan
      // interface MessageInterface, tentukan tipe pesan dan
      // kodekan dalam format DER pada keluaran yang diberikan
      // pada parameter.
    }
}
```

5.5 IMPLEMENTASI PESAN *CardCInitReq*

Menurut [VIMA97b] CardCInitReq adalah salah satu pilihan dari Message. Karena itu kelas CardCInitReq harus mengimplementasikan kelas MessageInterface.

Dalam seksi 4.3 sudah disebutkan bahwa CardCInitReq merupakan SEQUENCE dari beberapa tipe. Implementasi dari kelas CardCInitReq diterangkan dalam potongan *source-code* sebagai berikut:

```
public class CardCInitReq implements MessageInterface {
    private byte[] rrpid;      // rrpid
    private byte[] lid_EE;     // lid-EE
    private byte[] chall_EE;   // chall-EE
    private SETString brandID; // brandID
    private Thumbs thumbs;     // thumbs

    public CardCInitReq(DerValue derval)
        throws IOException
    {
      // Bangun CardCInitReq dari suatu nilai DER
    }

    public CardCInitReq(byte[] rrpiddata, byte[] liddata,
            byte[] chaldata, SETString branddata, Thumbs thumbdata)
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter
    }

    public void derEncode(OutputStream out)
        throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }

    public void encode(DerOutputStream derout)
        throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }

    public int getType()
    {
      // Implementasi dari interface MessageInterface untuk menentukan
      // tipe pesan, dalam hal ini tipenya adalah CardCInitReq.
    }
```

```
}
```

5.6 IMPLEMENTASI PESAN *CardCInitRes*

Pada seksi 4.4 telah disebutkan bahwa CardCInitRes menyimpan hasil operasi S{ CA, CardCInitResTBS } ke dalam suatu obyek bertipe PKCS #7 SignedData. Karena itu kelas CardCInitRes berisi suatu variabel kelas dengan tipe SignedData. Kelas ini mengimplementasikan *interface* MessageInterface. Potongan programnya adalah:

```
public class CardCInitRes implements MessageInterface {
```

```
private SignedData cardCInitRes; // hasil operasi S{CA,CardCInitResTBS}

    // constructors
    public CardCInitRes(DerValue derval)
        throws IOException
    {
      // Bangun CardCInitRes dari suatu nilai DER
    }

    public CardCInitRes(SignedData ccires)
        throws IOException
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter
    }

    public CardCInitRes(PrivateKey key, CardCInitResTBS ccirt,
           X509Certificate[] certs, X509CRL[] crls)
        throws IOException
    {
      // Inisialisasi variabel kelas cardCInitRes dengan membentuk
      // instance SignedData berdasarkan nilai yang diberikan
      // lewat parameter
    }

    // methods
    public void derEncode(OutputStream out)
      throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }

    public void encode(DerOutputStream derout)
        throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya. Nilainya sama dengan nilai format
      // DER dari variabel kelas cardCInitRes yang bertipe SignedData.
```

```
}
```

Sedangkan untuk membuat CardCInitResTBS, prinsipnya sama dengan membuat CardCInitReq.

```
public class CardCInitResTBS {
    private byte[] rrpid;
    private byte[] lid_EE;
    private byte[] chall_EE;
    private byte[] lid_CA;
    private CertThumb caeThumb;
    private BrandCRLId brandCRLIdentifier;
    private Thumbs thumbs;

    // Constructors
    public CardCInitResTBS(byte[] ri, byte[] le, byte[] ce, byte[] lc,
           CertThumb ct, BrandCRLId bci, Thumbs th)
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter.
    }

    public CardCInitResTBS(DerValue derval)
      throws IOException
    {
      // Buat instance CardCInitResTBS berdasarkan suatu nilai DER
      // yang diberikan.
    }

    // methods
     public void encode(DerOutputStream derout)
      throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }
}
```

5.7 IMPLEMENTASI PESAN *RegFormReq*

Pada seksi 4.5 telah disebutkan bahwa RegFormReq merupakan hasil operasi EXH { CA, RegFormReqData, PANOnly }. Hasil dari operasi tadi disimpan ke dalam suatu obyek bertipe PKCS #7 EnvelopedData. Karena RegFormRes merupakan salah satu pilihan dari Message, maka implementasi RegFormReq adalah suatu kelas yang memiliki variabel kelas bertipe EnvelopedData, dan mengimplementasikan *interface* MessageInterface.

Potongan implementasi kelas RegFormReq adalah sebagai berikut:

```
public class RegFormReq implements MessageInterface {
    // kelas variabel
    private EnvelopedData regFormReq;

    // constructors
    public RegFormReq(DerValue derval)
        throws IOException
    {
      // Inisialisasi kelas variabel dari nilai DER yang diberikan.
    }

    public RegFormReq(X509Certificate cert, RegFormData data, PANOnly po)
        throws Exception
    {
      // Inisialisasi kelas variabel dengan membentuk instance
      // EnvelopedData dan simpan ke variabel regFormReq.
    }

    public RegFormReq(PublicKey key, byte[] data, PANOnly po)
        throws Exception
    {
      // Inisialisasi kelas variabel dengan membentuk instance
      // EnvelopedData dan simpan ke variabel regFormReq.
    }

    public void encode(DerOutputStream derout)
        throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }

    public int getType()
    {
      // Implementasi dari interface MessageInterface untuk menentukan
      // tipe pesan, dalam hal ini tipenya adalah CardCInitRes.
    }

    // Tambahkan dengan method untuk mengambil nilai masing-masing kelas
```

```
}
```

5.8 IMPLEMENTASI PESAN *RegFormRes*

RegFormRes mirip dengan CardCInitRes, merupakan hasil operasi S dengan parameter { CA, RegFormResTBS } seperti dijelaskan pada seksi 4.6. Dengan menggunakan kelas S yang dibuat oleh Dwinanda Prayudi, akan didapat hasil operasi tadi yang disimpan ke dalam suatu variabel kelas bertipe SignedData.

```
public class RegFormRes implements MessageInterface {
    private SignedData regFormRes; // hasil operasi S {CA,RegFormResTBS}

    // constructors
    public RegFormRes(DerValue derval)
        throws IOException
    {
      // Bangun RegFormRes dari suatu nilai DER
    }

    public RegFormRes(SignedData rfres)
        throws IOException
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter
    }

    public RegFormRes(PrivateKey key, RegFormResTBS rfrrt,
           X509Certificate[] certs, X509CRL[] crls)
        throws IOException
    {
      // Inisialisasi variabel kelas regFormRes dengan membentuk
      // instance SignedData berdasarkan nilai yang diberikan
      // lewat parameter. Method ini membuat kelas S untuk
      // menghasilkan instance variabel kelas.
    }

    // methods
    public void derEncode(OutputStream out)
      throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }

    public void encode(DerOutputStream derout)
        throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
```

```
}
```

RegFormResTBS mempunyai potongan *source-code* sebagai berikut:

```
public class RegFormResTBS {
    private byte[] rrpid;
    private byte[] lid_EE;
    private byte[] chall_EE2;
    private byte[] lid_CA;
    private byte[] chall_CA;
    private CertThumb caeThumb;
    private int requestType;
    private RegFormOrReferral;
    private BrandCRLId brandCRLIdentifier;
    private Thumbs thumbs;

    // Constructors
    public RegFormResTBS(byte[] ri, byte[] le, byte[] ce2, byte[] lc,
           byte[] cc, CertThumb ct, int rt, RegFormOrReferral ror,
           BrandCRLId bci, Thumbs th)
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter.
    }

    public RegFormResTBS(DerValue derval)
      throws IOException
    {
      // Inisialisasi variabel kelas berdasarkan suatu nilai DER
      // yang diberikan.
    }

    // methods
     public void encode(DerOutputStream derout)
      throws IOException
    {
      // Kodekan nilai variabel kelas pada suatu stream keluaran sesuai
      // dengan struktur ASN.1-nya.
    }
```

```
}
```

5.9 IMPLEMENTASI PESAN *CertReq*

Dalam seksi 4.7 telah disebutkan CertReq dalam protokol permintaan sertifikat ini mempunyai tipe ASN.1:

```
CertReq ::= CHOICE {
    encx     [0] EXPLICIT EncX { EE, CA, CertReqData, AcctInfo },
    enc      [1] EXPLICIT Enc { EE, CA, CertReqData }
}
```

Dari keterangan itu diketahui bahwa implementasi kelas CertReq mempunyai suatu variabel kelas bertipe data PKCS #7 EnvelopedData. Dalam pengkodean format DER-nya harus dikodekan dengan eksplisit bahwa CertReq merupakan tipe ASN.1 yang *constructed* dan *context-specific*.

Potongan source-code kelas CertReq adalah sebagai berikut:

```
public class CertReq implements MessageInterface {
    // variabel untuk menampung hasil enkapsulasi EncX
    private EnvelopedData envData;
    // variabel penunjuk konteks (pilihan yang diambil)
    private int context;
    private CertReqData certReqData;
    private AcctInfo acctInfo;

    public CertReq(DerValue derval) throws IOException
    {
      // Inisialisasi variabel kelas berdasarkan suatu nilai DER
      // yang diberikan.
    }

    public CertReq(PrivateKey eeKey, PublicKey caKey, byte[] data,
           AcctInfo acctinfo)
      throws Exception
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter. Constructor ini dipakai untuk membuat encx.
    }

    public CertReq(PrivateKey eeKey, PublicKey caKey, byte[] data)
      throws Exception
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter. Constructor ini dipakai untuk mebuat enc.
    }

    public CertReqData getCertReqData()
    {
      // Kembalikan nilai CertReqData yang disimpan.
    }

    public AcctInfo getAcctInfo()
    {
      // Kembalikan nilai AcctInfo yang disimpan, bila ada.
    }
```

```
}
```

5.10 IMPLEMENTASI PESAN *CertRes*

Dalam seksi 4.8 telah disebutkan CertRes dalam protokol ini mempunyai tipe ASN.1:

```
CertRes ::= CHOICE {
    certResTBS      [0] EXPLICIT S { CA, CertResData },
    certResTBSK     [1] EXPLICIT EncK { CAKey, CA, CertResData }
}
```

Dari keterangan di atas diketahui bahwa CertRes memiliki dua kemungkinan tempat penyimpanan. Bila CertRes berupa CertResTBS maka CertRes menyimpan suatu obyek SignedData. Bila CertRes berupa CertResTBSK maka CertRes menyimpan suatu obyek EnvelopedData. Potongan *source-code* CertRes adalah sebagai berikut:

```
public class CertRes implements MessageInterface {
    private SignedData certResTBS;
    private EnvelopedData certResTBSK;
    private CertResData certResData

    public CertRes(DerValue derval)
      throws IOException
    {
      // Inisialisasi variabel kelas berdasarkan suatu nilai DER
      // yang diberikan.
    }

    public CertRes(PrivateKey caKey, byte[] crd)
      throws Exception
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter. Constructor ini dipakai untuk membuat CertResTBS.
    }

    public CertRes(Key desKey, PrivateKey caKey, byte[] crd)
      throws Exception
    {
      // Inisialisasi variabel kelas dengan nilai yang diberikan lewat
      // parameter. Constructor ini dipakai untuk membuat CertResTBSK.
    }

    public int getType()
    {
      // Implementasi dari interface MessageInterface untuk menentukan
      // tipe pesan, dalam hal ini tipenya adalah CertRes.
    }

    public CertResData getCertResData()
    {
       // Method untuk mengembalikan nilai CertResData.
    }

    public void encode(DerOutputStream derout)
      throws IOException
```

```
}
```

5.11 IMPLEMENTASI PEMROSESAN PESAN PERMINTAAN

Di dalam spesifikasi SET tidak disebutkan bagaimana aplikasi pada CH menangani suatu permintaan sertifikat. Untuk membuat prototipe harus dibuat suatu aplikasi yang mampu menangani permintaan sertifikat, aplikasi di pihak CH dan di pihak CA. Dari penjelasan sebelumnya bisa dibuat suatu obyek yang merupakan engine untuk membuat permintaan sertifikat ini. Engine ini menjadi suatu kelas yang dinamakan CertificateRequester. Kelas ini berada di aplikasi pada CH. Kelas ini menangani pembuatan dan pengiriman CardCInitReq, RegFormReq, dan CertReq, serta menangani penerimaan dan pemrosesan CardCInitRes, RegFormRes, dan CertRes.

Kelas ini memiliki *method-method* sebagai berikut:

```
public void sendMessage(Message msg)
```

Method ini untuk mengirimkan suatu pesan ke socket yang dispesifikasikan aplikasi.

```
public Message receiveMessage()
```

Method ini untuk menerima pesan dari socket yang dispesifikasikan aplikasi.

```
public CardCInitReq generateCardCInitReq()
```

Method ini untuk membuat pesan CardCInitReq, berinteraksi dengan user bila perlu.

```
public boolean validateCardCInitRes(CardCInitRes cardCInitRes)
```

Method ini untuk memvalidasi pesan CardCInitRes yang diterima lewat receiveMessage, untuk digunakan pada RegFormReq.

```
public RegFormReq generateRegFormReq(Object[] data)
```

Method ini untuk membuat pesan pesan RegFormReq, dengan parameter data untuk membuat instance dari obyek yang diperlukan, berinteraksi dengan user bila perlu.

```
public boolean validateRegFormRes(RegFormRes regFormRes)
```

Method ini untuk memvalidasi pesan RegFormRes yang diterima lewat receiveMessage, untuk digunakan pada CertReq.

```
public CertReq generateCertReq(Object[] data)
```

Method ini untuk membuat pesan CertReq, dengan parameter data untuk membuat instance dari obyek yang diperlukan, berinteraksi dengan user bila perlu.

```
public boolean validateCertRes(CertRes certRes)
```

Method ini untuk memvalidasi pesan CertRes yang diterima lewat receiveMessage, untuk mengambil sertifikat yang diberikan CA.

```
public void requestCertificate()
```

Method ini membungkus *method* lain untuk membentuk proses permohonan sertifikat.

5.12 IMPLEMENTASI PEMROSESAN PESAN JAWABAN

Untuk menjawab permintaan pembuatan sertifikat dari Cardholder, perlu dibuat prototipe aplikasi yang menanganinya. Prototipe ini akan berlaku sebagai aplikasi pada pihak CA yang bertugas menangani pembuatan CardCInitRes, RegFormRes, dan CertRes, serta menangani penerimaan dan pemrosesan CardCInitReq, RegFormReq, dan CertReq.

Dalam implementasinya di dalam Java dibuat kelas CertificateResponder yang memiliki *method-method* sebagai berikut:

```
public sendMessage(Message msg)
```

Method ini untuk menerima pesan dari socket yang dispesifikasikan aplikasi.

```
public Message receiveMessage()
```

Method ini untuk mengirimkan suatu pesan ke socket yang dispesifikasikan aplikasi.

```
public boolean validateCardCInitReq(CardCInitReq cardCInitReq)
```

Method ini untuk memvalidasi pesan CardCInitReq yang diterima lewat receiveMessage, untuk digunakan pada CardCInitRes.

```
public CardCInitRes generateCardCInitRes(Object[] data)
```

Method ini untuk membuat CardCInitRes, dengan parameter data untuk membuat instance dari obyek yang diperlukan.

```
public boolean validateRegFormReq(RegFormReq regFormReq)
```

Method ini untuk memvalidasi pesan RegFormReq yang diterima lewat receiveMessage, untuk digunakan pada RegFormRes.

```
public RegFormRes generateRegFormRes(Object[] data)
```

Method ini untuk membuat pesan RegFormRes, dengan parameter data untuk membuat instance dari obyek yang diperlukan.

```
public boolean validateCertReq(CertReq certReq)
```

Method ini untuk memvalidasi pesan CertReq yang diterima lewat receiveMessage, untuk mengambil sertifikat yang diberikan CA.

```
public CertRes generateCertRes(Object[] data)
```

Method ini untuk membuat pesan CertRes, dengan parameter data untuk membuat instance dari obyek yang diperlukan.

Bab 6

EVALUASI HASIL PENELITIAN

Untuk menguji hasil penelitian, dibuat beberapa skenario yang diharapkan dapat menggambarkan kejadian yang mungkin terjadi pada saat pemakaian objek ini pada aplikasi nyata. Tiap skenario dibuat berdasarkan kemungkinan yang akan terjadi pada saat pembuatan suatu pesan, berdasarkan pesan yang sebelumnya diterima, kecuali pada CardCInitReq yang tidak membutuhkan validasi terhadap pesan lain karena merupakan pesan pertama.

Untuk melakukan uji coba diperlukan pasangan kunci privat/publik milik CA. Kemudian pengujian dilakukan tanpa melakukan validasi terhadap rantai sertifikat, seperti yang dijelaskan pada pembatasan masalah.

6.1 PENGUJIAN PEMBENTUKAN PESAN

Skenario untuk menguji pembentukan pesan adalah sebagai berikut:

1. Bentuk pesan dengan obyek-obyek di dalamnya dengan data-data dummy.
2. Sesudah pesan berhasil dibuat, cetak keterangan mengenai isi pesan menggunakan *method* toString() yang diimplementasikan dalam tiap kelas.
3. Pesan dalam format DER-nya kemudian disimpan ke dalam file.
4. Baca file yang merupakan pesan dalam format DER tadi, dan buat lagi pesannya.
5. Cetak format DER pesan ke layar.
6. Bila pembuatan berhasil, tampilkan lagi ke layar keterangan mengenai isi pesan menggunakan *method* toString().
7. Bandingkan hasil langkah kedua dengan langkah ketujuh. Bila sama berarti pembentukan pesan berhasil, jika tidak sama berarti pembentukan pesan gagal.

Setelah diuji, semua pesan berhasil dibentuk lewat konstruksi obyek-obyek komponennya, dan berhasil dibuat lagi melalui pembacaan format DER-nya dari file.

Contoh pengujian pembuatan pesan berikut menggunakan pesan CardCInitReq. Berikut adalah hasil dari langkah kedua:

```
CardCinitReq:
rrpid: 14:67:E9:F1:C9:DD:E9:DD:44:D8:12:45:C6:4F:7F:3F:F9:33:7D:08
lid_EE: CC:43:7F:BE:B2:EB:20:C6:86:46:BF:0C:A6:19:18:06:E0:99:33:48
chall_EE: 30:EB:A9:3B:6C:F6:6E:B2:1B:2D:2D:BD:A9:AC:B2:1A:2C:04:5B:F2
brandID: VISA
```

Data berikut adalah hasil dari langkah kelima pengujian:

```
DER encoding:
0000: 30 48 04 14 14 67 E9 F1  C9 DD E9 DD 44 D8 12 45  0H...g......D..E
0010: C6 4F 7F 3F F9 33 7D 08  04 14 CC 43 7F BE B2 EB  .O.?.3.....C....
0020: 20 C6 86 46 BF 0C A6 19  18 06 E0 99 33 48 04 14   ..F........3H..
0030: 30 EB A9 3B 6C F6 6E B2  1B 2D 2D BD A9 AC B2 1A  0..;l.n..-.-.....
0040: 2C 04 5B F2 1A 04 56 49  53 41                    ,.[...VISA
```

Data berikut adalah hasil dari langkah ketujuh pengujian:

```
CardCinitReq:
rrpid: 14:67:E9:F1:C9:DD:E9:DD:44:D8:12:45:C6:4F:7F:3F:F9:33:7D:08
lid_EE: CC:43:7F:BE:B2:EB:20:C6:86:46:BF:0C:A6:19:18:06:E0:99:33:48
chall_EE: 30:EB:A9:3B:6C:F6:6E:B2:1B:2D:2D:BD:A9:AC:B2:1A:2C:04:5B:F2
brandID: VISA
```

Dari data di atas terlihat bahwa hasil pembentukan dari langkah kedua dan langkah ketujuh adalah sama. Hal ini menunjukkan bahwa pesan dapat dibuat satu entitas dan dapat dibaca entitas yang lain. Hasil pengujian serupa juga terjadi pada pesan yang lain.

6.2 PENGUJIAN PROSES PERTUKARAN PESAN

Dari spesifikasi protokol permintaan sertifikat digital X.509 Cardholder, dapat disarikan inti dari proses komunikasinya. Pengujian dilakukan terhadap inti proses komunikasi yang merupakan pertukaran pesan antara Cardholder dengan Certificate Authority. Inti dari proses pertukaran pesan itu dapat menjadi skenario untuk uji coba proses, yaitu:

1. Pesan dibuat berdasarkan pembentukan menggunakan komposisi komponen-komponen yang dispesifikasikan dalam spesifikasi SET.

2. Pesan dibuat berdasarkan nilainya dalam format DER.

3. Komponen dari beberapa pesan diambil dari pesan yang sebelumnya diterima, sehingga kedua pesan itu membentuk pasangan pesan. Uji coba dilakukan terhadap pasangan komponen yang cocok, dan yang tidak cocok.

4. Komponen suatu pesan kadang-kadang dibuat berdasarkan nilai dari suatu atau beberapa komponen pesan yang sebelumnya diterima. Setelah komponen pesan yang diterima itu dievaluasi, nilai komponen pesan yang baru dapat diisi ditentukan. Uji coba dilakukan dengan mengkombinasikan komponen pesan yang diterima, untuk menghasilkan berbagai komponen yang harus dipilih berdasarkan pesan itu.

5. Pada beberapa pesan dilakukan operasi kriptografi, menghasilkan bentuk pesan yang tersandikan dan harus disimpan dalam bentuk khusus yang dispesifikasikan dalam standar PKCS #7.

Pengujian proses pertukaran pesan dilakukan berdasarkan hal-hal yang disebut di atas. Tabel 6.1 menjelaskan hasil pengujian yang telah dilakukan sesuai dengan inti pemrosesan tadi yang dianggap sebagai skenario. Untuk skenario 1 dan 2 sudah dijelaskan sebelumnya bahwa pengujian telah berhasil. Pengujian dilakukan dengan mencoba pembuatan oleh entitas yang membuat pesan, dan validasi oleh entitas yang menerima.

Proses	Skenario	Kondisi	Hasil Yang Diinginkan	Hasil Sesuai/ Tidak Sesuai
Pembuatan CardCInitRes	5	Kunci valid	Berhasil	Sesuai
Pembuatan CardCInitRes	5	Kunci tidak valid	Gagal	Sesuai
Validasi CardCInitRes	5	Nilai DER yang diberikan valid	Berhasil	Sesuai
Validasi CardCInitRes	5	Nilai DER yang diberikan tidak valid	Gagal	Sesuai
Validasi CardCInitRes	3	Nilai DER yang diberikan valid	Berhasil	Sesuai
Validasi CardCInitRes	3	Nilai DER yang diberikan tidak valid	Gagal	Sesuai
Pembuatan RegFormReq	3	Komponen yang diberikan valid	Berhasil	Sesuai
Pembuatan RegFormReq	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Pembuatan RegFormReq	5	Parameter valid	Berhasil	Sesuai
Pembuatan RegFormReq	5	Parameter tidak valid	Gagal	Sesuai
Validasi RegFormReq	3	Komponen yang diberikan valid	Berhasil	Sesuai
Validasi RegFormReq	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Validasi RegFormReq	5	Parameter valid	Berhasil	Sesuai
Validasi RegFormReq	5	Parameter tidak valid	Gagal	Sesuai
Pembuatan RegFormRes	3	Komponen yang diberikan valid	Berhasil	Sesuai
Pembuatan RegFormRes	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Pembuatan RegFormRes	4	Menghasilkan RegForm	Berhasil	Sesuai
Pembuatan RegFormRes	4	Menghasilkan ReferralData	Berhasil	Sesuai
Pembuatan RegFormRes	5	Parameter valid	Berhasil	Sesuai
Pembuatan RegFormRes	5	Parameter tidak valid	Gagal	Sesuai
Validasi RegFormRes	3	Komponen yang diberikan valid	Berhasil	Sesuai
Validasi RegFormRes	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Validasi RegFormRes	5	Parameter valid	Berhasil	Sesuai
Validasi RegFormRes	5	Parameter tidak valid	Gagal	Sesuai
Pembuatan CertReq	3	Komponen yang diberikan valid	Berhasil	Sesuai
Pembuatan CertReq	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Pembuatan CertReq	5	Parameter valid	Berhasil	Sesuai
Pembuatan CertReq	5	Parameter tidak valid	Gagal	Sesuai
Validasi CertReq	3	Komponen yang diberikan valid	Berhasil	Sesuai
Validasi CertReq	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Validasi CertReq	5	Parameter valid	Berhasil	Sesuai
Validasi CertReq	5	Parameter tidak valid	Gagal	Sesuai
Pembuatan CertRes	3	Komponen yang diberikan valid	Berhasil	Sesuai
Pembuatan CertRes	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Pembuatan CertRes	4	Menghasilkan sertifikat	Berhasil	Sesuai
Pembuatan CertRes	4	Menghasilkan status kegagalan	Berhasil	Sesuai
Pembuatan CertRes	5	Parameter valid	Berhasil	Sesuai
Pembuatan CertRes	5	Parameter tidak valid	Gagal	Sesuai
Validasi CertRes	3	Komponen yang diberikan valid	Berhasil	Sesuai
Validasi CertRes	3	Komponen yang diberikan tidak valid	Gagal	Sesuai
Validasi CertRes	5	Parameter valid	Berhasil	Sesuai
Validasi CertRes	5	Parameter tidak valid	Gagal	Sesuai

Tabel 6.1: Pengujian Pesan

Bab 7

PENUTUP

Sebagai penutup dari laporan tugas akhir ini dapat diambil beberapa kesimpulan yang ditarik dari tahapan analisa, perancangan, implementasi, dan evaluasi. Selain itu penulis menemukan beberapa saran yang dapat dipakai untuk kelanjutan penelitian.

7.1 KESIMPULAN

Kesimpulan yang diambil dari penelitian ini adalah:

1. Implementasi prototipe proses permintaan sertifikat Cardholder pada protokol SET dapat dilakukan, selama mengikuti spesifikasi dan aturan pendukungnya. Untuk mengerti spesifikasi itu diperlukan pengetahuan tentang kriptografi, format ASN.1 dan aturan pengkodeannya, standar sertifikat digital X.509, serta standar PKCS.

2. Di dalam protokol SET, proses permintaan sertifikat oleh *Cardholder* ke *Certificate Authority* mempunyai enam pesan yang membentuk rantai pesan. Rantai pesan ini terdiri dari CardCInitReq, CardCInitRes, RegFormReq, RegFormRes, CertRes, dan CertRes.

3. Pesan dalam protokol SET ini sedapat mungkin dijaga keamanannya dengan menggunakan operasi kriptografi. Pesan-pesan yang isinya merupakan hasil fungsi kriptografi meliputi CardCInitRes, RegFormReq, RegFormRes, CertReq, dan CertRes.

4. Pada dasarnya tiap pesan dipertukarkan antara dua entitas dalam format DER. Penggunaan format DER ini bisa dijadikan acuan untuk *interoperability* antar berbagai aplikasi yang mengimplementasikan SET.

5. Protokol komunikasi SET dapat dilakukan pada semua media komunikasi elektronik karena penggunaan teknik kriptografi yang mengkombinasikan berbagai teknik kriptografi dan enkapsulasi, sehingga protokol ini mempunyai tingkat keamanan yang sangat tinggi.

6. Proses permintaan sertifikat dalam SET lebih aman dibandingkan dengan proses yang sama menggunakan SSL. Hal ini sudah diterangkan pada seksi 4.9.

7. Implementasi protokol SET ini bisa dilakukan dalam berbagai bahasa pemrograman. Dari tahapan analisa dan implementasi dirasakan bahwa spesifikasi SET dalam ASN.1 mirip dengan bahasa pemrograman yang berorientasi obyek. Dalam tahap implementasi yang menggunakan bahasa pemrograman Java menjadi mudah karena strukturnya mirip. Karena itu implementasi protokol ini juga bisa dilakukan dengan mudah dalam bahasa pemrograman lain yang juga berorientasi obyek.

7.2 SARAN

Setelah melakukan penelitian ini, penulis mempunyai beberapa saran untuk pengembangan lebih lanjut:

1. Penelitian ini dapat digunakan sebagai referensi untuk membuat implementasi permintaan sertifikat digital untuk *Merchant* dan *Payment Gateway*, sesuai dengan spesifikasi SET.

2. Penelitian ini belum melakukan validasi rantai sertifikat, CRL, dan BrandCRLIdentifier. Proses ini juga dapat dijadikan suatu penelitian tersendiri.

3. Implementasi dalam penelitian ini belum teruji *interoperability*-nya dengan aplikasi SET yang dibuat oleh pihak lain. Bila sumber daya memungkinkan, hasil dari prototipe ini sebaiknya diujikan dengan aplikasi SET lain.

LAMPIRAN

DAFTAR KELAS YANG DIBUAT

Paket	Kelas
digsec.set.certificate.components	AcctInfo
	BrandCRLId
	CRLId
	CRLIdSeq
	SETString
	UnsignedBrandCRLId
	Digest
	AcctInfo
digsec.set.message	CardCInitReq
	CardCInitRes
	CardCInitResTBS
	Message
	MessageInterface
	RegFormResTBS.java
	CAThread.java
	CertificateRequester.java
	CertificateResponder.java
	CertReq
	CertReqData
	CertRes
	Message
	MessageInterface
	RegFormReq
	RegFormReqData
	RegFormRes

Bibliografi

[SCHN96] Schneier, Bruce, "Applied Cryptography", *John Wiley & Sons, Inc.*, Second Edition, 1996.

[ITUT97] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1997, *Information technology - Open Systems Interconnection - The Directory: Authentication Framework.*

[LARM99] Larmout, Prof. John, "Complete ASN.1", *Open Systems Solutions*, 1999.

[KALI93] Kaliski Jr., Burton S., "A Layman's Guide to a Subset of ASN.1, BER, and DER", *An RSA Laboratories Technical Note*, 1993.

[VIMA97a] Visa dan Mastercard, "Book 1: Business Description", *SET Secure Electronic Transaction Specification*, 1997.

[VIMA97b] Visa dan Mastercard, "Book 2: Programmer's Guide", *SET Secure Electronic Transaction Specification*, 1997.

[VIMA97c] Visa dan Mastercard, "Book 3: Formal Protocol Definition", *SET Secure Electronic Transaction Specification*, 1997.

[PKCS1] RSA Laboratories, "PKCS #1: RSA Encryption Standard", Version 1.5, November 1993.

[PKCS6] RSA Laboratories, "PKCS #6: Extended-Certificate Syntax Standard", Version 1.5, November 1993.

[PKCS7] RSA Laboratories, "PKCS #7: Cryptographic Message Syntax Standard", Version 1.1, November 1993.