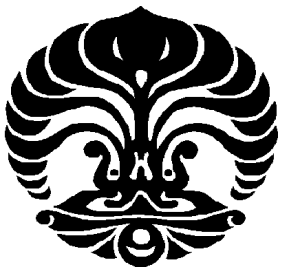
IMPLEMENTASI PROTOTIPE PROSES OTORISASI KARTU PEMBAYARAN ANTARA MERCHANT DAN PAYMENT GATEWAY PADA PROTOKOL SECURE ELECTRONIC TRANSACTION

Disusun sebagai Laporan Tugas Akhir

Oleh :
I. Arif Priharsanta



Fakultas Ilmu Komputer
Universitas Indonesia
Depok
1999

KATA PENGANTAR

Puji dan syukur saya panjatkan kepada Tuhan Yang Maha Kasih, karena berkat dan anugerah-Nya, penulis dapat menyelesaikan tugas akhir dengan judul "Implementasi Prototipe Proses Otorisasi Kartu Pembayaran antara Merchant dan Payment Gateway pada Protokol Secure Electronic Transaction".

Tugas akhir ini merupakan salah satu syarat untuk memperoleh kelulusan untuk mahasiswa Fakultas Ilmu Komputer UI pada jenjang pendidikan tingkat S1.

Selanjutnya penulis sampaikan ucapan terima kasih yang sebesar-besarnya kepada :

1. Ibu dan bapak, yang telah mendidik dan membesarkan penulis dengan penuh kasih sayang dan sangat berharap agar penulis lekas menyelesaikan tugas akhir ini.
2. Bapak Bob Hardian M.Kom, selaku pembimbing utama tugas akhir yang telah memberikan waktu dan perhatian serta arahan dalam mengerjakan tugas akhir ini.
3. Bapak FX. Nursalim Hadi Ph.D, selaku pembimbing awal yang telah membantu menyediakan segala fasilitas untuk memulai tugas akhir ini.
4. Arrianto Mukti Wibowo S.Kom, selaku asisten pembimbing tugas akhir yang telah menyemangati dan memotivasi dalam mengerjakan tugas akhir ini.
5. Bapak Suryana Setiawan M.Sc, selaku pembimbing akademis yang telah membimbing penulis selama menjadi mahasiswa di Fakultas Ilmu Komputer UI .
6. Teman-teman seperjuangan di DIGSEC yang telah meramaikan hari-hari selama pengerjaan tugas akhir ini.
7. Segenap staf, dosen, karyawan dan rekan-rekan mahasiswa di lingkungan Fasilkom UI yang tidak dapat penulis sebutkan satu persatu, atas segala sumbangsih, perhatian dan dukungannya.

Penulis menyadari tugas akhir ini masih banyak kekurangannya sehingga saran dan kritik pembaca merupakan masukan yang sangat berguna. Semoga laporan tugas akhir ini dapat berguna bagi pihak-pihak yang berkepentingan.

Depok, 1999

Penulis

ABSTRAK

Perdagangan elektronik saat ini berkembang sangat pesat di dunia termasuk di Asia. Seiring dengan meningkatnya perdagangan elektronik khususnya di internet, meningkat pula jumlah pengguna kartu pembayaran sebagai alat pembayaran yang paling praktis di internet. Peningkatan tersebut diikuti pula dengan peningkatan jumlah penipuan dan kejahatan di internet.

Secure Electronic Transaction (SET) adalah sebuah protokol yang khusus dibangun untuk menangani keamanan transaksi kartu pembayaran di internet. SET menjamin autentisitas, kerahasiaan dan integritas data transaksi yang dikirimkan melalui internet.

Protokol SET mengatur bagaimana *cardholder* (pemakai kartu pembayaran) dan *merchant* (pedagang) bertransaksi, mengatur bagaimana *merchant* dan *payment gateway* (gerbang pembayaran) melakukan otorisasi kartu pembayaran dan permintaan pembayaran, mengatur bagaimana setiap pihak yang terlibat memiliki suatu sertifikat digital sebagai jaminan atas dirinya.

Tujuan penelitian ini adalah mengimplementasikan prototipe proses otorisasi kartu pembayaran antara *merchant* dan *payment gateway* pada protokol SET.

Penelitian dilakukan dengan mempelajari dan menganalisa spesifikasi SET beserta standar-standar kriptografi pendukungnya, dilanjutkan dengan perancangan dan implementasi dari spesifikasi SET tersebut.

Hasil yang ingin dicapai adalah pembuktian bahwa spesifikasi protokol SET tersebut dapat diimplementasikan khususnya untuk proses otorisasi tersebut di atas.

DAFTAR ISI

KATA PENGANTAR ..... **I**

ABSTRAK ..... **II**

DAFTAR ISI ..... **III**

DAFTAR GAMBAR ..... **V**

DAFTAR TABEL ..... **VIII**

B A B I PENDAHULUAN **1**

1.1 LATAR BELAKANG MASALAH 1
1.2 TUJUAN PENELITIAN 3
1.3 PEMBATASAN MASALAH 3
1.4 METODOLOGI PENELITIAN 3
1.5 SISTEMATIKA PENULISAN 4

B A B I I LANDASAN TEORI SET **5**

1.6 PARTISIPAN SISTEM PEMBAYARAN SET 5
1.7 KRIPTOGRAFI 6
1.7.1 Hash dan Message Digest *7*
1.7.2 Kriptografi Simetris atau Kriptografi Kunci Rahasia *7*
1.7.3 Kriptografi Asimetris atau Kriptografi Kunci Publik *8*
1.7.4 Tanda Tangan Digital *8*
1.7.5 Tanda Tangan Digital Ganda *9*
1.7.6 Amplop Digital *10*
1.7.7 Sertifikat Digital *11*
1.7.8 Ringkasan Kriptografi SET *13*
1.8 PROSES TRANSAKSI PEMBAYARAN 14
2.1.1 Fase Registrasi *16*
1.8.1 Fase Belanja *17*
1.8.2 Fase Otorisasi *21*
1.8.3 Fase Capture *24*
1.9 ASN.1 DAN DER 27
1.9.1 Abstract Syntax Notation One (ASN.1) *28*
1.9.2 Distinguished Encoding Rules (DER) *31*

B A B I I I ANALISA DAN PERANCANGAN 36

1.10 PESAN 38

1.10.1 Objek DER 39

1.10.2 PKCS dan X.509 39

1.10.3 Optimal Asymmetric Encryption Padding (OAEP) 44

1.1.4 Operator Kriptografi SET 48

1.1.5 Komponen Pesan 53

1.1.6 Pasangan Pesan Otorisasi (Authorization Pair) 65

1.1.7 Pembungkus Pesan (Message Wrapper) 74

1.1.8 Pesan Kesalahan (Error) 76

1.2 PROSES 78

1.2.1 Pembuatan Authorization Request 80

1.2.2 Pemrosesan Authorization Request 82

1.2.3 Pembuatan Authorization Response 86

1.2.4 Pemrosesan Authorization Response 90

B A B I V IMPLEMENTASI 93

1.3 PEMILIHAN BAHASA PEMROGRAMAN 93

1.4 TAHAPAN IMPLEMENTASI 94

1.5 IMPLEMENTASI PESAN 94

1.5.1 Karakteristik Objek Pesan SET 94

1.5.2 Contoh Implementasi Sebuah Objek Pesan 99

1.5.3 Pemaketan Objek Pesan 103

1.6 IMPLEMENTASI PROSES 106

B A B V PENGUJIAN 113

1.7 SKENARIO PENGUJIAN 113

1.7.1 Pengujian Objek Pesan 113

1.7.2 Pengujian Proses 113

1.8 HASIL PENGUJIAN 115

1.8.1 Pengujian Objek Pesan Umum 115

1.8.2 Pengujian Proses 118

B A B V I KESIMPULAN DAN SARAN 119

1.9 KESIMPULAN 119

1.10 SARAN PENGEMBANGAN 119

L A M P I R A N A KODE OTORISASI 120

DAFTAR PUSTAKA 121

DAFTAR GAMBAR

Gambar II-1 Kriptografi Simetris 7
Gambar II-2 Kriptografi Asimetris 8
Gambar II-3 Tanda Tangan Digital Ganda 9
Gambar II-4 Hirarki Kepercayaan *Certificate Authority* (CA) 12
Gambar II-5 Ringkasan Kriptografi SET 13
Gambar II-6 Simbol Diagram Transaksi SET 1 15
Gambar II-7 Simbol Diagram Transaksi SET 2 16
Gambar II-8 *Cardholder* membuat *Initiate Request* 17
Gambar II-9 *Merchant* membuat *Initiate Response* 18
Gambar II-10 *Cardholder* membuat *Purchase Request* 19
Gambar II-11 *Merchant* membuat *Purchase Response* 20
Gambar II-12 *Cardholder* menerima *Purchase Response* 21
Gambar II-13 *Merchant* membuat *Authorization Request* 22
Gambar II-14 *Payment Gateway* membuat *Authorization Response* 23
Gambar II-15 Merchant menerima *Authorization Response* 24
Gambar II-16 *Merchant* membuat *Capture Request* 25
Gambar II-17 *Payment Gateway* membuat *Capture Response* 26
Gambar II-18 *Merchant* menerima *Capture Response* 27
Gambar II-19 Pengkodean Oktet *Identifier* (*tag* < 31) 31
Gambar II-20 Pengkodean Oktet *Identifier* (*tag*>=31) 32
Gambar II-21 Bentuk Pendek Pengkodean Panjang 33
Gambar II-22 Bentuk Panjang Pengkodean Panjang 33
Gambar III-1 Transaksi SET 36
Gambar III-2 Struktur Lapisan Objek SET 39
Gambar III-3 Algorithm Identifier 39
Gambar III-4 Content Info 40
Gambar III-5 Issuer And Serial Number 40
Gambar III-6 Digested Data 41
Gambar III-7 Signed Data 41
Gambar III-8 Enveloped Data 43
Gambar III-9 Encrypted Data 44

Gambar III-10 Optimal Asymmetric Encryption Padding (OAEP) 45
Gambar III-11 Hash (H) 48
Gambar III-12 Data Ter-*digest* (DD) 48
Gambar III-13 Koneksi (L) 49
Gambar III-14 Hash Menggunakan Kunci (HMAC) 49
Gambar III-15 Tanda Tangan (SO) 49
Gambar III-16 Pesan Tertandatangani (S) 50
Gambar III-17 Enkripsi Asimetris (E) 50
Gambar III-18 Enkripsi dengan Integritas (EH) 50
Gambar III-19 Enkripsi Ekstra (EX) 51
Gambar III-20 Enkripsi Ekstra dengan Integritas (EXH) 51
Gambar III-21 Enkripsi Simetris (EK) 51
Gambar III-22 Enkapsulasi Sederhana dengan Tanda Tangan (Enc) 51
Gambar III-23 Enkapsulasi Sederhana dengan Tanda Tangan (EncK) 52
Gambar III-24 Enkapsulasi Extra dengan Tanda Tangan (EncX) 52
Gambar III-25 Enkapsulasi dengan Bagasi Eksternal Terenkripsi (EncB) 52
Gambar III-26 Enkapsulasi dengan Bagasi Eksternal Terenkripsi (EncBX) 53
Gambar III-27 TransIDs 53
Gambar III-28 RRTags 54
Gambar III-29 InstallRecurData 55
Gambar III-30 PANData 55
Gambar III-31 PANToken 56
Gambar III-32 Payment Instruction (PI) 56
Gambar III-33 Tree PIUnsigned 57
Gambar III-34 Tree PIDualSigned 58
Gambar III-35 Tree ASN.1 PIHead dan OIData 59
Gambar III-36 Tree AuthToken 60
Gambar III-37 Tree ASN.1 AuthTokenData 60
Gambar III-38 AcqCardMsg 61
Gambar III-39 CapToken 61
Gambar III-40 SaleDetail 62
Gambar III-41 Tree ASN.1 SaleDetail 64
Gambar III-42 Pesan-Pesan dalam SET 65
Gambar III-43 Authorization Request (AuthReq) 66
Gambar III-44 Authorization Response (AuthRes) 69

Gambar III-45 AuthResData 70
Gambar III-46 AuthResBaggage 73
Gambar III-47 Pembungkus Pesan (*Message Wrapper*) 74
Gambar III-48 Pesan (*Message*) 75
Gambar III-49 Pesan Kesalahan (Error) 76
Gambar III-50 Kode Kesalahan (ErrorCode) 77
Gambar III-51 Diagram Konteks Proses Otorisasi 79
Gambar III-52 DAD Tingkat 1 - Proses Otorisasi 79
Gambar III-53 Pembuatan *Authorization Request* 80
Gambar III-54 Pemrosesan *Authorization Request* 82
Gambar III-55 Pembuatan *Authorization Response* (1) 86
Gambar III-56 Pembuatan *Authorization Response* (2) 87
Gambar III-57 Pemrosesan *Authorization Response* 90
Gambar IV-1 Ringkasan Objek DerOutputStream 96
Gambar IV-2 Ringkasan Objek DerInputStream 97
Gambar IV-3 Ringkasan Objek DerValue 98
Gambar IV-4 *Encode* dan *Decode* Objek 99
Gambar IV-5 Komentar Program tentang Spesifikasi TransIDs dalam ASN.1 100
Gambar IV-6 Awal program TransIDs 101
Gambar IV-7 *Encode-Decode* TransIDs 103
Gambar IV-8 *Field-field* pada AuthReq 108
Gambar IV-9 Inisialisasi Data-Data yang Sebelumnya Diterima dari Cardholder 110
Gambar IV-10 Proses-proses yang dibuat mendekati DAD Pembuatan AuthReq 112

DAFTAR TABEL

Tabel II-1 Tabel Inisial Partisipan 15
Tabel II-2 Nomor *Tag Universal* ASN.1 29
Tabel II-3 Bit ke-8 dan ke-7 pada *identifier* 32
Tabel II-4 Pengkodean Integer dengan DER 35
Tabel III-1 OAEP - Actual Data Block 46
Tabel III-2 OAEP - Data Block 46
Tabel III-3 OAEP - A 47
Tabel III-4 OAEP - B 47
Tabel III-5 OAEP – Padded Data Block 47
Tabel III-6 OAEP - R 48
Tabel III-7 Keterangan TransIDs 54
Tabel III-8 Keterangan RRTags 54
Tabel III-9 Keterangan InstallRecurData 55
Tabel III-10 Keterangan PANData 56
Tabel III-11 Tiga Macam Payment Instruction (PI) 57
Tabel III-12 Keterangan SaleDetail 63
Tabel III-13 Keterangan AuthReq 67
Tabel III-14 Keterangan AuthReqPayload 68
Tabel III-15 Keterangan AuthRes 69
Tabel III-16 Keterangan AuthResData 71
Tabel III-17 Keterangan AuthHeader 72
Tabel III-18 Keterangan CapResPayload 72
Tabel III-19 Keterangan AuthResBaggage 73
Tabel III-20 Keterangan ErrorCode 78
Tabel III-21 Keterangan Proses Pembuatan AuthReq 82
Tabel III-22 Keterangan Proses Pemrosesan AuthReq 85
Tabel III-23 Keterangan Proses Pembuatan AuthRes 89
Tabel III-24 Keterangan Proses Pemrosesan AuthRes 92
Tabel IV-1 Tabel Objek-Objek Pesan SET 106
Tabel V-1 Hasil Pengujian Objek Pesan 117
Tabel V-2 Hasil Pengujian Proses 118

B A B I
PENDAHULUAN

1.1 LATAR BELAKANG MASALAH

Perdagangan elektronik saat ini berkembang sangat pesat, dilihat dari nilai investasinya di dunia sudah sangat tinggi. Sebagai gambaran, menurut studi yang dilakukan oleh *University of Texas,* harga pasar perdagangan elektronik di Amerika Utara mencapai 301 miliar dolar AS. Di Eropa Barat menurut *Data Monitor*, harga pasar untuk perdagangan elektronik telah mencapai 775 juta dolar AS dan akan meningkat menjadi 8,6 miliar dolar AS pada tahun 2003. Untuk Asia, yaitu di Jepang menurut *Daily Yomiuri* diperkirakan pada tahun 2003 pendapatan dari perdagangan elektronik bisnis ke konsumen mencapai 1 triliun yen, sekitar 8,2 miliar dolar AS; sedangkan Korea pada tahun 1998 sudah memiliki pasar perdagangan elektronik seharga 20,8 miliar dolar Amerika [Kali93] Burton S. Kaliski Jr.: *A Layman's Guide to a Subset of ASN.1, BER, and DER.;* RSA Laboratories, 1993.

[Nua99].

Sebagai dampak dari pesatnya perkembangan perdagangan elektronik, tidak akan ada satu pun institusi ekonomi yang tidak terpengaruh. Konsumen pengguna kartu pembayaran semakin banyak karena kartu pembayaran adalah cara yang paling efektif untuk melakukan transaksi, begitu pula bank-bank yang menyediakan fasilitas transaksinya.

Berkembangnya perdagangan elektronik juga memicu percobaan-percobaan kejahatan penipuan, terutama perdagangan yang bertransaksi melalui internet. Menurut sumber *National Consumer League*, tujuh persen dari pemakai internet mengalami penipuan kartu kredit. Tujuh persen tersebut senilai enam juta pemakai. Penipuan lewat internet ini menempati posisi penipuan pertama di AS [Kali93] Burton S. Kaliski Jr.: *A Layman's Guide to a Subset of ASN.1, BER, and DER.;* RSA Laboratories, 1993.

[Nua99]. Timbul pertanyaan, kelemahan apa yang dimanfaatkan oleh para penjahat pada sistem perdagangan elektronik di internet?

MasterCard International dan *Visa International* sudah meramalkan bahwa tren penipuan di internet akan semakin berkembang bersamaan dengan perkembangan perdagangan elektronik. Mereka sudah mengetahui bahwa kelemahan sistem pembayaran di Internet terletak pada kerahasiaan, keaslian, dan keutuhan transaksi yang tidak terjamin. Maka sejak 1 Februari 1996, mereka mempublikasikan bahwa mereka sudah mempelopori pengembangan sebuah standar teknis untuk menjamin keamanan transaksi kartu kredit melalui internet. Standar tersebut dikenal dengan nama *Secure Electronic Transaction* (SET). Standar ini selesai dikembangkan dan dikeluarkan versi perdananya pada tanggal 31 Mei 1997.

SET lahir atas tujuh kebutuhan bisnis [Set97a], yaitu:

1. Menjamin kerahasiaan dari informasi pembayaran dan informasi pemesanan.
2. Menjamin keutuhan semua data yang dikirimkan.
3. Menyediakan autentisitas bagi *cardholder* bahwa dia memiliki rekening pada suatu bank untuk kartu pembayaran miliknya.
4. Menyediakan autentisitas bagi pedagang bahwa dia dapat menerima pembayaran dari pemegang kartu melalui relasinya dengan bank tertentu.
5. Memastikan penggunaan teknik keamanan yang terbaik untuk menjaga semua pihak yang sah dalam transaksi perdagangan elektronik.
6. Membuat protokol yang tidak tergantung pada mekanisme keamanan transportasi tertentu namun juga tidak mengabaikannya.
7. Menyediakan fasilitas *interoperability* antar penyedia perangkat lunak dan jaringan komputer.

Sudah banyak *vendor-vendor* asing yang menjual produk SET. Mereka berasal dari Spanyol (ACE), AS(CyberCash, Verifone, IBM, Microsoft), Jerman (Brokat), Jepang (Fujitsu, Hitachi), Korea (Samsung), dsb [Setco99].

Di Indonesia sistem perdagangan elektronik seperti ini masih sangat sedikit diterapkan. Bidang-bidang usaha yang akan lebih dahulu mengarah ke sana adalah bidang usaha yang berorientasi bisnis ke bisnis. Untuk bidang usaha yang berorientasi bisnis ke konsumen mungkin masih lama, sebagai gambaran jumlah konsumen pemakai internet saat ini baru 80.000 orang atau 0,04% dari seluruh jumlah penduduk Indonesia [Kali93] Burton S. Kaliski Jr.: *A Layman's Guide to a Subset of ASN.1, BER, and DER.;* RSA Laboratories, 1993.

[Nua99]. Namun tidak ada salahnya jika teknologinya sudah dipersiapkan dari sekarang sehingga pada saat masyarakat sudah membutuhkan kita mudah-mudahan sudah punya produk SET buatan dalam negeri.

1.2 TUJUAN PENELITIAN

Tujuan dari penelitian ini adalah mengimplementasikan prototipe proses otorisasi pembayaran antara *merchant* dan *payment gateway* pada protokol SET.

1.3 PEMBATASAN MASALAH

Secara garis besar, ada 4 entitas yang terlibat langsung dalam transaksi SET. Entitas tersebut adalah *cardholder*, *merchant*, *payment gateway* dan *certificate authority* (*CA*). Pada penelitian ini penulis hanya melibatkan 2 entitas yaitu *merchant* dan *payment gateway* serta komunikasi di antara keduanya. Entitas *cardholder* dan *CA* serta komunikasi dengan mereka berada di luar ruang lingkup penelitian ini.

Pada transaksi SET yang melibatkan *merchant* dan *payment Gateway* terdapat enam buah proses, yaitu: otorisasi, *capture*, pengembalian otorisasi, pengembalian *capture*, kredit, dan pengembalian kredit. Pada penelitian ini yang akan diimplementasikan adalah proses otorisasi.

Penelitian ini juga tidak menggali lebih dalam dan tidak mengimplementasikan objek-objek kriptografi yang dibutuhkan seperti RSA, DES, SHA-1, dan HMAC. Penulis hanya memakai implementasi kriptografi yang sudah ada seperti yang telah disediakan oleh *Sun*, *Cryptix* dan *Australian Bisnis Association* (ABA).

Prototipe proses otorisasi yang diimplementasikan tidak menyertakan modul verifikasi sertifikat karena modul ini merupakan penelitian tersendiri yang dilakukan oleh rekan penulis. Modul basis data dan modul antar muka juga tidak disertakan dalam penelitian ini.

1.4 METODOLOGI PENELITIAN

Penelitian dimulai dengan pemahaman pada spesifikasi bisnis SET yang terdapat pada Spesifikasi SET Buku Satu. Tahap ini lebih menekankan bagaimana penulis melihat SET secara keseluruhan, teknik kriptografi SET secara umum, dan proses transaksi SET secara garis besar.

Pada tahap kedua, penelitian berfokus pada survey. Penulis mencari paket-paket kriptografi yang dibutuhkan, standar-standar yang dipakai SET seperti *Public Key Cryptography System* (PKCS) 1-12, panduan *Abstract Syntax Notation One* (ASN. 1) dan *Distinguished Encoding Rules* (DER). Pada tahap ini penulis juga melakukan survey pada bahasa pemrograman yang akan digunakan.

Penulis mulai berfokus pada Spesifikasi SET Buku Dua tentang panduan pemrograman dan Spesifikasi SET Buku Tiga tentang protokol formal. Pada tahap ini pemahaman tentang SET secara mendalam dan implementasi mulai dilakukan.

Tahap akhir dari penelitian adalah tahap implementasi dan pengujian dari hasil implementasi SET.

1.5 SISTEMATIKA PENULISAN

Bab pertama diawali dengan penjelasan mengenai latar belakang masalah, dilanjutkan dengan tujuan penelitian, ruang lingkup permasalahan dan metode penelitian.

Bab kedua akan membahas landasan teori SET. Landasan teori tersebut berisi konsep SET dan proses transaksi pembayaran serta sedikit ulasan mengenai ASN.1 dan DER.

Bab ketiga mengulas analisa dan perancangan penulis terhadap spesifikasi SET ditinjau dari sudut pandang pesan dan proses.

Bab keempat membahas impelementasi SET, antara lain mengenai bahasa pemrograman, tahapan implementasi dan penjelasan implementasi objek-objek SET.

Bab kelima berisi metode-metode pengujian dan hasil pengujian terhadap implementasi SET ini.

Bab keenam merupakan bab penutup, berisi kesimpulan dan saran penulis terhadap penelitian ini.

BAB II
LANDASAN TEORI SET

Untuk mengimplementasikan protokol SET kita terlebih dulu melihat siapa saja pihak yang ikut bermain, kriptografi apa saja yang menjamin keamanannya, dan bagaimana proses SET menangani transaksi pembayaran. Selanjutnya untuk dapat mengerti spesifikasi teknis protokol SET kita juga perlu dilengkapi dengan pengetahuan mengenai ASN.1 dan DER.

Untuk alasan-alasan di atas pembahasan-pembahasan berikut ini dituliskan.

1.6 PARTISIPAN SISTEM PEMBAYARAN SET

Ada tujuh partisipan yang akan ikut bermain dalam sistem pembayaran SET [Set97a]. Ketujuh partisipan tersebut adalah:

1. Cardholder

Cardholder adalah orang yang menggunakan kartu pembayaran yang dikeluarkan oleh sebuah *issuer*. Dalam sistem pembayaran SET, informasi nomor rekening yang dikirimkan *cardholder* dijamin kerahasiaannya.

2. Issuer

Issuer adalah suatu institusi ekonomi (bank) yang membuat rekening dan mengeluarkan kartu pembayaran bagi *cardholder*. *Issuer* menjamin pembayaran untuk transaksi yang terotorisasi menggunakan kartu pembayaran yang dikeluarkannya, sesuai dengan regulasi yang dikeluarkan oleh pemegang merek dan pemerintah setempat. Contoh *issuer* adalah bank-bank yang mengeluarkan kartu kredit seperti Bank BCA, Bank Bali, Bank Lippo, dsb.

3. Merchant

Merchant adalah orang yang menyediakan barang atau jasa untuk dipertukarkan dengan pembayaran, dalam hal ini pembayaran dilakukan melalui internet. Dalam SET, *merchant* dapat menyediakan transaksi yang aman bagi *cardholder*. *Merchant* terlebih dahulu harus memiliki relasi dengan a*cquirer*.

4. Acquirer

Acquirer adalah suatu institusi ekonomi yang membuat rekening bagi *merchant* dan melakukan proses otorisasi kartu pembayaran yang diterima oleh *merchant* dari *cardholder.* Contoh *acquirer* adalah bank-bank yang menyediakan layanan otorisasi bagi toko-toko yang menerima pembayaran dengan kartu kredit seperti Bank BCA, Bank Bali, City Bank, dsb.

5. Payment Gateway

Payment gateway adalah suatu perangkat yang dioperasikan oleh *acquirer* atau pihak ketiga untuk memproses pesan pembayaran termasuk instruksi pembayaran dari *cardholder.*

6. Pemegang Merek

Pemegang merek adalah suatu badan usaha yang mengembangkan sistem kartu pembayaran, yang melindungi, mempromosikan dan menciptakan aturan-aturan penggunaannya.

Ada beberapa pemegang merek yang merupakan institusi ekonomi dan berperan sebagai *issuer* maupun *acquirer*.

Contoh pemegang merek yaitu Visa, Master Card, Amex, dsb.

7. Pihak Ketiga

Terkadang *acquirer* maupun *issuer* memilih pihak ketiga untuk menjalankan *payment gateway*.

1.7 KRIPTOGRAFI

Sebelum kita melangkah untuk membahas elemen-elemen kriptografi SET, pertama-tama kita lihat apa sebenarnya tujuan dari kriptografi ini pada SET.

SET adalah protokol yang dibangun di atas jaringan internet, jaringan yang sangat umum dan sangat rawan; oleh karena itu SET memerlukan suatu teknik keamanan transmisi pesan di atas jaringan internet. Karena alasan di atas maka SET membutuhkan elemen-elemen kriptografi yang akan dibahas pada sub bab berikut. Ada tiga kebutuhan keamanan transaksi yang perlu ditangani, yaitu:

1. Autentisitas atau keaslian, yaitu pesan yang diterima si penerima adalah benar-benar yang dikirimkan oleh si pengirim sebelumnya.

2. Kerahasiaan, yaitu pesan yang dikirimkan tidak bisa dibaca oleh orang yang tidak berhak, orang selain si penerima.
3. Integritas, yaitu pesan yang dikirimkan tidak mengalami suatu perubahan di tengah jalan.

Atas ketiga kebutuhan tersebut, elemen-elemen kriptografi pada sub bab berikut mengacu.

1.7.1 Hash dan *Message Digest*

Hash adalah suatu fungsi yang memetakan suatu dokumen asli ke suatu dokumen hasil pemetaannya dengan sifat:

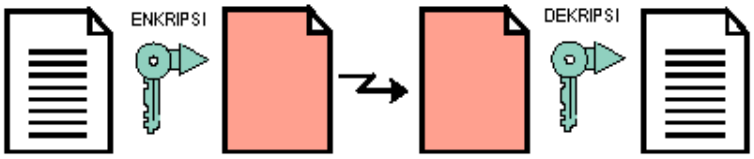
1. Dokumen hasil pemetaannya tidak dapat dipetakan kembali menjadi dokumen aslinya.
2. Tidak ada dua atau lebih dokumen asli dipetakan menjadi dokumen hasil pemetaan yang sama.

Pada protokol SET algoritma fungsi hash yang dipakai adalah Secure Hash Algorithm One (SHA.1). Dengan algoritma ini, besar dokumen hasil pemetaannya adalah 160 bit atau 20 byte [1].

Hasil pemetaan fungsi *hash* dari suatu data atau pesan pada protokol SET disebut *message digest* atau untuk lebih singkatnya bisa disebut *digest* saja.

1.7.2 Kriptografi Simetris atau Kriptografi Kunci Rahasia

Kriptografi simetris atau sering disebut juga kriptografi dengan kunci rahasia adalah suatu fungsi yang melakukan enkripsi dan dekripsi[2] dengan kunci yang sama. Antara pengirim dan penerima harus ada mekanisme pengiriman kunci yang aman agar kunci rahasia tidak jatuh ke tangan orang lain.



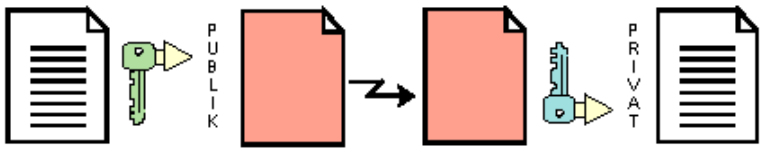
Gambar II-1 Kriptografi Simetris

[1] Penulis tidak mengulas lebih dalam algoritma-algoritma kriptografi yang dipakai dalam protokol SET karena penulis memakai modul-modul kriptografi yang sudah jadi. Sebagai referensi lebih lanjut mengenai hal ini, silakan lihat buku Applied Cryptography [Schneier 96].

Data Encryption Standard (DES) merupakan algoritma penyandian simetris yang dipakai pada protokol SET. Panjang kunci DES adalah 64 bit atau 8 byte.

1.7.3 Kriptografi Asimetris atau Kriptografi Kunci Publik

Berbeda dengan kriptografi simetris, kriptografi asimetris atau sering disebut kriptografi dengan kunci publik menggunakan 2 buah kunci yaitu kunci publik dan kunci privat. Kriptografi ini disebut kriptografi asimetris karena ia memiliki sifat jika suatu dokumen dienkripsi menggunakan kunci publik, maka dokumen hasil enkripsinya hanya bisa didekripsi menggunakan kunci privat; begitu pula sebaliknya.



Gambar II-2 Kriptografi Asimetris

Kunci privat biasanya disimpan dan dirahasiakan oleh pengirim, sedangkan kunci publik disebarluaskan kepada calon-calon penerima. Pada protokol SET, kunci publik disebarkan bersama sertifikat digital yang akan diterangkan pada sub-bab selanjutnya.

Algoritma penyandian asimetris yang dipakai dalam protokol SET adalah RSA, singkatan dari nama penemunya (Rivest, Shamir dan Adleman). Kunci RSA yang dipakai berkekuatan 1024 byte[3].

1.7.4 Tanda Tangan Digital

Apabila kita menulis sebuah surat, biasanya kita menandatangani surat tersebut. Kita melakukan itu dengan tujuan untuk menunjukkan bahwa surat itu otentik, surat itu memang buatan kita. Tanda tangan digital juga bertujuan sama dengan tanda tangan biasa, bedanya proses penandatanganannya juga bersifat digital.

Tanda tangan digital menggunakan gabungan dua teknik kriptografi yaitu *hash* dan kriptografi asimetris. Dokumen yang akan ditandatangani pertama-tama dibuatkan *digest*-nya, setelah itu *digest* tersebut dienkripsi dengan teknik kriptografi asimetris menggunakan kunci privat, hasilnya adalah tanda tangan digital. Dokumen asli dan tanda

---

[2] Enkripsi berarti menyandikan suatu pesan, sedangkan dekripsi adalah kebalikannya, yaitu mengartikan suatu pesan yang tersandikan.

[3] Kekuatan kunci RSA ditentukan oleh panjang byte-nya. Semakin panjang, semakin lama waktu diperlukan untuk membongkar pesan yang tersandikan.

tangan digital kemudian dikirim secara bersaman. Tujuan dari cara ini akan lebih jelas bila kita sudah melihat bagaimana cara melakukan verifikasinya.

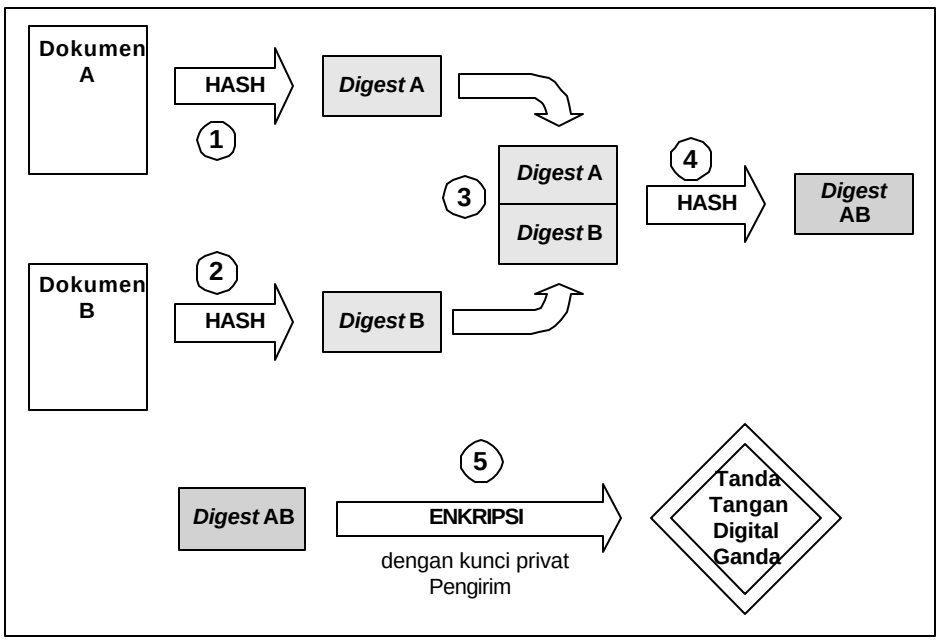
Dokumen dan tanda tangan digital yang diterima kemudian diverifikasi. Tanda tangan digital yang diterima mula-mula didekripsi menggunakan kunci publik yang diasumsikan sebelumnya sudah dimiliki si penerima. Hasil dari dekripsi tersebut adalah *digest*, kita sebut saja D1 (*digest* yang diperoleh dari tanda-tangan digital). Langkah selanjutnya adalah membuat *digest* dari dokumen yang kita terima, hasilnya kita sebut saja D2 (*digest* yang diperoleh dari dokumen). Langkah terakhir kita bandingkan D1 dan D2, keduanya harus sama.

Apa maksudnya jika D1 dan D2 sama? Jika mereka keduanya sama, berarti:

1. Dokumen yang diterima terbukti otentik. Jika tanda tangan digital yang diterima bisa didekripsi dengan kunci publik pengirim, pasti sebelumnya telah dienkripsi menggunakan kunci privat pengirim dan kunci privat tersebut hanya dimiliki oleh si pengirim.
2. Dokumen yang diterima terbukti isinya tidak diubah di tengah jalan pada waktu dikirimkan. Jika dokumen tersebut diubah di tengah jalan, D1 dan D2 tidak akan sama.

Tanda tangan digital berfungsi untuk menjamin autentisitas dan integritas pada protokol SET. Tanda tangan digital menggunakan gabungan algoritma SHA.1 dan RSA.

1.7.5 Tanda Tangan Digital Ganda



Gambar II-3 Tanda Tangan Digital Ganda

Tanda tangan digital ganda mirip dengan tanda tangan digital biasa, bedanya adalah tujuan tanda tangan digital ganda ini untuk menjamin keutuhan dari dua dokumen yang saling berhubungan (tidak satu pun dari kedua dokumen tersebut yang dapat diubah di tengah jalan), dan menjamin kedua dokumen tersebut tetap berhubungan; disamping menjamin autentisitas. Untuk lebih jelasnya dapat kita lihat pada gambar.

Pada gambar, dokumen A dan dokumen B kita asumsikan punya sesuatu yang berhubungan, misalnya masing-masing menyimpan nomor seri yang sama. Untuk menjamin keutuhan masing-masing dokumen dan menjamin dokumen A maupun dokumen B tetap berhubungan di tengah pengiriman, maka kita harus membuat tanda tangan digital ganda.

Cara membuat tanda tangan digital ganda: pertama, kedua dokumen itu masing-masing kita *hash* (langkah 1 dan 2); kemudian kedua hasil *hash* yang kita sebut *digest*, digabungkan (langkah 3); selanjutnya hasil gabungan *digest-digest* tersebut kita *hash* lagi (langkah 4); dan hasilnya kita enkripsi dengan kunci privat si pengirim (langkah 5). Kedua dokumen dan tanda tangan digital ganda tersebut lalu dikirimkan kepada si penerima.

Bila si penerima sudah menerima kedua dokumen dan tanda tangan digital ganda tersebut, ia akan melakukan verifikasi. Cara melakukan verifikasi adalah: pertama ia membuat *hash* ganda dari kedua dokumen (langkah 1 s.d. 4), katakan hasilnya kita sebut HG1; kemudian ia juga mendekripsi tanda tangan digital ganda tersebut dengan kunci publik si pengirim hasilnya merupakan *hash* ganda juga, kita sebut HG2; langkah terakhir adalah membandingkan HG1 dan HG2, keduanya harus sama. Jika HG1 dan HG2 sama membuktikan:

1. Kedua dokumen adalah otentik, karena tanda tangan digital ganda berhasil didekripsi menggunakan kunci publik si pengirim, ini menandakan bahwa sebelumnya *hash* ganda telah dienkripsi menggunakan kunci privat si pengirim dan yang memiliki kunci privat ini hanya si pengirim.
2. Keutuhan masing-masing dokumen terjamin.
3. Hubungan antara kedua dokumen tetap ada dan tidak berubah, karena data yang menghubungkan kedua dokumen yang berada di masing-masing dokumen tetap seperti sedia kala.

1.7.6 Amplop Digital

Amplop digital memiliki tujuan yang sama seperti amplop biasa, yaitu isi amplop hanya bisa dibaca oleh penerima yang sah.

Secara digital, proses pengamplopan ini adalah: data yang mau dikirimkan dienkripsi menggunakan kunci simetris yang dibuat secara acak; kemudian kunci simetris ini juga dienkripsi menggunakan kunci publik si penerima. Data yang mau dikirimkan tersebut bisa dikatakan teramplop karena hanya bisa dibaca oleh si penerima, yang bisa membuka amplopnya hanya kunci privat si penerima dan ini hanya dimiliki oleh si penerima.

Pertanyaan yang mungkin timbul di sini adalah mengapa data yang mau dikirimkan perlu dienkripsi dengan kunci simetris terlebih dahulu, tidak langsung dienkripsi dengan kunci publik si penerima? Jawabannya adalah untuk menghemat waktu komputasi, waktu komputasi untuk enkripsi dengan kunci publik/privat jauh lebih lama daripada enkripsi dengan kunci simetris, apalagi jika ukuran datanya besar.

Pada protokol SET, amplop digital berfungsi untuk menjamin kerahasiaan pesan.

1.7.7 Sertifikat Digital

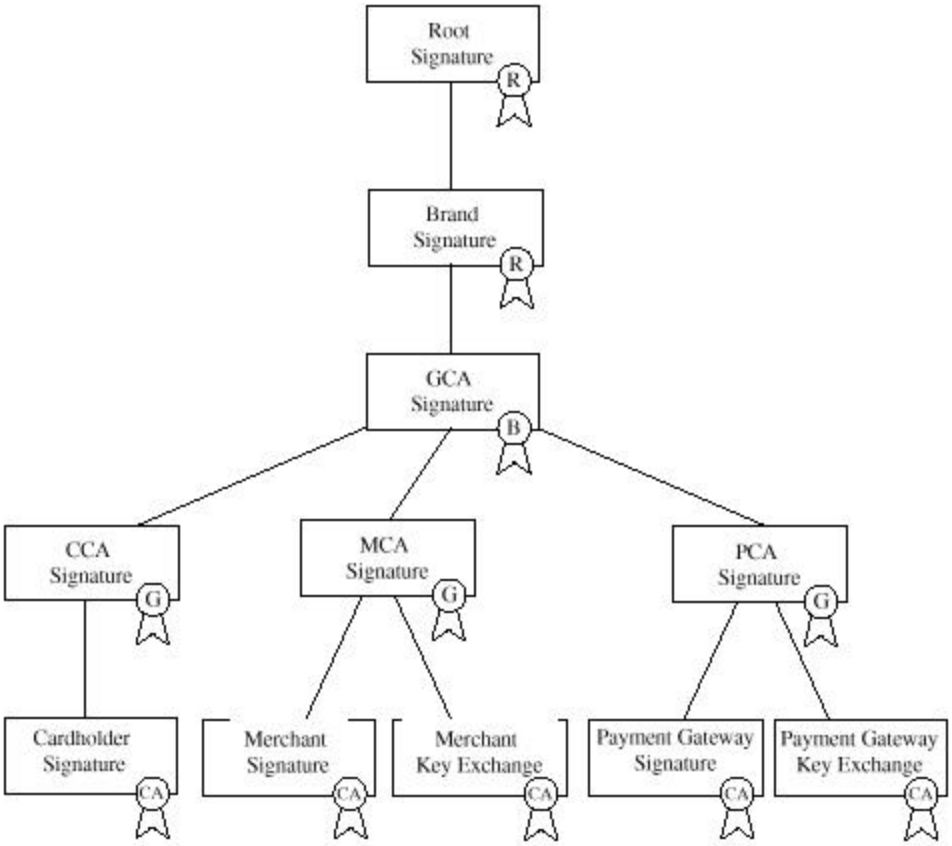
Dalam sebuah transaksi jual beli yang dilakukan melalui internet di mana kedua belah pihak tidak saling bertemu, harus ada suatu mekanisme tertentu yang menjamin identitas kedua pihak tersebut. Tidak ada pihak yang mau ditipu, bertransaksi dengan orang yang menyamar jadi orang lain atau dengan orang yang tidak memiliki sesuatu yang dapat ditransaksikan, namun hanya berpura-pura.

Sertifikat digital adalah informasi mengenai identitas pemilik sertifikat yang ditandatangani secara digital oleh sebuah badan independen yang menjamin bahwa si pemilik sertifikat layak untuk ikut dalam transaksi jual beli tersebut. Badan independen ini selanjutnya kita sebut *Certificate Authority* (CA). Termasuk dalam informasi yang terdapat dalam sertifikat digital adalah kunci publik, sehingga sertifikat digital ini juga merupakan mekanisme pertukaran kunci publik.

Sertifikat digital ini selain mempunyai hubungan dengan kunci publik dan identitas pemilik, ia juga memiliki hubungan yang sangat erat dengan nomor rekening bank pemilik sertifikat ini. Walaupun tidak secara langsung informasi rekening bank ini tercantum dalam sertifikat , CA menyimpan nomor rekening tersebut dalam basis data miliknya, sehinggga nomor rekening tersebut dapat diasosiasikan dengan sertifikatnya.

Berbicara tentang CA, terdapat hirarki kedudukan di antara mereka. Sebuah CA dapat memiliki sertifikat yang ditandatangani oleh CA di tingkat atasnya, demikian pula CA di tingkat atasnya tersebut dapat memiliki sertifikat yang ditandatangani oleh CA di tingkat lebih atasnya lagi, begitu seterusnya sampai *Root CA*. Sertifikat milik *Root CA*

ditandatangani oleh dirinya sendiri. Karena karena tingkatan sertifikat itu identik dengan tingkatan kunci publik, maka *Root CA* sering disebut *Root Key*.



Gambar II-4 Hirarki Kepercayaan *Certificate Authority* (CA)

Pada gambar dapat kita lihat terdapat bermacam-macam CA. Terdapat *Cardholder* CA (CCA) yang mengeluarkan sertifikat *cardholder*, *Merchant* CA (MCA) yang mengeluarkan sertifikat *merchant*, *Payment Gateway* CA (PCA) yang mengeluarkan sertifikat *payment gateway*, *Geopolitical* CA (GCA) yang menaungi CA-CA dalam satu negara, *Brand* yang menaungi CA-CA dalam satu merek tertentu, dan yang paling atas adalah *Root*. Masing-masing sertifikat CA tersebut memiliki hubungan ke sertifikat CA di tingkat atasnya, terutama melalui tanda tangan CA di tingkat atasnya.

Saat ini kita mungkin memiliki pertanyaan bahwa *carholder*, *merchant*, dan *payment gateway* juga perlu jaminan ketika sedang berhubungan dengan sebuah CA. *Certificate Authority* (CA) tentu memiliki sertifikat, namun bagaimana cara memverifikasi sertifikat CA tersebut. Ada beberapa fakta singkat yang menggambarkan hal itu, yakni:

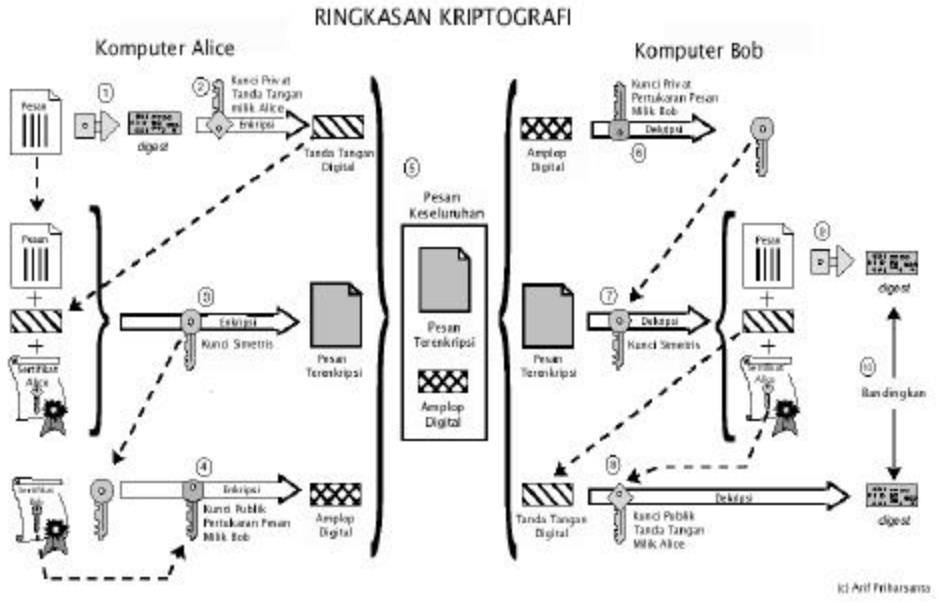
Seluruh vendor yang menyediakan perangkat lunak bagi cardholder, merchant, maupun payment gateway telah mempunyai hubungan dengan suatu Root CA. Mereka menyisipkan sertifikat *root* dalam setiap perangkat lunak yang mereka pasarkan.

Cardholder, *merchant* dan *payment gateway* ketika melakukan registrasi sertifikat pada suatu CA; selain diberikan kembali sertifikat milik mereka sendiri, juga diberikan kumpulan sertifikat para CA.

Cardholder, *merchant* dan *payment gateway* menelusuri jalur tanda-tangan sertifikat-sertifikat CA tersebut mulai dari sertifikat CA yang akan diverifikasi sampai ke sertifikat *root*. Kemudian sertifikat *root* yang ada dalam kumpulan sertifikat tersebut dibandingkan dengan sertifikat *root* yang tersimpan dalam perangkat lunak keluaran *vendor* dan harus sama.

1.7.8 Ringkasan Kriptografi SET

Setelah beberapa sub-bab di atas membahas elemen-elemen kriptografi SET, maka di bawah ini kita akan melihat ringkasan kriptografi SET menggunakan gabungan elemen-elemen di atas.



Gambar II-5 Ringkasan Kriptografi SET

Pada ringkasan kriptografi SET ini Alice hendak mengirimkan sebuah pesan pada Bob. Sebelum proses pengiriman dilakukan, diasumsikan bahwa Alice sudah memiliki

sertifikat pertukaran pesan milik Bob yang didalamnya berisi kunci publik pertukaran pesan milik Bob.

Berikut ini adalah penjelasan langkah-langkah di dalam gambar:

1. Alice membuat *digest* dari pesannya.
2. *Digest* tersebut dienkripsi dengan kunci privat tanda tangan miliknya sehingga menjadi tanda tangan digital. Pesan asli, tanda tangan digital, serta sertifikat tanda tangan miliknya digabungkan menjadi satu.
3. Gabungan ketiganya pada langkah kedua dienkripsi dengan sebuah kunci simetris, hasilnya kita sebut saja pesan terenkripsi.
4. Kunci simetris pada langkah ketiga dienkripsi menggunakan kunci publik pertukaran pesan milik Bob, hasil enkripsi ini kita sebut amplop digital.
5. Pesan terenkripsi bersama dengan amplop digital dikirimkan ke Bob dan katakanlah sudah diterima Bob.
6. Bob mendekripsi amplop digital untuk mendapatkan kunci simetris.
7. Pesan terenkripsi didekripsi dengan kunci simetris yang diperoleh pada langkah keenam, hasilnya adalah: pesan asli, tanda tangan digital, dan sertifikat tanda tangan milik Alice.
8. Bob mendapatkan kunci publik tanda tangan milik Alice dari sertifikat tanda tangan milik Alice. Kunci publik ini digunakan untuk mendekripsi tanda tangan digital yang diperoleh pada langkah ketujuh, hasilnya adalah sebuah *digest*, katakanlah *digest1*.
9. Bob membuat *digest* dari pesan asli yang diperoleh pada langkah ketujuh, hasilnya kita sebut *digest2*.
10. Bob membandingkan *digest1* dan *digest2* untuk membuktikan integritas dan autentisitas dari pesan yang dikirimkan Alice.

1.8 PROSES TRANSAKSI PEMBAYARAN

Pada sub bab ini kita akan melihat bagaimana alur transaksi SET yang diproses oleh berbagai macam sistem.

Ada beberapa simbol yang dipakai dalam mendefinisikan alur proses transaksi SET [Set97a]. Simbol-simbol tersebut akan dijelaskan dalam tabel berikut:

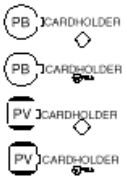
Inisial Partisipan

Inisial	Partisipan
C	Cardholder
M	Merchant
PG	Payment Gateway
CA	Certificate Authority

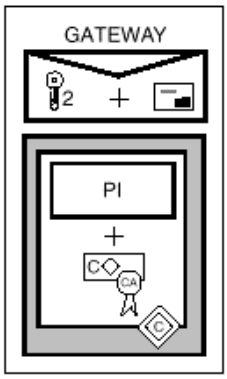
Tabel II-1 Tabel Inisial Partisipan

Daftar Simbol

Simbol-simbol berikut ini akan sering dipakai dalam gambar-gambar yang akan melukiskan proses-proses transaksi SET (Gambar II-5 dan II-6).

Simbol	Deskripsi
	Ini adalah kunci-kunci pada kriptografi kunci publik: - Tulisan pada batang kunci menandakan pemiliknya. - Inisial "PB" menandakan bahwa kunci tersebut adalah kunci publik, sedangkan inisial "PV" menandakan bahwa kunci tersebut adalah kunci privat. - Lambang wajik (◇) menandakan kunci tersebut dipergunakan untuk membuat tanda tangan, sedangkan lambang kunci (⊙═) menandakan kunci tersebut dipergunakan untuk pertukaran pesan.
	Ini adalah tanda tangan digital, inisial pada tanda tangan tersebut menandakan kunci privat milik siapa yang menandatangani. Pada simbol di samping, merupakan tanda *tangan merchant*.
	Ini adalah tanda tangan digital ganda, inisial yang tertera di sini menandakan kunci privat milik siapa yang menandatangani. Pada simbol di samping, merupakan tanda tangan *cardholder*
	Ini adalah sertifikat digital: - Inisial pada segel menandakan kunci privat milik siapa yang menandatangani sertifikat tersebut, contoh di samping adalah kunci privat milik *CA* yang menandatangani sertifikat tersebut. - Inisial pada sertifikat menandakan kunci publik milik siapa yang disertifikasi, contoh disamping adalah kunci publik milik *merchant*. - lambang wajik dan kunci menjadi pembeda antara sertifikat tanda tangan dengan sertifikat pertukaran pesan.

Gambar II-6 Simbol Diagram Transaksi SET 1

	Ini adalah kunci simetris yang digunakan untuk mengenkripsi data. Kunci ini selalu dikirimkan bersamaan dengan data terenkripsi dalam amplop digital. Nomor yang tertera di samping kunci berguna untuk membedakan dengan kunci simetris lainnya.
	Ini adalah simbol kartu pembayaran, simbol ini digunakan ketika nomor rekening *cardholder* dikirimkan dalam amplop digital bersamaan dengan kunci simetris.
	Ini adalah simbol pesan terenkripsi di dalam amplop digital. Pesan sesungguhnya adalah yang dikelilingi wilayah berwarna abu-abu. Pesan tersebut dienkripsi dengan kunci simetris, kemudian kunci simetris tersebut dienkripsi lagi menggunakan kunci publik penerima. Pada contoh disamping digambarkan PI, sertifikat tanda tangan *cardholder,* dan tanda tangan ganda *cardholder* dienkripsi dengan kunci simetris (#2). Kemudian kunci simetris (#2) dan nomor rekening *cardholder* dienkripsi menggunakan kunci publik pertukaran pesan milik *gateway* (ditandai dengan tulisan "Gateway" di atas gambar amplop). Disebut amplop digital, karena hanya penerima yang dapat membukanya, dalam hal ini *gateway*.

Gambar II-7 Simbol Diagram Transaksi SET 2

2.1.1 Fase Registrasi

Fase registrasi adalah fase di mana setiap entitas yang terlibat langsung dalam transaksi SET (*cardholder, merchant,* dan *payment gateway*) mendaftarkan diri ke CA untuk mendapatkan sertifikat sebagai bukti bahwa dirinya adalah pemain yang sah dalam transaksi SET.

Bagi *cardholder* sertifikat merupakan representasi elektronik kartu pembayaran. Kalau dalam transaksi biasa, untuk membayar ia harus menunjukkan kartunya, dalam transaksi SET karena melalui media internet ia tidak dapat menunjukkan kartunya. Sebagai gantinya ia harus mampu menunjukkan sertifikatnya. Sedangkan bagi *merchant* sertifikat merupakan pengganti logo bank yang berelasi dengannya dan biasa ditempel di tokonya, sebagai bukti bahwa ia punya relasi dengan sebuah *acquirer*. Sertifikat bagi *payment gateway* berfungsi sebagai bukti bahwa ia dapat memproses otorisasi, biasanya yang mengurus sertifikat *payment gateway* adalah *acquirer* yang menjalankannya.

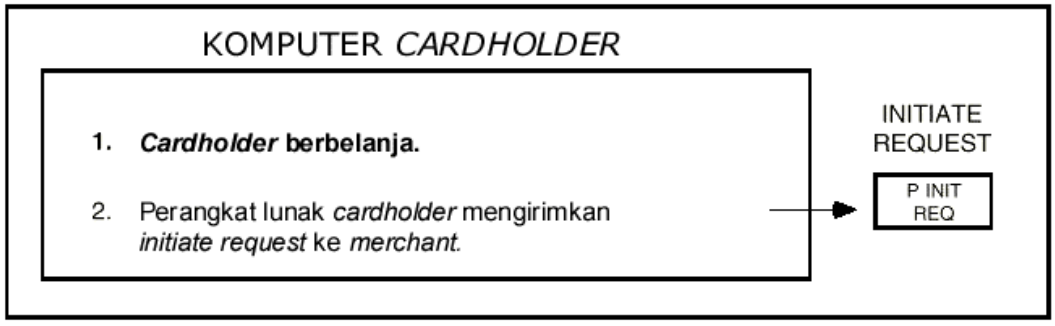
Fase ini tidak diulas lebih dalam lagi karena berada di luar ruang lingkup penelitian. Untuk fase berikutnya diasumsikan bahwa setiap entitas yang terlibat langsung

dalam transaksi SET telah memiliki sertifikat. C*arholder* mempunyai satu macam sertifikat yaitu sertifikat tanda tangan, sedangkan *merchant* dan *payment gateway* masing-masing memiliki dua macam sertifikat yaitu sertifikat tanda tangan dan sertifikat pertukaran pesan.

1.8.1 Fase Belanja

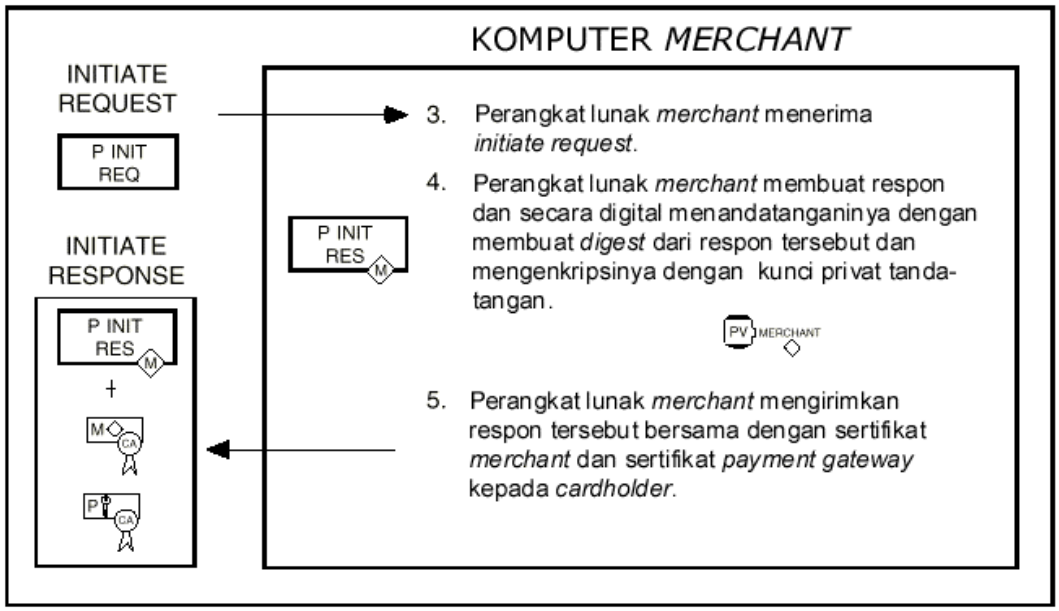
Pada fase ini, interaksi dilakukan antara *cardholder* dan *merchant*. Fase ini sebenarnya masih diluar ruang lingkup penelitian ini, namun penulis merasa penting menuliskan hal ini agar pembaca mengerti jalan cerita sebelum masuk ruang lingkup yang sebenarnya.

Sebelum fase ini, yaitu sebelum protokol SET mulai beraksi; *cardholder* sudah dihadapkan pada sejumlah barang atau jasa yang ditawarkan di *browser*, ia telah menyetujui cara pembayarannya (lunas atau cicilan) dan telah mengisi sejumlah formulir pemesanan yang dibutuhkan. Data pemesanan sudah dimiliki baik oleh *cardholder* maupun *merchant.*



Gambar II-8 *Cardholder* membuat *Initiate Request*

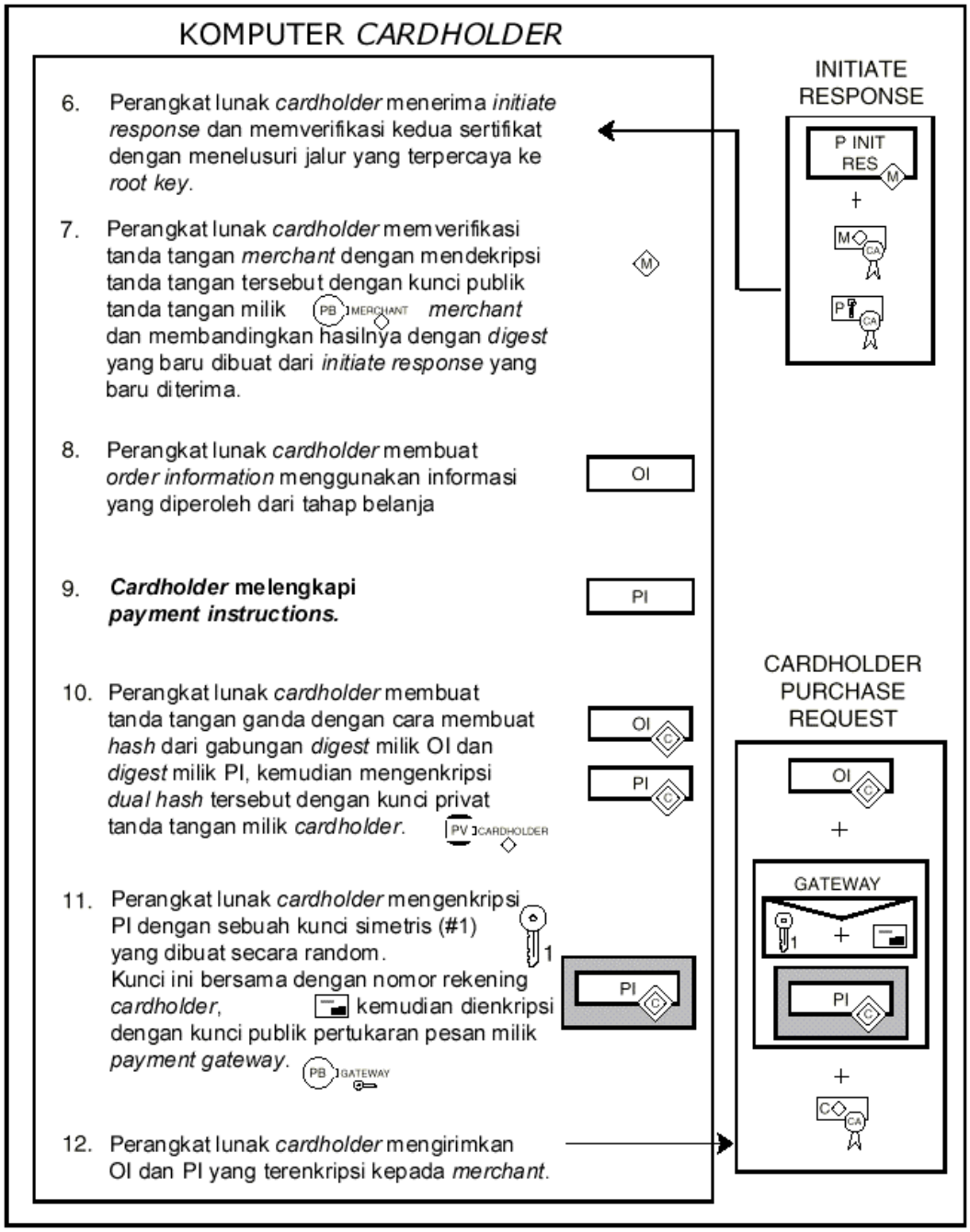
Ketika *merchant* menerima *initiate request* tersebut, ia memberikan sebuah nomor identifikasi yang unik pada transaksi ini. *Merchant* kemudian mengirimkan sertifikat *merchant* dan sertifikat *gateway* yang berhubungan dengan merek kartu pembayaran yang diminta oleh *cardholder*, bersamaan dengan nomor identifikasi transaksi tersebut.



Gambar II-9 *Merchant* membuat *Initiate Response*

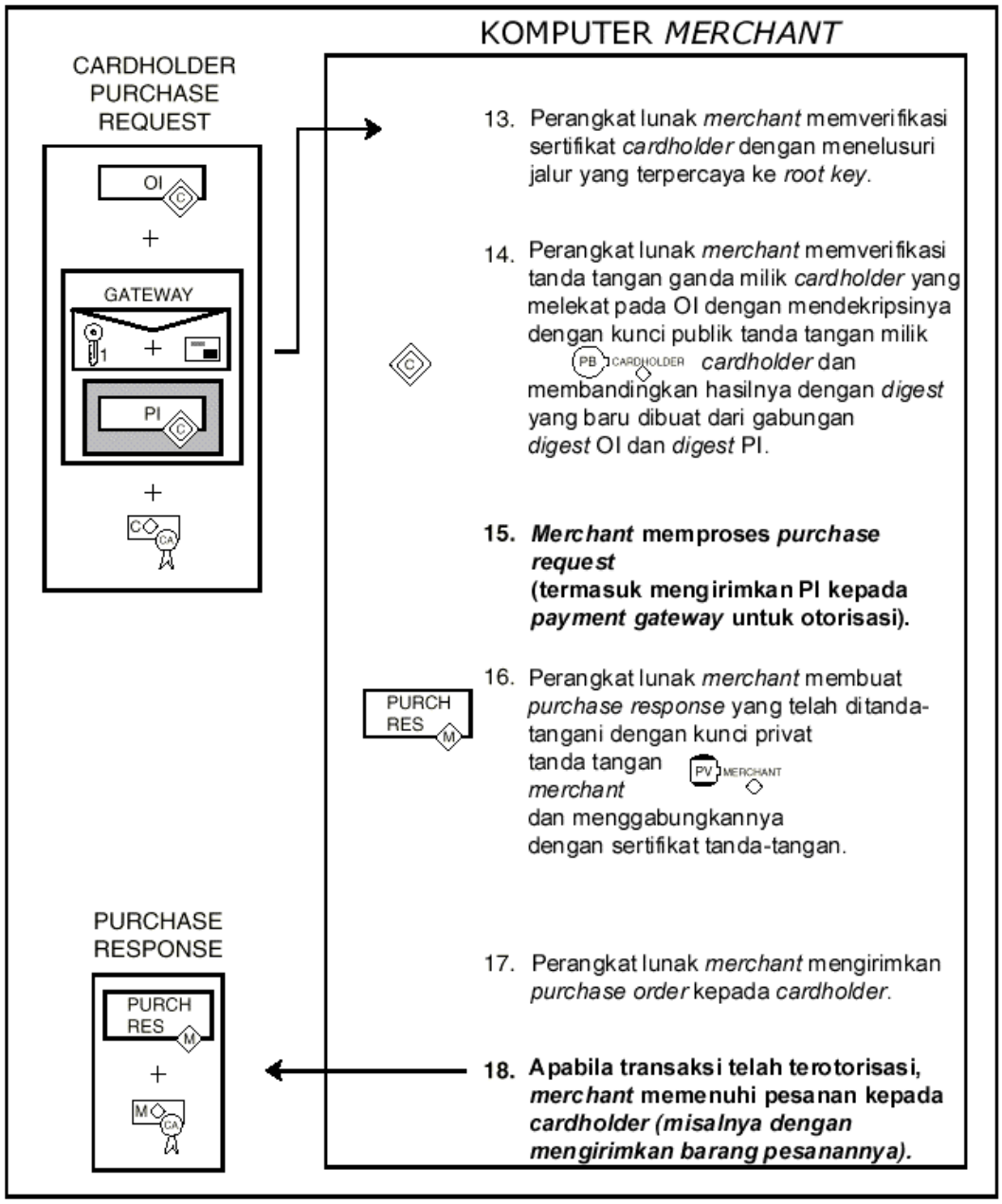
Perangkat lunak *cardholder* kemudian melakukan verifikasi terhadap sertifikat *merchant* dan sertifikat *gateway* dengan menelusuri jalur tertentu yang dapat dipercaya ke *root key.* Setelah itu ia juga memverifikasi tanda tangan *merchant* pada *initiate response.* Langkah selanjutnya, perangkat lunak *cardholder* membuat *Order Information* (OI) dan *Payment Instructions* (PI). Perangkat lunak *cardholder* juga telah menyelipkan nomor identifikasi transaksi sebelumnya pada OI dan PI yang kelak akan digunakan *payment gateway* untuk menghubungkan OI dan PI. Kemudian keduanya dibuatkan tanda tangan digital ganda dengan kunci privat tanda tangan milik *cardholder*. PI kemudian diamplopkan secara digital dengan kunci publik pertukaran pesan milik *gateway* yang didapat dari sertifikat *gateway* pada langkah sebelumnya. OI dan PI yang teramplop kemudian dikirimkan ke *merchant* bersama sertifikat tanda tangan *cardholder.*

Catatan: OI bukan berisi data-data pemesanan, cicilan, dsb; sekali lagi data-data ini sudah dipertukarkan pada tahap sebelum transaksi SET dimulai.



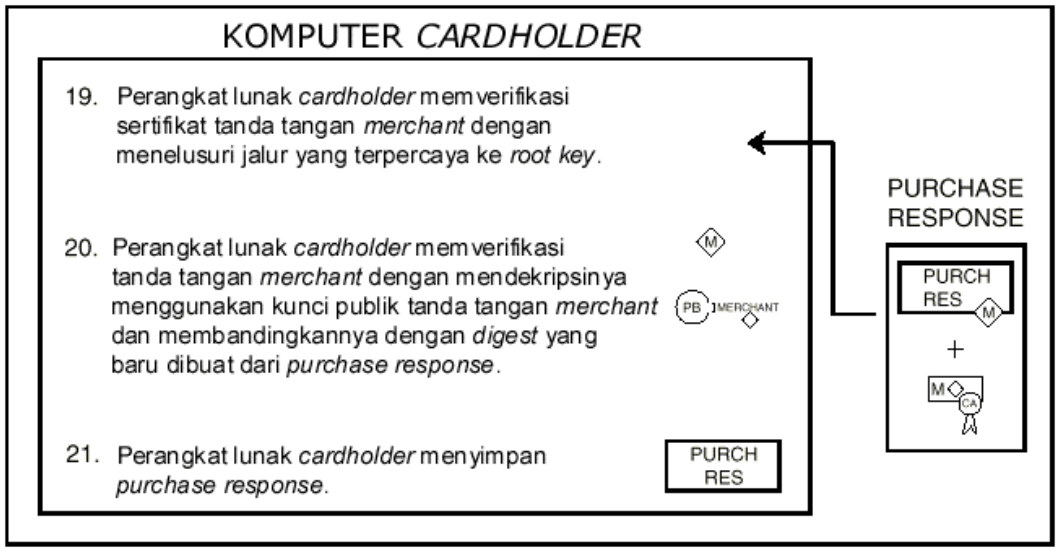
Gambar II-10 *Cardholder* membuat *Purchase Request*

Setelah *merchant* menerima *purchase request*, perangkat lunak *merchant* memverifikasi sertifikat *cardholder* terlebih dahulu, kemudian melakukan proses otorisasi ke *payment gateway* dan mengirimkan hasil otorisasinya dalam *purchase response* ke *cardholder.*



Gambar II-11 *Merchant* membuat *Purchase Response*

Perangkat lunak *cardholder* memverifikasi sertifikat tanda tangan *merchant,* kemudian juga memverifikasi tanda tangan *merchant.* Setelah itu perangkat lunak *cardholder* menampilkan di layar tentang hasil otorisasi, dan selanjutnya mungkin menyimpan dalam basis data di komputer *cardholder* data-data pembelian tersebut. *Purchase Response* disimpan oleh komputer *cardholder* sebagai pengganti bon pada transaksi pembelian biasa.

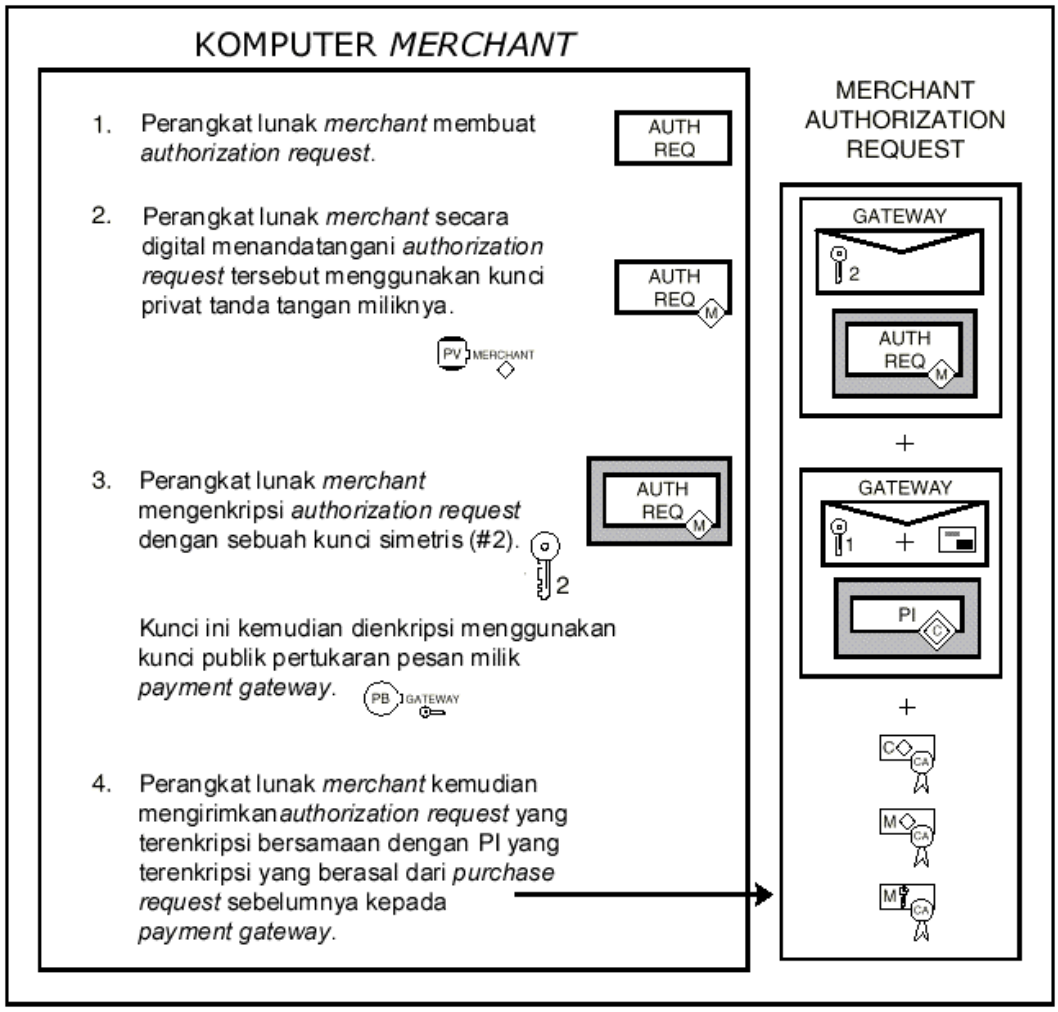


Gambar II-12 *Cardholder* menerima *Purchase Response*

1.8.2 Fase Otorisasi

Fase otorisasi melibatkan hubungan antara *merchant* dan *payment gateway*. Fase ini terjadi segera setelah *cardholder* mengirimkan *purchase request* kepada *merchant* yang di dalamnya terdapat OI dan PI teramplop

Perangkat lunak *merchant* membuat dan secara digital menandatangani *authorization request* yang di dalamnya tercakup nomor identifikasi transaksi, harga pembelian, dan data-data transaksi lainnya yang diperlukan. Setelah ditandatangani *authorization request* tersebut dienkripsi dengan sebuah kunci simetris yang dibuat secara acak, kemudian kunci ini dienkripsi dengan kunci publik pertukaran pesan milik *payment gateway* (diamplopkan). *Authorization request* teramplop dan PI teramplop dikirimkan ke *payment gateway* bersamaan dengan sertifikat tanda tangan *cardholder*, sertifikat tanda tangan *merchant* serta sertifikat pertukaran pesan *merchant*.

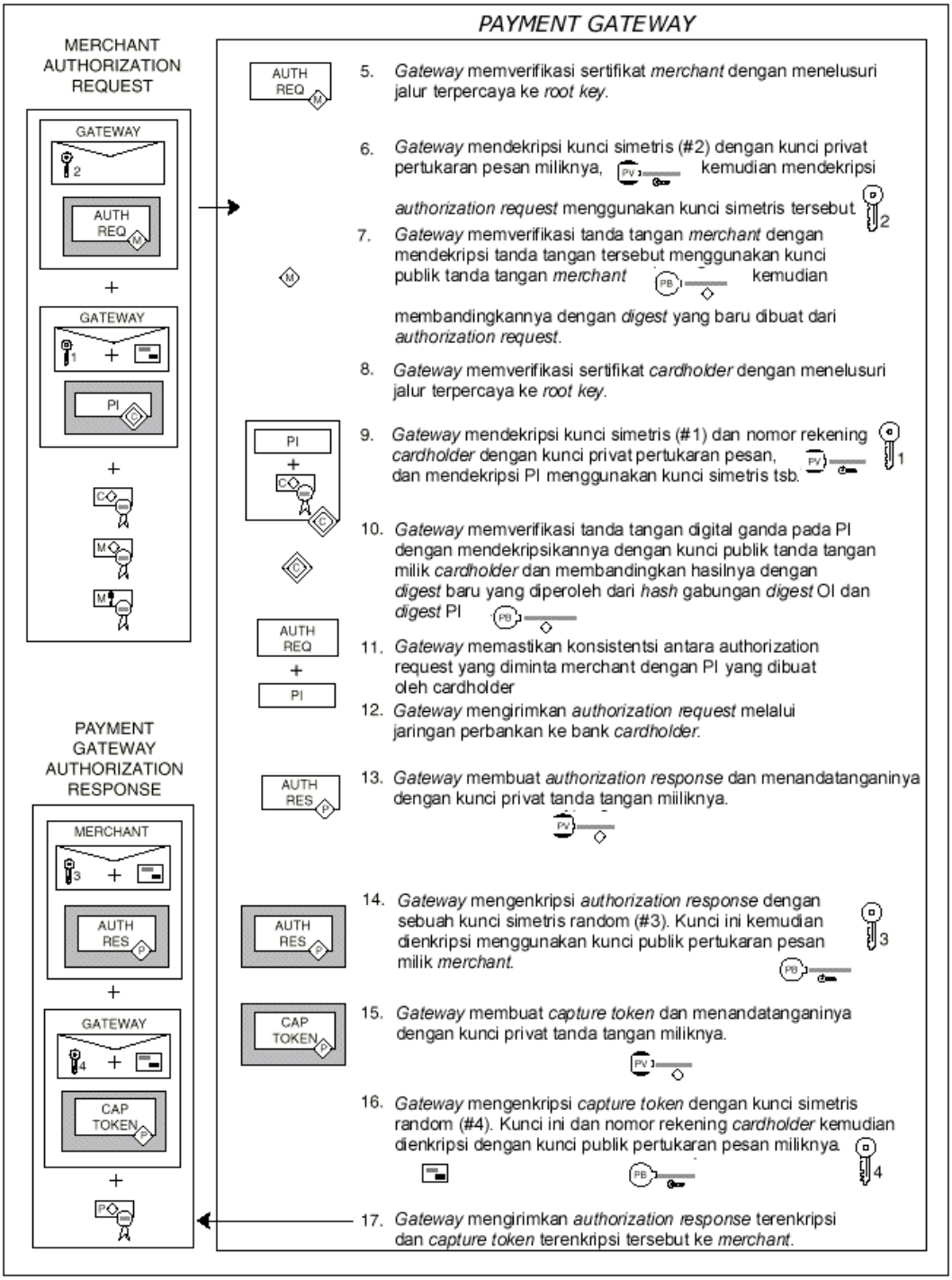


Gambar II-13 *Merchant* membuat *Authorization Request*

Ketika *payment gateway* menerima *authorization request*, ia memverifikasi seluruh sertifikat yang diterima dengan menelusuri jalur terpercaya ke *root key*. Kemudian ia membuka kedua amplop, yaitu amplop *authorization request* dan amplop PI, membandingkan konsistensi di antara keduanya (nomor identifikasi transaksi serta jumlah harga pembelian, dsb). Setelah itu *payment gateway* melakukan proses otorisasi lebih lanjut dengan *issuer*.

Selanjutnya setelah proses otorisasi dengan *issuer* selesai, *payment gateway* akan membuat *authorization response* dan *capture token*. *Authorization response* diamplopkan dengan alamat tujuan *merchant;* sedangkan *capture token* diamplopkan dengan alamat tujuan *payment gateway* sendiri. *Capture token* berfungsi sebagai tanda bukti jika *merchant* ingin melakukan proses *capture*, oleh karena itu *capture token* diamplopkan dengan alamat tujuan *payment gateway* sendiri. *Authorization response* teramplop dan

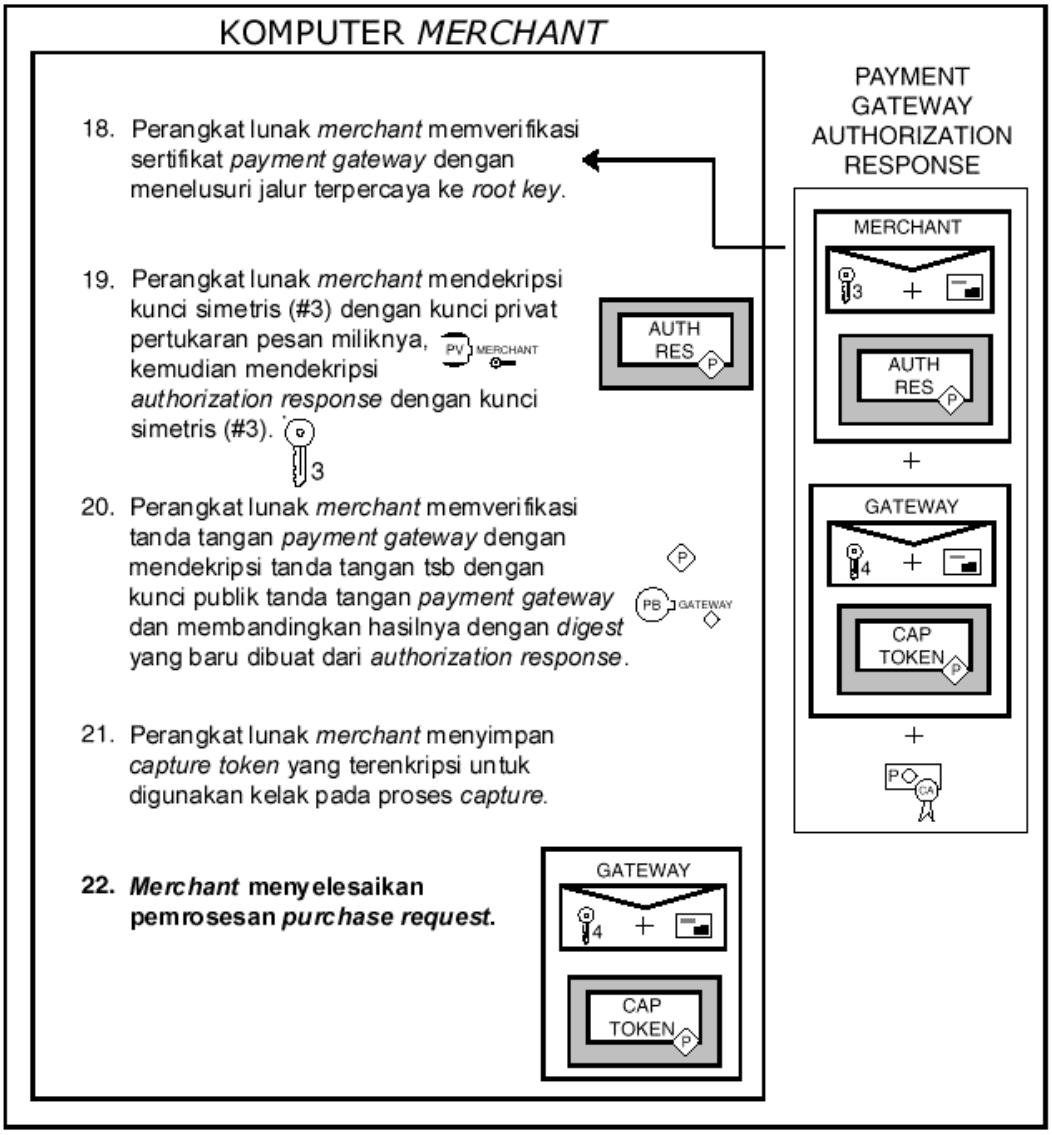
Capture token teramplop bersama dengan sertifikat tanda tangan *payment gateway* dikirimkan ke *merchant.*



Gambar II-14 *Payment Gateway* membuat *Authorization Response*

Setelah *merchant* menerima *authorization response* teramplop dan *capture token* teramplop; ia memverifikasi sertifikat *payment gateway*, kemudian ia membuka dan

memverifikasi *authorization response*, sedangkan *capture token* disimpan untuk proses *capture* berikutnya.



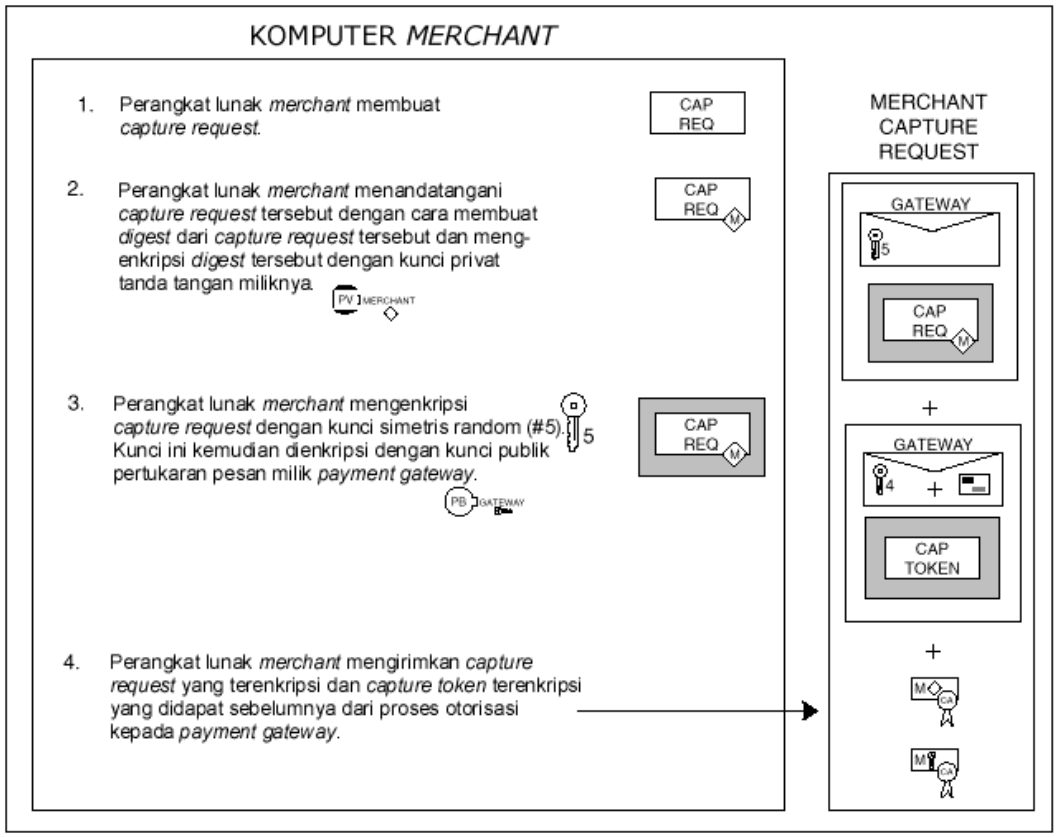
Gambar II-15 Merchant menerima *Authorization Response*

1.8.3 Fase *Capture*

Pada fase ini *merchant* hendak meminta pembayaran dari *issuer* atas transaksi perdagangan yang telah diotorisasi sebelumnya. *Merchant* tidak secara langsung meminta ke *issuer*, tapi melalui *acquirer*. Pihak yang terlibat langsung dalam fase *capture* ini adalah *merchant* dan *payment gateway*.

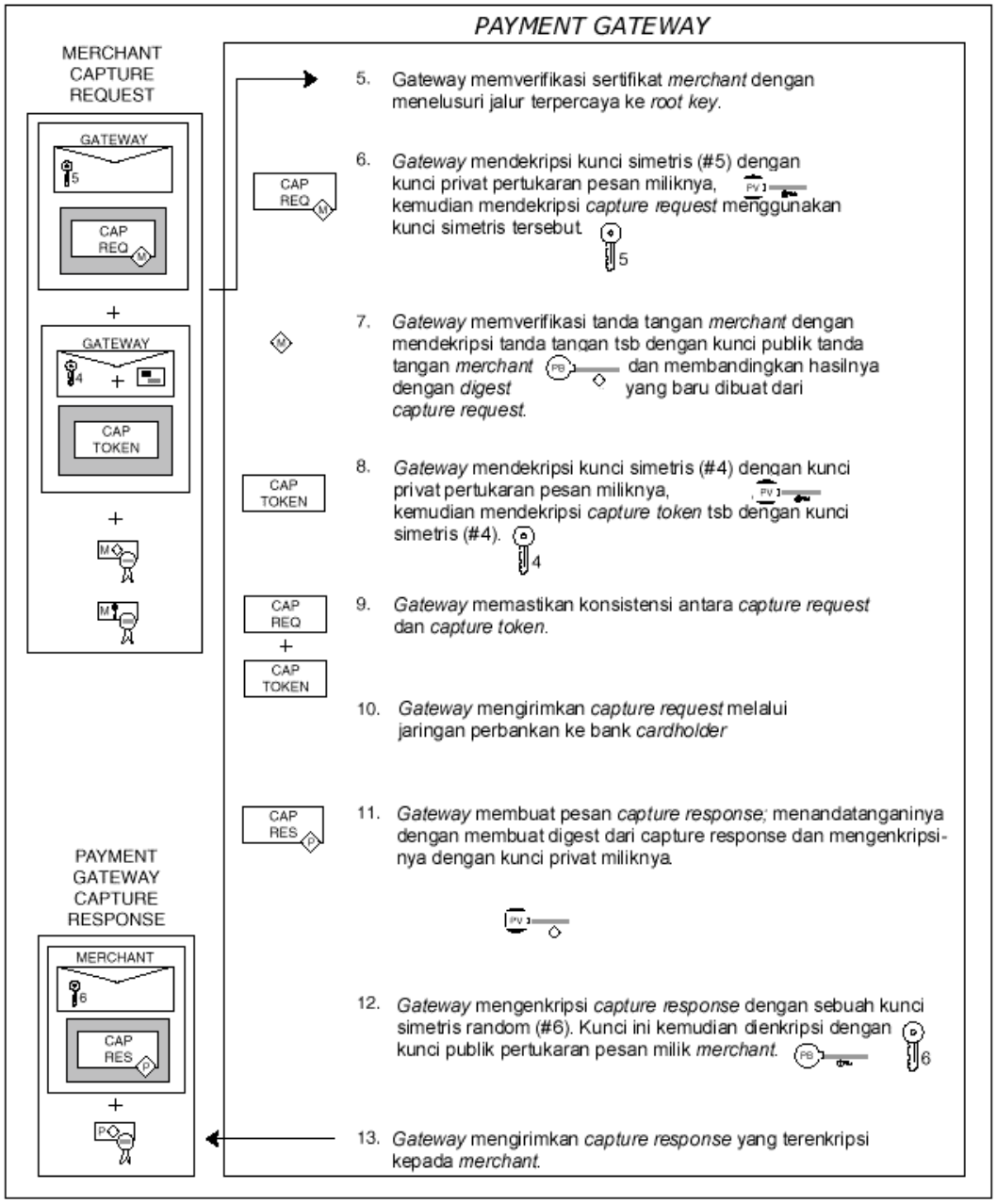
Pertama, perangkat lunak *merchant* membuat dan secara digital menandatangani *capture request* yang didalamnya terdapat nilai total transaksi, nomor identifikasi transaksi yang berasal dari OI, dan data transaksi lainnya yang diperlukan. *Capture*

request tersebut kemudian secara digital diamplopkan dengan alamat tujuan *payment gateway*. *Capture Request* yang teramplop, *capture token* yang teramplop, serta sertifikat-sertifikat *merchant* dikirimkan kepada *payment gateway*.



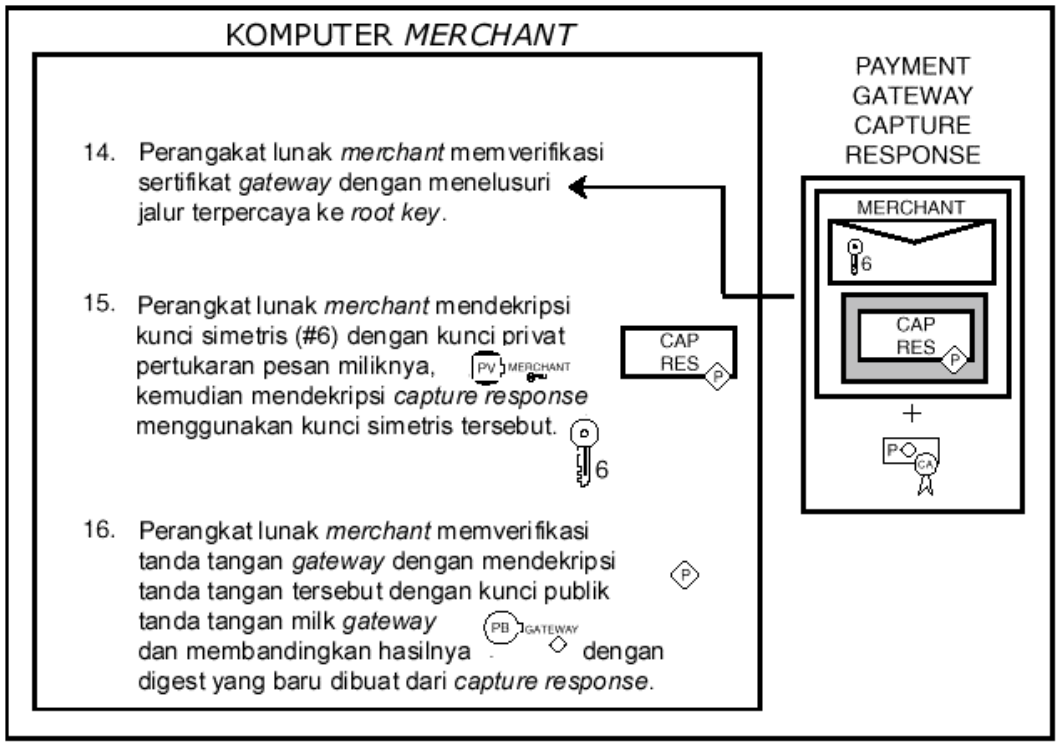
Gambar II-16 *Merchant* membuat *Capture Request*

Payment gateway menerima kiriman tersebut, ia memverifikasi sertifikat-sertifikat *merchant,* kemudian membuka kedua amplop *capture request* dan *capture token.* Selanjutnya ia menguji konsistensi keduanya dengan data transaksi yang sebelumnya didapatkan pada proses otorisasi yang tersimpan pada basis datanya. Setelah pengujian sukses, ia membuat permohonan *capture* ke *issuer* melalui jaringan antar bank yang ada agar *issuer* mentransfer dananya ke rekening *merchant* di *acquirer*. Hasil proses *capture* ini yaitu *capture response*, dikirimkan teramplop kepada *merchant* bersama sertifikat tanda tangan *payment gateway*.



Gambar II-17 *Payment Gateway* membuat *Capture Response*

Ketika *merchant* menerima *capture response* teramplop tersebut, ia memverifikasi sertifikat tanda tangan *payment gateway*, kemudian membuka amplop *capture response. Capture response* itu lalu disimpannya sebagai tanda bukti atau kwitansi jika nanti terjadi kekeliruan pada bank, bahwa ia telah berhasil melakukan proses *capture* tersebut.



Gambar II-18 *Merchant* menerima *Capture Response*

Fase *capture*, saat ini belum dicakup dalam penelitian ini. Batasan ruang lingkup penelitian ini akan lebih jelas pada bab analisa dan perancangan.

1.9 ASN.1 DAN DER

Adalah sesuatu yang sudah diketahui umum bahwa abstraksi merupakan kunci pengembangan dalam prinsip perancangan perangkat lunak.

Dengan abstraksi, seorang perancang dapat menentukan bagian dari suatu sistem tanpa memperhatikan bagaimana cara mengimplementasikan bagian tersebut. Cara pengimplementasiannya bebas, yang penting representasi hasilnya sesuai dengan yang dimaksud oleh abstraksi tersebut [Kali93].

Contoh abstraksi dalam kehidupan nyata yaitu bahasa, kalau kita bilang "makan", maka representasi dari kata tersebut sebenarnya adalah "ada sesuatu yang masuk melalui mulut kita dan perlu dikunyah terlebih dahulu, tidak langsung ditelan". Kita tidak peduli bagaimana proses implementasinya; entah mengambil piring lalu diisi nasi dan lauk, kemudian disuapkan ke mulut kita; atau beli biskuit di warung, kemudian kita ambil dan kita masukkan dalam mulut kita.

Open Systems Interconnection (OSI) telah mengeluarkan metode untuk menspesifikasikan suatu objek secara abstrak dengan *Abstract Syntax Notation One*

(ASN.1) beserta sekumpulan aturan untuk merepresentasikan objek abstrak tersebut menjadi sekumpulan bit nol dan satu yang dinamakan *Distinguished Encoding Rules* (DER).

SET memakai kedua metode ini untuk menspesifikasikan protokol pembayarannya.

1.9.1 Abstract Syntax Notation One (ASN.1)

ASN.1 adalah suatu notasi untuk menggambarkan tipe dan nilai secara abstrak.

Pada ASN.1, yang dimaksud dengan tipe adalah himpunan dari nilai-nilai. Banyaknya nilai-nilai tersebut bisa terbatas dan bisa tidak terbatas. ASN.1 memiliki empat macam tipe: tipe sederhana, yaitu tipe yang atomik dan tidak memiliki komponen; tipe terstruktur, yaitu tipe yang memiliki satu atau lebih komponen; yaitu tipe turunan, tipe yang diturunkan dari tipe lain; dan tipe lainnya adalah termasuk tipe CHOICE atau tipe ANY. CHOICE dan ANY akan diterangkan pada pembahasan selanjutnya. Tipe dan nilai dalam ASN.1 dapat dinamakan melalui operator *assignment* (::=), dan nama tersebut dapat digunakan dalam mendifinisikan tipe atau nilai lain.

Setiap tipe dalam ASN.1 selain CHOICE dan ANY memiliki sebuah *tag*, yang terdiri dari sebuah kelas dan bilangan *integer* non-negatif. Dua tipe dalam ASN.1 secara abstrak dikatakan sama jika dan hanya jika keduanya memiliki *tag* yang sama, walaupun namanya berbeda. Terdapat empat macam kelas dari *tag*, yaitu:

Universal

Kelas *Universal* adalah kelas untuk tipe-tipe yang memiliki arti sama dalam setiap aplikasi. Jenis tipe ini hanya didefinisikan dalam X.208.

Application

Kelas *Application* adalah kelas untuk tipe-tipe yang memiliki arti spesifik pada aplikasinya. Dua aplikasi yang berbeda mungkin memiliki *tag* aplikasi-spesifik yang sama untuk arti yang berbeda.

Private

Kelas *Private* adalah kelas untuk tipe-tipe yang memiliki hubungan spesifik ke suatu perusahaaan.

Context Specific

Kelas *Context-Specific* adalah kelas untuk tipe-tipe yang memiliki arti spesifik pada suatu tipe terstruktur. *Tag context-specific* digunakan untuk membedakan komponen-komponen yang memiliki tipe sama dalam suatu konteks tipe terstruktur. Dua komponen dalam dua tipe terstruktur yang berbeda dapat memiliki *tag* yang sama dalam arti yang berbeda.

Tipe dan nilai dalam ASN.1 diekspresikan secara fleksibel seperti bahasa pemrograman, dengan aturan-aturan khusus berikut ini:

1. Layout tidak signifikan, spasi tidak berpengaruh.
2. Komentar diawali dengan sepasang tanda strip (--) untuk setiap barisnya.
3. Nama variabel dan nama tipe dapat terdiri dari huruf kecil, huruf besar, angka, tanda strip (-) dan spasi. Untuk nama variabel diawali dengan huruf kecil, sedangkan untuk nama tipe diawali dengan huruf besar.

Berikut ini adalah tabel *tag* untuk tipe-tipe yang termasuk dalam kelas universal yang terdapat dalam standar OSI X.208.

Tipe	Nomor Tag (desimal)	Nomor Tag (heksadesimal)
BOOLEAN	1	1
INTEGER	2	2
BIT STRING	3	3
OCTET STRING	4	4
NULL	5	5
OBJECT IDENTIFIER	6	6
REAL	9	9
ENUMERATED	10	A
SEQUENCE (OF)	16	10
SET (OF)	17	11
NUMERIC STRING	18	12
PRINTABLE STRING	19	13
UTCTime	23	17
GENERALIZED TIME	24	18
VISIBLE STRING	26	1A
BMP STRING	30	1E

Tabel II-2 Nomor *Tag Universal* ASN.1

1. Tipe Sederhana

Tipe sederhana adalah tipe yang tidak disusun oleh satu atau beberapa komponen, ia bersifat atomik. Contoh tipe sederhana adalah BIT STRING, BOOLEAN, INTEGER, OCTET STRING, PRINTABLE STRING, GENERALIZED TIME, UTCTime, dsb.

2. Tipe Terstruktur

Tipe terstruktur adalah tipe yang tersusun oleh satu atau beberapa komponen. Pada ASN.1 terdapat empat macam tipe terstruktur, yaitu:

- SEQUENCE, himpunan terurut dari satu atau lebih tipe.
- SEQUENCE OF, himpunan terurut dari nol atau lebih tipe tertentu (satu tipe).
- SET, himpunan tidak terurut dari satu atau lebih tipe.
- SET OF, himpunan tidak terurut dari nol atau lebih tipe tertentu (satu tipe).

Tipe terstruktur juga dapat memiliki satu atau beberapa komponen yang bersifat *optional;* artinya bisa ada, bisa juga tidak.

3. Tipe Turunan

Tipe turunan adalah tipe yang diperoleh dari tipe tertentu dengan menambahkan atau mengganti *tag* dari tipe tersebut. Tujuannya adalah untuk mencegah kebingungan apabila suatu tipe terstruktur tersusun atas beberapa komponen setipe namun semuanya bersifat *optional.*

Ada dua tipe turunan yaitu IMPLICIT dan EXPLICIT. Perbedaan antara kedua tipe tersebut adalah; tipe IMPLICIT diperoleh dengan cara mengganti *tag* dari tipe lain, sedangkan tipe EXPLICIT diperoleh dengan cara menambahkan *tag* di depan *tag* dari tipe lain.

Penambahan *tag* yang bersifat IMPLICIT dinyatakan dalam ASN.1 dengan kata kunci: "[**kelas *tag***] IMPLICIT". Adapun penambahan tag yang bersifat EXPLICIT dinyatakan dalam ASN.1 dengan kata kunci: "[**kelas *tag***] EXPLICIT".

4. Tipe Lainnya

Tipe lainnya dari ASN.1 adalah termasuk tipe CHOICE dan ANY.

Tipe CHOICE adalah tipe yang menunjukkan gabungan beberapa tipe alternatif. *Tag* dari tipe CHOICE ini tergantung dari tipe yang dipilih dari tipe alternatif tersebut.

Tipe ANY adalah tipe yang bebas, dia bisa merupakan tipe apa saja. Tipe ANY bisa merupakan tipe sederhana seperti INTEGER, OCTET STRING, BOOLEAN, dsb; bisa merupakan tipe terstruktur seperti SEQUENCE, SET, dsb; bisa merupakan tipe

turunan; dan juga bisa merupakan tipe CHOICE. *Tag* dari tipe ANY ini tergantung tipe yang menempatinya.

1.9.2 Distinguished Encoding Rules (DER)

Setiap kumpulan *byte* dalam bentuk DER selalu memiliki bentuk umum yaitu kumpulan *byte* tersebut terbagi menjadi tiga bagian. Ketiga bagian tersebut adalah:

1. *Identifier* (I), yaitu bagian yang mendefinisikan tipe dari objek abstrak yang didefinisikan oleh kumpulan *byte* DER ini.
2. Panjang (P), yaitu bagian yang mendefinisikan panjang dari nilai yang akan diterangkan kemudian.
3. Nilai (N), yaitu representasi asli kumpulan *byte* dari nilai objek abstrak yang direpresentasikan (belum dalam bentuk DER).

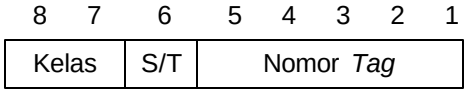
Pada sub bab selanjutnya akan dibahas satu-persatu dari ketiga bagian pembentuk representasi DER tersebut.

1. Identifier

Ada dua macam model pengkodean *identifier* dalam DER, yang pertama untuk *tag* bernomor kecil (< 31), yang kedua untuk *tag* bernomor besar (> 31).

Tag Kecil

Identifier untuk *tag* berukuran kecil hanya memerlukan 1 *byte* atau kita sebut 1 oktet karena 1 *byte* terbagi menjadi 8 *bit* (oktet berarti delapan).



Gambar II-19 Pengkodean Oktet *Identifier* (*tag* < 31)

Identifier terbagi lagi menjadi tiga bagian yaitu: 2 bit (bit ke-8 dan ke-7) mendefinisikan kelas, 1 bit (bit ke-6) mendefinisikan tipe sederhana atau terstruktur, sedangkan 5 bit sisanya (bit ke-5 s.d ke-1) mendefinisikan nomor *tag*.

Kombinasi bit ke-8 dan ke-7 mendefinisikan kelas dari sebuah tipe. Ada empat macam kombinasi yang dapat dilihat pada tabel berikut ini.

Kelas	Bit 8	Bit 7
Universal	0	0
Application	0	1
Context Specific	1	0
Private	1	1

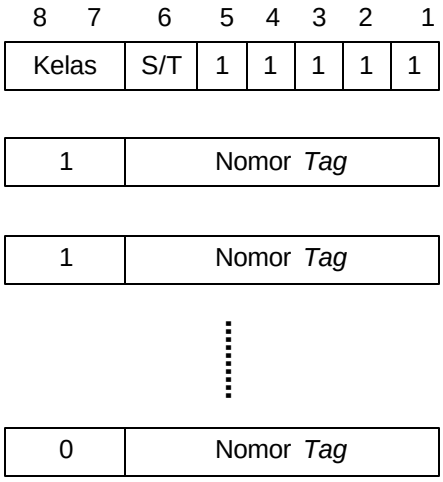
Tabel II-3 Bit ke-8 dan ke-7 pada *identifier*

Bit ke-6 membedakan antara tipe sederhana dan tipe terstruktur, "0" untuk tipe sederhana dan "1" untuk tipe terstruktur. Bit sisanya yaitu bit ke-5 s.d. bit ke-1 dipakai untuk menulis nomor *tag.*

Contoh *identifier* untuk tipe OCTET STRING, *identifier*-nya adalah `00000100` dalam *bit* atau `04` dalam heksadesimal. Hal terjadi karena OCTET STRING memiliki kelas *universal,* tipe sederhana, dan nomor *tag* 4.

Tag B*esar*

Identifier untuk *tag* berukuran besar dapat memakai sejumlah oktet. Aturan main untuk bit ke-8 s.d. bit ke-6 tetap sama dengan aturan untuk *tag* kecil, namun untuk bit ke-5 s.d. bit ke-1 berisi 1. Untuk mendefinisikan nomor *tag* digunakan oktet tambahan berikutnya yang jumlahnya tidak terbatas.



Gambar II-20 Pengkodean Oktet *Identifier* (*tag*>=31)

Oktet tambahan untuk nomor *tag* memiliki aturan: bit ke-8 bernilai 1 jika oktet ini bukan oktet terakhir yang dipakai untuk mendefinisikan nomor *tag*, sebaliknya jika ini adalah oktet yang terakhir, maka bit ke-8 dari oktet tersebut

bernilai 0; bit ke-7 s.d. ke-1 dari setiap oktet dipakai untuk mendefinisikan nomor *tag.*

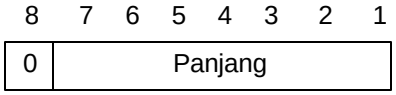
Contoh *identifier* untuk suatu tipe berkelas *application*, terstruktur, dan memiliki nomor *tag* 16383 adalah `01111111 11111111 01111111` dalam bentuk *bit*, atau 7F FF 7F dalam bentuk heksadesimal.

2. Panjang

Ada dua bentuk dalam mendefinisikan panjang pada DER yaitu bentuk pendek dan bentuk panjang. Pembagian bentuk ini tergantung dari panjang nilai yang akan dikodekan, bentuk pendek untuk nilai yang panjangnya lebih kecil atau sama dengan 127, sedangkan bentuk panjang untuk nilai yang panjangnya lebih besar daripada 127.

Bentuk Pendek

Bentuk pendek mempunyai aturan main: bit ke-8 selalu bernilai 0, sedangkan bit-bit sisanya mendefinisikan panjang.

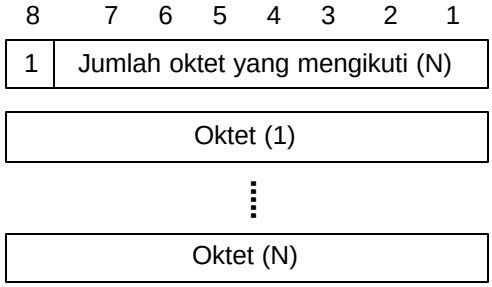


Gambar II-21 Bentuk Pendek Pengkodean Panjang

Contoh representasi DER untuk panjang 31 adalah `00011111` bit atau 1F dalam heksadesimal.

Bentuk Panjang

Bentuk panjang memiliki aturan: pada oktet pertama bit ke-8 bernilai 1, sedangkan bit-bit sisanya mendefinisikan jumlah oktet tambahan yang mengikuti oktet pertama ini; oktet-oktet selanjutnya mendefinisikan panjang dalam basis 256.



Gambar II-22 Bentuk Panjang Pengkodean Panjang

Sebagai contoh untuk panjang 43690, representasi panjang DER-nya adalah: `10000010 10101010 10101010` atau `82 AA AA`.

3. Nilai

Berikut ini adalah contoh pengkodean nilai beberapa tipe dalam DER. Tidak semua tipe tercakup di sini, untuk lebih jelasnya silakan mengacu pada referensi.

NULL

Null selalu dikodekan sebagai `05 00` (I=05, P=00, N=tidak ada).

BOOLEAN

Tipe BOOLEAN terdiri dari dua macam nilai yaitu *true* dan *false*. *True* dikodekan sebagai `01 01 FF` (I=01, P=01, N=FF) dan *false* dikodekan sebagai `01 01 00` (I=01, P=01, N=00).

OCTET STRING

Tipe OCTET STRING dikodekan sesuai dengan nilainya, hanya ditambahkan *identifier* dan panjang di depannya. Sebagai contoh untuk suatu OCTET STRING bernilai 05 07 AA FF dalam heksadesimal, dikodekan sebagai `04 04 05 07 AA FF` (I=04, P=04, N= 05 07 AA FF).

STRING

Beberapa tipe dapat dikategorikan sebagai tipe *string*.Tipe-tipe tersebut antara lain adalah NUMERIC STRING, PRINTABLE STRING dan VISIBLE STRING.

Pengkodean untuk tipe-tipe *string* ini adalah: I diisi sesuai dengan *tag* masing-masing *string*, L diisi sesuai dengan aturan yang berlaku, dan N diisi oleh kumpulan nilai ASCII dari untaian *string* tersebut.

Sebagai contoh suatu *string* bertipe VISIBLE STRING “ABC” dikodekan sebagai `1A 03 41 42 43` (I=1A, P=03, N=41 42 43).

INTEGER

Pengkodean integer menggunakan metode *two's complement* seperti contoh pada tabel berikut ini.

Integer	I	P	N
72	02	01	48
127	02	01	7F
-128	02	01	80
128	02	02	0080

Tabel II-4 Pengkodean Integer dengan DER

OBJECT IDENTIFIER (OID)

Nilai OID dalam ASN.1 dilambangkan sebagai `{nilai1 nilai2 nilai3 ... dst}`, contohnya OID untuk RSA Data Security Inc. adalah `{1 2 840 113549}`

Aturan pengkodean OID dalam DER adalah sebagai berikut:

- Oktet pertama bernilai 40 x nilai1 + nilai2.
- Oktet berikutnya apabila masih ada nilai3 dan seterusnya, masing-masing bernilai pengkodean dalam basis 128 untuk setiap nilainya.

Sebagai contoh oktet pertama untuk pengkodean RSA Data Security Inc adalah 40 x 1 + 2 = 42 = $2a_{16}$. Pengkodean untuk 840 = 6 x 128 + 48_{16} adalah `86 48` dan pengkodean untuk 113549 = 6 x 128^2 + 77_{16} x 128 + d_{16} adaalah `86 f7 0d`. Hasil pengkodean keseluruhannya adalah `06 06 2a 86 48 86 f7 0d`.

SEQUENCE dan SEQUENCE OF

SEQUENCE dan SEQUENCE OF dikodekan hanya dengan menambahkan *identifier* dan panjang di depang kumpulan oktet DER hasil pengkodean elemen-elemennya.

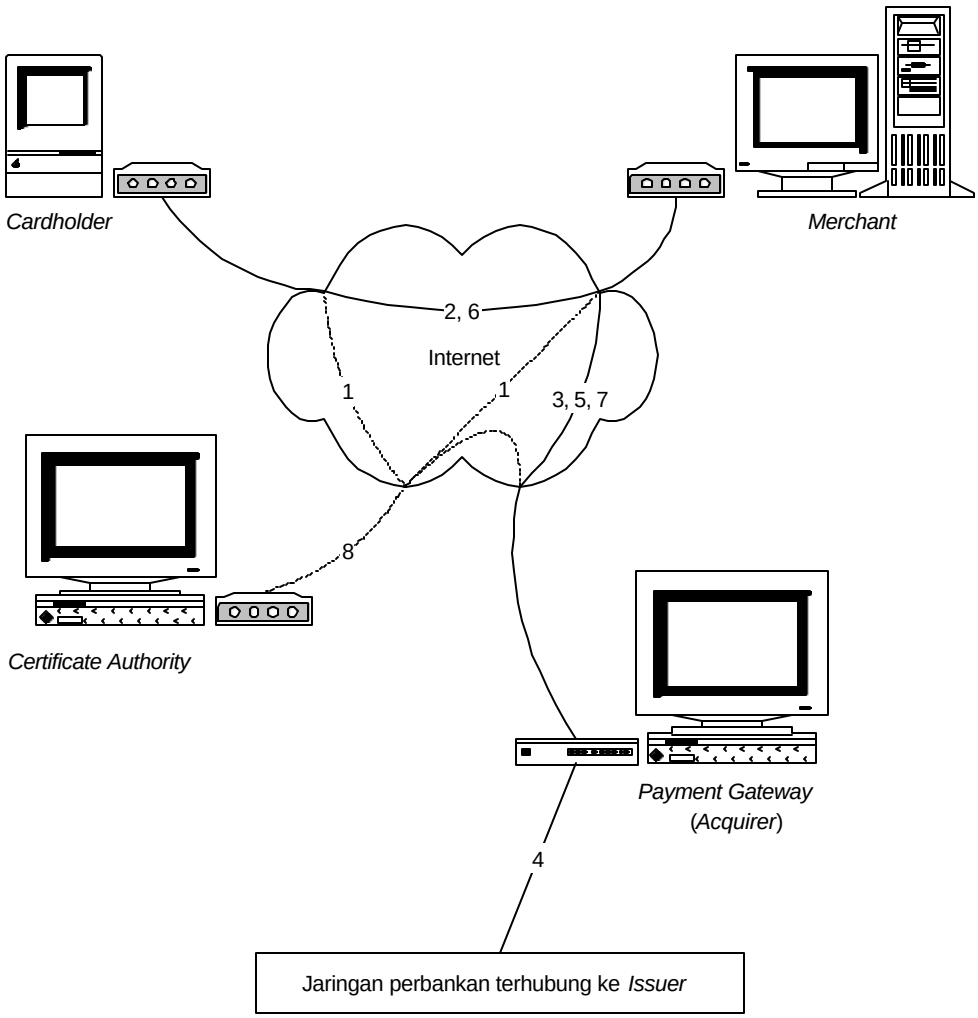
SET dan SET OF

SET dan SET OF dikodekan hanya dengan menambahkan *identifier* dan panjang di depang kumpulan oktet DER hasil pengkodean elemen-elemennya.

BAB III
ANALISA DAN PERANCANGAN

Bab ini bertujuan untuk mengulas secara terperinci protokol SET yang dideskripsikan secara umum pada bab dua. Ulasan berikut ini merupakan hasil analisa dan perancangan yang dilakukan pada waktu melakukan penelitian SET.

Sebelum kita benar-benar masuk dalam bab ini, penulis ingin menegaskan secara lebih rinci ruang lingkup penelitian ini. Kalau kita mengacu pada gambar di bawah ini, transaksi SET dapat kita definisikan dalam beberapa proses yaitu:



Gambar III-1 Transaksi SET

1. Proses registrasi; yaitu *cardholder*, *merchant, payment gateway* meminta sertifikat dari CA sebelum ikut bermain dalam transaksi SET (1).

2. Proses belanja; di mana *cardholder* memesan suatu barang dan menyetujui suatu sistem pembayarannya (2), kemudian mengirimkan OI dan PI kepada *merchant.* Setelah melakukan otorisasi (3,4,5), *merchant* mengirimkan hasil otorisasi tersebut kepada *cardholder* (6).
3. Proses otorisasi; yaitu *merchant* melakukan otorisasi dengan mengirimkan PI kepada *payment gateway* (3), setelah itu *payment gateway* melakukan otorisasi dengan *issuer* lewat jaringan perbankan (4), kemudian mengirimkan hasil otorisasinya kepada *merchant* (5).
4. Proses lanjutan dari proses otorisasi (7), bervariasi meliputi:
    - *Capture*

      Pada proses ini *merchant* menginginkan agar *payment gateway* meminta pembayaran dari *issuer* untuk transaksi yang telah diotorisasi sebelumnya
    - Pengembalian otorisasi

      Pada proses ini *merchant* ingin mengoreksi otorisasi pada suatu transaksi yang telah dilakukan sebelumnya. Sebagai contoh mengapa hal ini bisa terjadi adalah ketika *cardholder* ternyata ingin mengganti cara pembayaran yang sebelumnya telah diotorisasi, misalnya jika tadinya ingin membayar lunas ternyata sekarang minta dicicil.
    - Pengembalian *capture*

      Pada proses ini *merchant* ingin mengkoreksi *capture* pada suatu transaksi yang telah dilakukan sebelumnya. Hal ini bisa terjadi contohnya ketika *merchant* sudah melakukan *capture* pada suatu transaksi, ternyata ia tidak bisa memenuhi pesanan *cardholder*.
    - Kredit

      Proses ini pada hakekatnya sama dengan proses pengembalian *capture*, tapi sudah melewati batas waktu untuk pengembalian *capture*, sehingga terdapat perbedaan penanganan di bank.
    - Pengembalian kredit

      Proses ini terjadi jika ternyata kredit yang diberikan kepada *cardholder* tidak sesuai.

Mengacu pada penjelasan di atas, batasan masalah yang dapat penulis berikan adalah sebagai berikut:

1. Proses registrasi sertifikat untuk setiap entitas tidak masuk dalam ruang lingkup penelitian ini, diasumsikan mereka masing-masing sudah memiliki sertifikat.
2. Proses belanja antara *cardholder* dan *merchant* sudah terjadi; dan pada saat memasuki ruang lingkup penelitian ini, *merchant* sedang mengirimkan *authorization request* pada *payment gateway*.
3. Proses transaksi antara *merchant* dan *payment gateway* sebenarnya tidak hanya terdiri dari proses otorisasi saja, tetapi juga proses pengembalian otorisasi, proses *capture*, proses pengembalian *capture*, proses kredit, dan proses pengembalian kredit. Ruang lingkup penelitian ini hanya berkisar pada proses otorisasi saja, proses-proses yang lain diasumsikan dilakukan secara *off line*.

Pada bab analisa dan perancangan ini, implementasi protokol SET dapat dilihat dari dua sudut pandang, yaitu sudut pandang pesan (*message*) dan sudut pandang proses. Sudut pandang pesan berfokus pada struktur lapisan objek-objek pembentuk pesan-pesan pada protokol SET, sedangkan sudut pandang proses berfokuskan pada berbagai macam proses yang ada, baik di *merchant server* maupun di *payment gateway*.

1.10 PESAN

Pada sudut pandang pesan ini kita memfokuskan penalaran kita pada pesan-pesan dan komponen pembentuk pesan transaksi yang dikirimkan antar entitas pada protokol SET, khususnya pesan otorisasi (*Authorization Request* dan *Authorization Response*) sesuai dengan ruang lingkup penelitian ini.

Sturuktur objek pembentuk pesan SET dapat kita lihat dalam 6 lapisan. Pembagian lapisan ini berdasarkan ketergantungan objek-objek pada lapisan atas dengan objek-objek pada lapisan bawahnya. Maksud ketergantungan disini adalah suatu objek pada lapisan atas tersusun atau membutuhkan (sebagai alat bantu) satu atau lebih objek pada lapisan bawahnya.

Garis batas antar lapisan ada yang terputus-putus dan ada yang tidak. Garis batas yang tidak terputus melambangkan bahwa semua objek pada lapisan-lapisan diatas garis batas itu bergantung pada lapisan yang tepat berada di bawah garis batas itu, sedangkan yang terputus-putus melambangkan bahwa tidak semua objek pada lapisan-lapisan di atas garis batas itu bergantung pada lapisan yang tepat berada di bawah garis batas itu.

<table>
<tr><td>5</td><td colspan="3">Pembungkus Pesan (Message Wrapper)</td></tr>
<tr><td>4</td><td colspan="2">Pesan Kesalahan
(Error)</td><td>Pesan (Message)</td></tr>
<tr><td>3</td><td colspan="3">Komponen Pesan</td></tr>
<tr><td>2</td><td colspan="3">Kriptografi SET</td></tr>
<tr><td>1</td><td>PKCS</td><td>X.509</td><td>OAEP
(Optimal Asymmetric Encryption Padding)</td></tr>
<tr><td>0</td><td colspan="3">DER
(Distinguished Encoding Rules)</td></tr>
</table>

Gambar III-2 Struktur Lapisan Objek SET

Seluruh spesifikasi objek pada gambar ditulis dalam *Abstract Syntax Notation One* (ASN.1).

1.10.1 Objek DER

Objek DER tidak berfungsi sebagai objek penyusun untuk lapisan-lapisan atasnya, namun ia berfungsi sebagai alat bantu untuk mengubah objek-objek pada lapisan-lapisan atasnya menjadi kumpulan *byte* dalam representasi DER. Mengenai objek ini akan dibahas lebih dalam di bab implementasi.

1.10.2 PKCS dan X.509

Tidak semua tipe dalam PKCS dan X.509 dipakai sebagai objek dalam protokol SET. Kebanyakan tipe yang diambil berasal dari PKCS #7, dua dari PKCS #9 dan satu dari X.509.

1. AlgorithmIdentifier

```
AlgorithmIdentifier ::= SEQUENCE {
  algorithm    ALGORITHM.&id ({SupportedAlgorithms}),
  parameters ALGORITHM.&Type ({SupportedAlgorithms}{ @algorithm}) OPTIONAL
}

SupportedAlgorithms ALGORITHM ::= { ... }
```

Gambar III-3 Algorithm Identifier

AlgorithmIdentifier didefinisikan dalam X.509 dipakai untuk meng-identifikasikan suatu algoritma.

Suatu algoritma didefinisikan dengan dua komponen pada ASN.1, yaitu:

1. algorithm memiliki tipe OBJECT IDENTIFIER (OID) yang berisi suatu nilai yang sudah ditentukan untuk algoritma tersebut, contohnya OID untuk algoritma DES dengan *Cypher Block Chainning* atau DES-CBC adalah: {iso(1) identified-organization(3) oiw(14) secsig(3) 2 7}.
2. parameters adalah parameter-parameter untuk algoritma tersebut, seperti algoritma DES-CBC tersebut memiliki parameter untuk *internal vector*-nya. parameters ini bersifat *optional* dan bentuknya bebas.

2. ContentInfo

```
ContentInfo ::= SEQUENCE {
        contentType       ContentType,
        content           [0] EXPLICIT ANY DEFINED BY ContentType OPTIONAL
}

ContentType ::= OBJECT IDENTIFIER
```

Gambar III-4 Content Info

ContentInfo bertujuan mendefinisikan suatu nilai berdasarkan tipe dan isinya [PKCS #7].

ContentInfo memiliki dua buah *field*, yaitu:

1. Tipe dari nilai tersebut (contentType), *field* ini berisi OID yang mendefinisikan tipe tersebut.
2. Isi dari nilai tersebut (content), *field* ini bersifat *optional* dan bentuknya bebas.

Sebagai contoh pada protokol SET; jika content berisi PI, maka contentType berisi OID bernilai {joint-iso-itu-t(2) internationalRA(23) set(42) 0 4}.

3. IssuerAndSerialNumber

```
IssuerAndSerialNumber ::= SEQUENCE {  -- Uniquely identifies certificate
   issuer         Name,
   serialNumber   CertificateSerialNumber
}

CertificateSerialNumber ::= INTEGER
```

Gambar III-5 Issuer And Serial Number

IssuerAndSerialNumber bertujuan untuk mengidentifikasi sebuah sertifikat berdasarkan nama badan yang mengeluarkan sertifikat tersebut (issuer) dan nomor seri sertifikat tersebut (serialNumber) [PKCS #7].

4. DigestedData

```
DigestedData ::= SEQUENCE {
   ddVersion        INTEGER { ddVer0(0) } (ddVer0),
   digestAlgorithm  AlgorithmIdentifier {{DigestAlgorithms}},
   contentInfo      ContentInfo,
   digest           Digest
}

Digest ::= OCTET STRING (SIZE(1..20))
```

Gambar III-6 Digested Data

DigestedData bertujuan untuk menyimpan informasi yang berhubungan dengan proses *digest* [PKCS #7].

Proses pembuatan DigestedData adalah sebagai berikut:

1. Dokumen yang akan di-*digest* disimpan dalam *field* contentInfo pada DigestedData.
2. Dokument tersebut di-*digest* dengan algoritma yang tertulis pada *field* digestAlgorithm, dan hasilnya disimpan dalam *field* Digest.

Proses verifikasi DigestedData adalah sebagai berikut:

1. Si penerima membuat *digest* lain yang diperoleh dengan men-*digest* dokumen asli yang tertera pada *field* contentInfo.
2. Si penerima membandingkan *digest* lain tersebut dengan *digest* yang tersimpan pada *field* digest.

5. SignedData

```
SignedData ::= SEQUENCE {                              -- PKCS#7
   sdVersion          INTEGER { sdVer2(2) } (sdVer2),
   digestAlgorithms   DigestAlgorithmIdentifiers,
   contentInfo        ContentInfo,
   certificates       [2] IMPLICIT Certificates  OPTIONAL,
   crls               [3] IMPLICIT CRLSequence  OPTIONAL,
   signerInfos        SignerInfos
}

SignerInfos ::= SEQUENCE OF SignerInfo
   (WITH COMPONENTS { ..., authenticatedAttributes   PRESENT,
                      unauthenticatedAttributes ABSENT })

SignerInfo ::= SEQUENCE {
   siVersion                    INTEGER { siVer2(2) } (siVer2),
   issuerAndSerialNumber        IssuerAndSerialNumber,
   digestAlgorithm              AlgorithmIdentifier {{DigestAlgorithms}},
   authenticatedAttributes      [0] EXPLICIT
                                AttributeSeq {{Authenticated}}  OPTIONAL,
   digestEncryptionAlgorithm    AlgorithmIdentifier     {{DigestEncryptionAlgorithms}},
   encryptedDigest              EncryptedDigest,
   unauthenticatedAttributes    [1] EXPLICIT AttributeSeq {{...}}  OPTIONAL
 }
```

Gambar III-7 Signed Data

SignedData bertujuan menyimpan informasi yang berhubungan dengan proses tanda tangan digital [PKCS #7].

SignedData memiliki beberapa *field*, yaitu:

1. Versi dari tipe SignedData ini (sdVersion).
2. Algoritma-algoritma pembuat *digest* (digestAlgorithms), dapat terdiri dari beberapa algoritma.
3. Dokumen yang ditandatangani, tersimpan dalam contentInfo.
4. Sertifikat-sertifikat milik penandatangan (certificates).
5. Daftar sertifikat-sertifikat yang sudah tidak berlaku (crls).
6. Info penandatangan-penandatangan (signerInfos), merupakan kumpulan dari beberapa info penandatangan (SignerInfo). Di dalam SignerInfo ini tersimpan tanda tangan digital. Jadi sebuah dokumen bisa memiliki beberapa tanda tangan, satu tanda tangan satu SignerInfo.

Informasi yang disimpan pada setiap SignerInfo adalah:

1. Versi pada tipe SignerInfo ini (siVersion).
2. Nama badan dan nomor seri sertifikat badan yang mengeluarkan sertifikat si penandatangan (issuerAndSerialNumber).
3. Algoritma pembuatan *digest* dari dokumen yang ditandatangani (digest Algorithm).
4. Atribut-atribut tentang dokumen yang terautentikasi (authenticatedAttributes) berisi atribut bertipe ContentType dan atribut bertipe MessageDigest (*digest* dari dokumen tersebut) [PKCS #9]. *Field* ini bersifat *optional.*
5. Algoritma pengenkripsian *digest* dari dokumen yang ditandatangani (digestEncryptionAlgorithm).
6. Atribut-atribut lain yang tidak terautentikasi [PKCS #9]. *Field* ini bersifat *optional.*

Proses pembuatan SignedData ini adalah sebagai berikut:

1. Untuk setiap penandatangan, *digest* dibuat dari dokumen dalam contentInfo sesuai dengan algoritma pembuatan *digest* yang tertera di masing-masing SignerInfo pada *field* digestAlgorithm.
2. Untuk setiap hasil *digest* yang dibuat pada nomor 1, dienkripsi dengan kunci privat tanda tangan milik penandatangan menggunakan algoritma enkripsi

digest yang terkait dengan masing-masing *digest* tersebut, hasilnya adalah yang kita sebut tanda tangan digital.

3. Setiap tanda tangan tersebut dan informasi-informasi lainnya ditaruh pada SignerInfo yang terkait dengannya (tanda tangan ditaruh dalam *field* encryptedDigest). Seluruh sertifikat penandatangan disatukan dan disimpan pada *field* certificates pada SignedData.
4. Seluruh algoritma pembuatan *digest* (digestAlgorithm) dari setiap signerInfo dikumpulkan menjadi satu dan ditaruh dalam *field* digestAlgorithms pada SignedData.

Si penerima melakukan verifikasi SignedData tersebut dengan cara mendekripsi seluruh tanda tangan (encryptedDigest) dengan kunci publik yang didapat dari sertifikat-sertifikat pengirim (certificates) setelah itu membandingkan dengan semua *digest* baru yang dibuat (prosesnya seperti pada langkah 1 dan 2 pada pembuatan SignedData).

6. EnvelopedData

```
EnvelopedData ::= SEQUENCE {
   edVersion           INTEGER { edVer1(1) } (edVer1),
   recipientInfos      RecipientInfos,
   encryptedContentInfo  EncryptedContentInfo
}

EncryptedContentInfo ::= SEQUENCE {
   contentType      ContentType,
   contentEncryptionAlgorithm
                AlgorithmIdentifier {{ContentEncryptionAlgorithms}},
   encryptedContent  [0] IMPLICIT EncryptedContent  OPTIONAL
}

EncryptedContent ::= OCTET STRING

RecipientInfos ::= SEQUENCE OF RecipientInfo

RecipientInfo ::= SEQUENCE {
   riVersion            INTEGER { riVer0(0) } (riVer0),
   issuerAndSerialNumber   IssuerAndSerialNumber,
   keyEncryptionAlgorithm  AlgorithmIdentifier {{KeyEncryptionAlgorithms}},
   encryptedKey          EncryptedKey
}

EncryptedKey ::= OCTET STRING (SIZE(1..128))
```

Gambar III-8 Enveloped Data

EnvelopedData merepresentasikan amplop digital yang kita kenal pada bab dua [PKCS #7]. Sebuah dokumen dienkripsi menggunakan kunci simetris dan kunci simetris tersebut dienkripsi menggunakan kunci publik pertukaran pesan milik si penerima. Dokumen yang terenkripsi disimpan pada tipe EncryptedContentInfo sedangkan kunci simetris yang terenkripsi disimpan pada tipe RecipientInfo. EnvelopedData memiliki

sebuah EncryptedContentInfo dan dapat memiliki beberapa RecipientInfo, dengan kata lain EnvelopedData dapat ditujukan pada beberapa penerima.

Proses pembuatan EnvelopedData adalah:

1. Dokumen yang akan diamplopkan, dienkripsi dengan sebuah kunci simetris yang dibuat secara acak. Dokumen yang terenkripsi kemudian disimpan dalam *field* encryptedContent pada EncryptedContentInfo. Algoritma enkripsi simetris yang digunakan juga disimpan dalam *field* contentEncryptionAlgorithm. *Field* contentType pada EncryptedContentInfo diisi dengan OID yang bersesuaian dengan dokumen tersebut.
2. Kunci simetris yang digunakan pada nomor satu dienkripsi dengan kunci publik si penerima dan hasilnya disimpan dalam *field* encryptedKey pada RecipientInfo. *Field-field* lainnya yaitu issuerAndSerialNumber dan keyEncryptionAlgorithm juga dilengkapi.
3. Proses tahap dua juga dilakukan untuk penerima lainnya, sehingga nantinya terdapat satu atau lebih RecipientInfo dan semuanya disatukan dalam *field* recipientInfos dalam EnvelopedData.

Ketika si penerima menerima EnvelopedData ini, ia membuka salah satu RecipientInfo yang terasosiasi dengan kunci privat miliknya melalui issuerAndSerialNumber yang tertera di sana. Setelah itu ia mendekripsi encryptedKey dan memperoleh kunci simetris yang selanjutnya dipakai untuk membuka encryptedContent pada EncryptedContentInfo.

7. EncryptedData

```
EncryptedData ::= SEQUENCE {
   version              INTEGER { enVer0(0) } (enVer0),
   encryptedContentInfo  EncryptedContentInfo
}
```

Gambar III-9 Encrypted Data

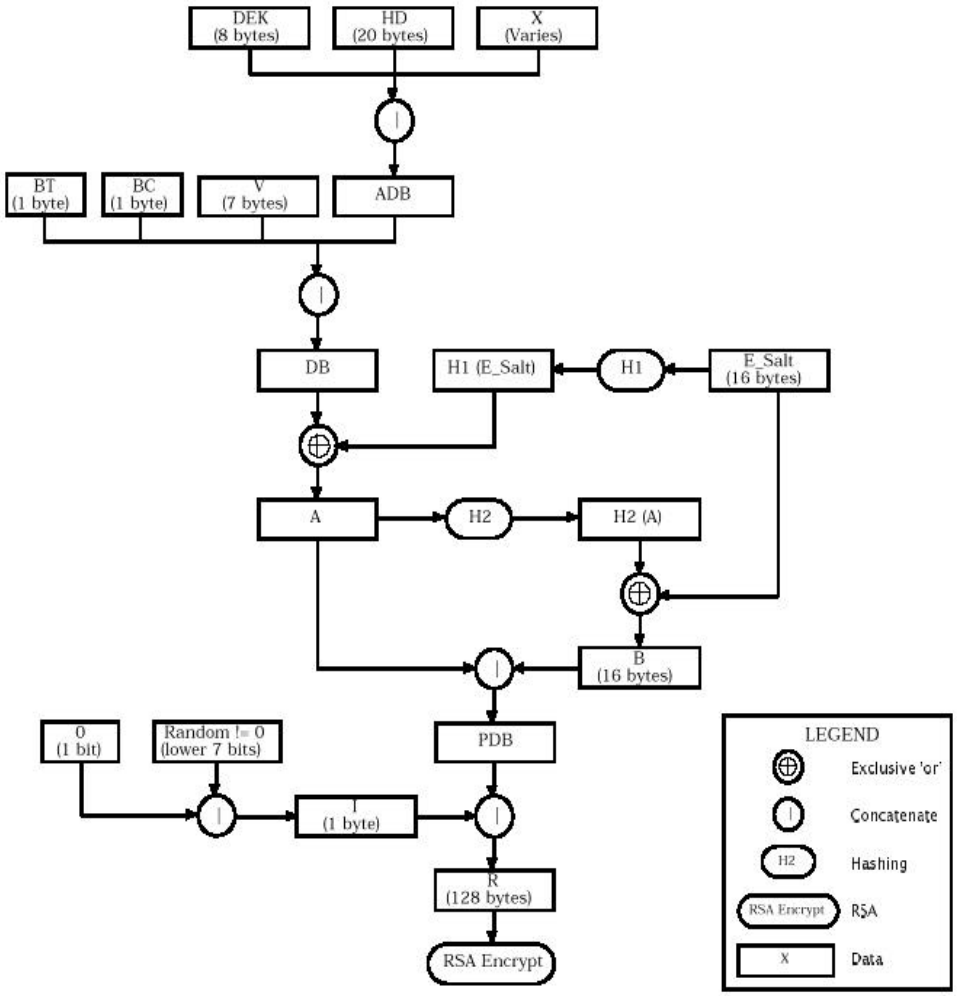
EncryptedData hanya menyimpan informasi tentang dokumen yang terenkripsi, tersimpan dalam *field* enryptedContentInfo [PKCS #7]. EncryptedData tidak menangani pendistribusian kunci seperti pada SignedData dan EnvelopedData. Pendistribusian kunci diasumsikan dilakukan dengan cara lain.

1.10.3 *Optimal Asymmetric Encryption Padding* (OAEP)

Blok *Optimal Asymmetric Encryption Padding* (OAEP) digunakan untuk mengisi *field* encryptedKey pada RecipientInfo dalam EnvelopedData. Blok ini dimaksudkan

untuk dapat menyimpan kunci DES, *hash* dari data, dan parameter ekstra yang digunakan pada E, EH, EX, dan EXH.

OAEP bertujuan untuk melindungi agar bagian-bagian dari blok tersebut tidak dapat dikenali, mengingat ada kriptoanalisis[4] yang membuat beberapa *bit* lebih mudah dapat dibedakan dari *bit-bit* lainnya.



Gambar III-10 Optimal Asymmetric Encryption Padding (OAEP)

Sesuai dengan gambar ada beberapa langkah membentuk blok OAEP, yaitu:

[4] Teknik untuk memecahkan suatu sandi atau pesan terenkripsi.

1. **ADB = DEK | HD | X**

Nama	Keterangan	Panjang (byte)
ADB	*Actual Data Block*	102
DEK	Kunci DES	8
HD	*Hash* dari data	20
X	Ekstra parameter	variasi

Tabel III-1 OAEP - Actual Data Block

2. **DB = BT | BC | V | ADB**

Nama	Keterangan	Panjang (byte)
DB	*Data Block*	111
BT	Sebuah *byte* yang berisi 0x03, bertujuan untuk meng-identifikasi format blok	1
BC	*Block Content*, blok yang meng-identifikasi isi ADB. Bit order tertinggi berisi 1 jika ADB mengandung HD dan 0 jika sebaliknya. Bit-bit order yang lebih rendah berikutnya meng-identifikasi jenis data pada ekstra parameter(angka di dalam tanda kurung menandakan HD ada). 00 (80) Tidak ada 01 (81) PANData 02 (82) PANData0 03 (83) PANToken 04 (84) PANOnly 05 (85) AcctData	1
V	7 buah *byte* berisi 0, bertujuan untuk memverifikasi kebenaran dekripsi dari blok RSA. (Cat: kombinasi BT dan V memberikan 8 nilai *byte* tetap yang mem-verifikasi kebenaran dekripsi.	7

Tabel III-2 OAEP - Data Block

3. A = DB H1(E_Salt)

Nama	Keterangan	Panjang (byte)
A		111
H1(t)	H1 dibentuk dari ekspresi berikut ini: H1(x|00)|H1(x|01)|…… H1(x|05) Di mana: • H(x|n) dibuat sebanyak di-butuhkan, dalam hal ini 6 kali • n sebuah *byte* penghitung, dalam hal ini (00 – 05) • H adalah SHA-1, menghasilkan 20 *byte hash*. • t adalah parameter untuk H1, dalam hal ini adalah E_Salt	111
E_Salt	16 *byte salt* acak.	16

Tabel III-3 OAEP - A

4. B = H2(A) E_Salt

Nama	Keterangan	Panjang (byte)
B		16
H2(t)	H2 mengembalikan 16 *byte hash* SHA-1	16

Tabel III-4 OAEP - B

5. PDB = A | B

Nama	Keterangan	Panjang (byte)
PDB	*Padded Data Block*	127

Tabel III-5 OAEP – Padded Data Block

6. R = I | PDB

Nama	Keterangan	Panjang (byte)
I	*Byte* inisial, adalah sebuah *byte* acak yang *bit* order tertingginya adalah 0, sedangkan *bit* sisanya acak. Nilai *Byte* tersebut tidak	1

	boleh 0	

Tabel III-6 OAEP - R

Adapun untuk mendapatkan DEK, HD, X kembali dari R adalah sebagai berikut:

1. Dari R didapatkan I dan PDB, verifikasi I tidak boleh 0.
2. Dari PDB didapatkan A dan B.
3. Dapatkan E_Salt = H2(A) B
4. Dapatkan DB = A H1(E_Salt)
5. Dari DB didapatkan BT, BC, V, dan ADB. Lakukan verifikasi pada BT, BC, dan V sesuai tabel pada langkah dua pembentukan blok OAEP.
6. Dari ADB didapatkan DEK, HD, dan X.

1.10.4 Operator Kriptografi SET

Operator kriptografi SET merupakan kombinasi dari objek-objek yang didefinisikan dalam PKCS, X509, dan OAEP yang bertujuan untuk menjamin tiga isu keamanan SET yaitu autentisitas, kerahasiaan dan integritas data-data yang dikirimkan di atas protokol SET [Set97c].

1. Hash (H)

```
H { ToBeHashed } ::= OCTET STRING (SIZE(1..20)) (CONSTRAINED BY {
      -- HASH is an n-byte value, which is the results --
      -- of the application of a valid digest procedure    --
      -- applied to -- ToBeHashed })
```

Gambar III-11 Hash (H)

Hash (H) menerima masukan yaitu dokumen yang akan di-*hash* (ToBeHashed), dan hasilnya adalah OCTET STRING berukuran 1-20 dalam representasi DER.

2. Data Ter-digest (Digested Data)

```
DD { ToBeHashed } ::= DetachedDigest
   (CONSTRAINED BY { -- digest of the DER representation, including --
            -- the tag and length octets, of -- ToBeHashed })

DetachedDigest ::= DigestedData                    -- No parameter
 (WITH COMPONENTS {..., contentInfo (WITH COMPONENTS {..., content ABSENT}) })
```

Gambar III-12 Data Ter-*digest* (DD)

Data ter-*digest* (DD) pada hakekatnya sama dengan H tapi dalam representasi yang berbeda, dokumen yang akan di-*digest* juga harus dalam bentuk DER.

DD adalah DetachedDigest dengan syarat dokumen yang akan di-*digest* (ToBeHashed) harus dalam bentuk DER, sedangkan DetachedDigest adalah DigestedData dengan syarat *field* content yang seharusnya menjadi tempat untuk ToBeHashed pada ContentInfo yang terdapat di DigestedData absen keberadaannya.

3. Koneksi (*Linkage*)

```
L { T1, T2 } ::= SEQUENCE {                -- Linkage from t1 to t2
   t1  T1,
   t2  DD { T2 }                          -- PKCS#7 DigestedData
}
```

Gambar III-13 Koneksi (L)

Koneksi atau *linkage* (L) antara kedua dokumen (T1 dan T2), merupakan SEQUENCE dari T1 dan DD{T2} untuk menandakan bahwa T1 mempunyai koneksi dengan T2. Namun koneksi ini tidak berlaku bolak-balik, yaitu L{T1,T2} tidak sama dengan L{T2,T1}.

4. Hash Menggunakan Kunci (Key Hash Mechanism)

```
HMAC { ToBeHashed, Key } ::= Digest
  (CONSTRAINED BY { -- HMAC keyed digest of -- ToBeHashed,
                                    -- using -- Key })

Digest ::= OCTET STRING (SIZE(1..20))
```

Gambar III-14 Hash Menggunakan Kunci (HMAC)

Hash menggunakan kunci (HMAC) adalah suatu tipe yang memiliki dua masukan yaitu dokumen yang akan di-*hash* (ToBeHashed) dan kunci untuk melakukan *hash*(Key), dan direpresentasikan dalam bentuk Digest pada DER, di mana Digest itu adalah OCTET STRING berukuran 1-20 *byte*.

5. Tanda Tangan (*Signature Only*)

```
SO { SIGNER, ToBeSigned } ::= SignedData          -- Detached content
   (CONSTRAINED BY { SIGNER, -- signs -- ToBeSigned })
   (WITH COMPONENTS { ..., contentInfo
   (WITH COMPONENTS{ ..., content ABSENT }) } ^
    WITH COMPONENTS { ..., signerInfos (SIZE(1..2)) })
```

Gambar III-15 Tanda Tangan (SO)

SO adalah SignedData dengan syarat *field* content pada *field* contentInfo dikosongkan, artinya dokumen yang ditandatangani tidak disimpan dalam SO; dan juga

jumlah penandatangan (SignerInfo) hanya boleh satu atau dua penandatangan. Si penandatangan dilambangkan dengan SIGNER.

Kalau kita ingat pada diagram proses pada bab dua, tanda tangan (SO) ini digambarkan sebagai  , belah ketupat dengan inisial penandatangan di tengahnya.

6. Pesan Tertandatangani (*Signed Message*)

```
S { SIGNER, ToBeSigned } ::= SignedData
   (CONSTRAINED BY { SIGNER, -- signs -- ToBeSigned })
   (WITH COMPONENTS { ..., contentInfo
      (WITH COMPONENTS {
            ..., content PRESENT }) } ^
    WITH COMPONENTS { ..., signerInfos (SIZE(1..2)) })
```

Gambar III-16 Pesan Tertandatangani (S)

Berbeda dengan SO, S adalah SignedData dengan syarat *field* content pada *field* contentInfo harus ada, artinya dokumen yang ditandatangani diikutsertakan dalam S. Selain itu jumlah penandatangan dibatasi satu atau dua penandatangan.

Kalau kita ingat pada diagram proses pada bab dua, pesan tertandatangani (S) ini digambarkan sebagai 

, persegi panjang (pesan) yang dibawahnya terdapat belah ketupat (tanda tangan).

7. Enkripsi Asimetris (*Asymmetric Encryption*)

```
E { RECIPIENT, ToBeEnveloped } ::= EnvelopedData
   (CONSTRAINED BY { ToBeEnveloped, -- is encrypted, and the --
    -- session key is encrypted using the --
    -- public key of -- RECIPIENT } )
   (WITH COMPONENTS {..., encryptedContentInfo
   (WITH COMPONENTS { ..., encryptedContent PRESENT }) } ^
    WITH COMPONENTS { ..., recipientInfos (SIZE(1)) })
```

Gambar III-17 Enkripsi Asimetris (E)

E {RECIPIENT, ToBeEnveloped} merupakan EnvelopedData, di mana *field* encryptedContent pada EncryptedContentInfo ada dan jumlah RecipientInfo hanya satu penerima. RECIPIENT melambangkan data penerima. Dalam kenyataannya RECIPIENT adalah sertifikat si penerima yang didalamnya terdapat kunci publik si penerima.

8. Enkripsi dengan Integritas (*Integrity Encryption*)

```
EH { RECIPIENT, ToBeEnveloped } ::= E {
    RECIPIENT,
    ToBeEnveloped
} (CONSTRAINED BY { -- H(ToBeEnveloped) included in the OAEP block -- })
```

Gambar III-18 Enkripsi dengan Integritas (EH)

EH {RECIPIET, ToBeEnveloped} merupakan E {RECIPIENT, ToBeEnveloped}, di mana *hash* dari ToBeEnveloped disertakan dalam blok OAEP.

9. Enkripsi Ekstra (*Extra Encryption*)

```
EX { RECIPIENT, ToBeEnveloped, Parameter } ::= E {
   RECIPIENT,
   L { ToBeEnveloped, Parameter }
}(CONSTRAINED BY { Parameter – data is included in the OAEP block -- })
```

Gambar III-19 Enkripsi Ekstra (EX)

EX {RECIPIENT, ToBeEnveloped, Parameter} merupakan E {RECIPIENT, L{ToBeEnveloped, Parameter}}; di mana Parameter juga disertakan dalam blok OAEP.

10. Enkripsi Ekstra dengan Integritas (*Extra Encryption with Integrity*)

```
EXH { RECIPIENT, ToBeEnveloped, Parameter } ::= EX {
      RECIPIENT,
      ToBeEnveloped,
      Parameter
} (CONSTRAINED BY { -- H(ToBeEnveloped) included in the OAEP block -- })
```

Gambar III-20 Enkripsi Ekstra dengan Integritas (EXH)

EXH {RECIPIENT, ToBeEnveloped, Parameter} merupakan EX {RECIPIENT, ToBeEnveloped, Parameter}, di mana *hash* dari ToBeEnveloped juga disertakan dalam blok OAEP.

11. Enkripsi Simetris (*Symmetric Encryption*)

```
EK { KeyData, ToBeEnveloped } ::= EncryptedData
   (CONSTRAINED BY { ToBeEnveloped, -- encrypted with -- KeyData } )
   (WITH COMPONENTS { ..., encryptedContentInfo
   (WITH COMPONENTS { ..., encryptedContent PRESENT}) })
```

Gambar III-21 Enkripsi Simetris (EK)

EK {KeyData, ToBeEnveloped} adalah EncryptedData, di mana komponen EncryptedContent ada dalam EncryptedContentInfo.

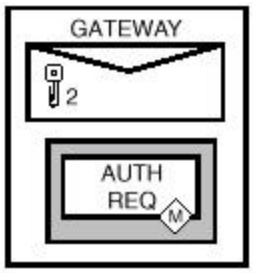
12. Enkapsulasi Sederhana dengan Tanda Tangan

```
Enc { SIGNER, RECIPIENT, T } ::= E {
    RECIPIENT,
    S { SIGNER, T }
}
```

Gambar III-22 Enkapsulasi Sederhana dengan Tanda Tangan (Enc)

Enc {SIGNER, RECIPIENT, T} merupakan E{RECIPIENT, S{SIGNER,T}}.

Pada diagram transaksi SET di bab dua, Enc digambarkan seperti gambar disamping. Pada gambar tersebut yang menjadi SIGNER adalah *merchant* (inisial M), yang menjadi RECIPIENT adalah *gateway* dan T adalah *Authorization Request.*




```
EncK { KeyData, SIGNER, T } ::= EK {
    KeyData,
    S { SIGNER, T }
}
```

Gambar III-23 Enkapsulasi Sederhana dengan Tanda Tangan (EncK)

EncK {KeyData, SIGNER, T} merupakan EK{KeyData, S{SIGNER, T}}, mengenai EK dan S dapat dilihat pada penjelasan sebelumnya.

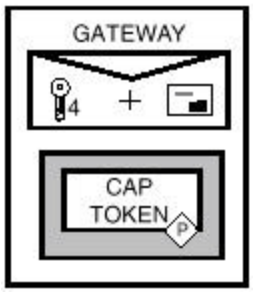
13. Enkapsulasi Ekstra dengan Tanda Tangan

```
EncX { SIGNER, RECIPIENT, T, Parameter } ::= E {
    RECIPIENT,
    SEQUENCE {
    t T,
    s SO { SIGNER, SEQUENCE { t T, p Parameter } }
    }
} (CONSTRAINED BY { Parameter -- data, which shall contain a fresh --
   -- nonce 'n', is included in the OAEP block. -- } )
```

Gambar III-24 Enkapsulasi Extra dengan Tanda Tangan (EncX)

EncX {SIGNER, RECIPIENT, T, Parameter} merupakan E, di mana komponen ToBeEnveloped-nya diisi dengan SEQUENCE {t T, s SO {SIGNER, SEQUENCE {t T, p Parameter}}}; selain itu Parameter juga disertakan dalam blok OAEP.

Pada gambar; SIGNER adalah *gateway* (inisial P pada belah ketupat), RECIPIENT adalah *gateway*, T adalah *capture token*, dan Parameter adalah PANToken (digambarkan dalam bentuk kartu pada amplop). Tanda tangan *gateway* (belah ketupat dengan inisial P) adalah SO {*gateway*, SEQUENCE {*capture token*, PANToken}}.



14. Enkapsulasi dengan Bagasi Eksternal Terenkripsi (*Encapsulation with External Encrypted Baggage*)

```
EncB { SIGNER, RECIPIENT, T, Baggage } ::= SEQUENCE {
    enc     Enc { SIGNER, RECIPIENT, L { T, Baggage } },
    baggage Baggage
}
```

Gambar III-25 Enkapsulasi dengan Bagasi Eksternal Terenkripsi (EncB)

EncB {SIGNER, RECIPIENT, T, Baggage} merupakan SEQUENCE dari Enc {SIGNER, RECIPIENT, L{T,Baggage}} dan Baggage. Baggage diasumsikan sudah dalam bentuk terenkripsi.

```
EncBX { SIGNER, RECIPIENT, T, Baggage, Parameter } ::= SEQUENCE {
    encX     EncX { SIGNER, RECIPIENT, L { T, Baggage }, Parameter },
    baggage  Baggage
}
```

Gambar III-26 Enkapsulasi dengan Bagasi Eksternal Terenkripsi (EncBX)

EncBX {SIGNER, RECIPIENT, T, Baggage, Parameter} merupakan SEQUENCE dari EncX {SIGNER, RECIPIENT, L{T,Baggage}, Parameter} dan Baggage yang sudah dalam bentuk terenkripsi.

1.10.5 Komponen Pesan

1. TransIDs

TransIDs menyediakan seluruh informasi yang berguna untuk membedakan suatu transaksi dan karakteristiknya secara unik, di mana pesan-pesan (*messages*) merupakan bagian dari itu.

```
TransIDs ::= SEQUENCE {
    lid-C      LocalID,
    lid-M      [0] LocalID  OPTIONAL,
    xid        XID,
    pReqDate   Date,
    paySysID   [1] PaySysID  OPTIONAL,
    language   Language         -- Cardholder requested session language
 }

LocalID ::= OCTET STRING (SIZE(1..20))

XID ::= OCTET STRING (SIZE(20))

Date ::= GeneralizedTime

PaySysID ::= VisibleString (SIZE(1..ub-paySysID))

Language ::= VisibleString (SIZE(1..ub-RFC1766-language))
```

Gambar III-27 TransIDs

TransIDs terdiri dari beberapa *field* yang akan dijelaskan dalam tabel berikut ini.

Nama *field*	Keterangan
LID-C	Label yang sesuai yang dibuat oleh dan untuk sistem *cardholder.*
LID-M	Label yang sesuai yang dibuat oleh dan untuk sistem *merchant.*
XID	Identitas keseluruhan yang unik per transaksi.
PReqDate	Tanggal pembelian yang dibuat oleh *merchant* pada pInitRes

	atau oleh *cardholder* pada pReq.
PaySysID	Digunakan oleh beberapa kartu pembayaran sebagai label transaksi untuk otorisasi sampai saat ini
Language	Bahasa yang digunakan *cardholder*.

Tabel III-7 Keterangan TransIDs

XID dibuat oleh sistem *merchant* pada PInitRes (*Payment Initiate Response*); kecuali tidak ada pInitRes, maka XID dibuat oleh sistem *cardholder* pada PReq (*Purchase Request*).

2. RRTags

RRTags berisi data identifikasi pesan, di mana RRPID berfungsi sebagai identitas unik pasangan pesan (*message pair*).

```
RRTags ::= SEQUENCE {
   rrpid       RRPID,
   merTermIDs  MerTermIDs,
   currentDate Date
}

RRPID ::= OCTET STRING(SIZE(20)) -- Request response pair identification

MerTermIDs ::= SEQUENCE {
   merchantID  MerchantID,
   terminalID   VisibleString (SIZE(1..ub-terminalID)) OPTIONAL,
   agentNum    INTEGER (0..MAX)  OPTIONAL,
   chainNum    [0] INTEGER (0..MAX)  OPTIONAL,
   storeNum    [1] INTEGER (0..MAX)  OPTIONAL
}

Date ::= GeneralizedTime

MerchantID ::= SETString { ub-MerchantID }
```

Gambar III-28 RRTags

RRTags memiliki beberapa *field* yang akan dijelaskan dalam tabel berikut ini.

Nama *field*	Keterangan
RRPID	*Request Response Pair Identification*, Identitas unik pasangan pesan, misalnya *Authorization Request* dan *Authorization Response* memeliki sebuah RRPID yang unik.
MerTermIDs	Informasi mengenai identitas terminal *merchant*
CurrentDate	Tanggal saat RRTags dibuat.

Tabel III-8 Keterangan RRTags

MerTermIDs berisi informasi mengenai terminal *merchant*, salah satunya adalah MerchantID yang didapat dari sertifikat tanda tangan *merchant.*

3. InstallRecurData

```
InstallRecurData ::= SEQUENCE {
   installRecurInd  InstallRecurInd,
   irExtensions     [0] MsgExtensions {{IRExtensionsIOS}} OPTIONAL
}

InstallRecurInd ::= CHOICE {
   installTotalTrans  [0] INTEGER (2..MAX),
   recurring          [1] Recurring
}

Recurring ::= SEQUENCE {
   recurringFrequency  INTEGER (1..ub-recurringFrequency),
   recurringExpiry     Date
}
```

Gambar III-29 InstallRecurData

InstallRecurData digunakan apabila *cardholder* ingin melakukan pembelian barang secara mengangsur atau berlangganan.

Ada dua cara mendefinisikan InstallRecurData (InstallRecurInd), yaitu:

Nama *field*	Keterangan
1. InstallTotalTrans	*Cardholder* memberikan jumlah maksimum otorisasi yang diperbolehkan untuk pembayaran angsuran atau berlangganan.
2. Recurring	{RecurringFrequency, RecurringExpiry}
• RecurringFrequency	Jumlah hari antara dua otorisasi yang berurutan (28 untuk otorisasi bulanan).
• RecurringExpiry	Tanggal akhir otorisasi diperbolehkan.

Tabel III-9 Keterangan InstallRecurData

4. PANData

```
PANData ::= SEQUENCE {
    pan         PAN,
    cardExpiry  CardExpiry,
    panSecret   Secret,
    exNonce     Nonce
 }
```

Gambar III-30 PANData

PANData berisi informasi yang mengidentifikasikan rekening kartu pembayaran tertentu. PANData sering dipakai sebagai parameter ekstra yang diletakkan pada blok OAEP pada EX, EXH, EncX dan EncBX.

PANData memiliki beberapa *field* yang masing-masing memiliki informasi tertentu terhadap kartu pembayaran tersebut yang akan diterangkan pada tabel berikut.

Nama *field*	Keterangan
PAN	*Primary Account Number*, biasanya nomor rekening pada

	kartu.
CardExpiry	Tanggal kadaluarsa kartu pembayaran.
PANSecret	Nilai rahasia yang dipegang bersama antara *cardholder*, *payment gateway*, dan CA. Bertujuan untuk mencegah *guessing attack* pada PAN pada sertifikat *cardholder*.
ExNonce	*Nonce*, bertujuan untuk mencegah *dictionary attack*[5] pada PANData.

Tabel III-10 Keterangan PANData

5. PANToken

```
PANToken ::= SEQUENCE {
    pan         PAN,
    cardExpiry  CardExpiry,
    exNonce     Nonce
}
```

Gambar III-31 PANToken

PANToken berfungsi sama dengan PANData, namun ia digunakan saat PANSecret tidak diperlukan.

6. PaymentInstructions (PI)

```
PI ::= CHOICE {
    piUnsigned    [0] EXPLICIT PIUnsigned,
    piDualSigned  [1] EXPLICIT PIDualSigned,
    authToken     [2] EXPLICIT AuthToken
}
```

Gambar III-32 Payment Instruction (PI)

PI (*Payment Instruction*) adalah data yang paling penting dan sensitif dalam SET. PI digunakan untuk mengirimkan data yang diperlukan untuk otorisasi kartu pembayaran dari *cardholder* kepada *payment gateway* melalui perantaraan *merchant* tanpa bisa dibaca oleh *merchant.*

PI terdiri dari 3 macam yaitu PIUnsigned, PIDualSigned, dan AuthToken dengan kegunaan yang berbeda seperti kita lihat pada tabel berikut ini.

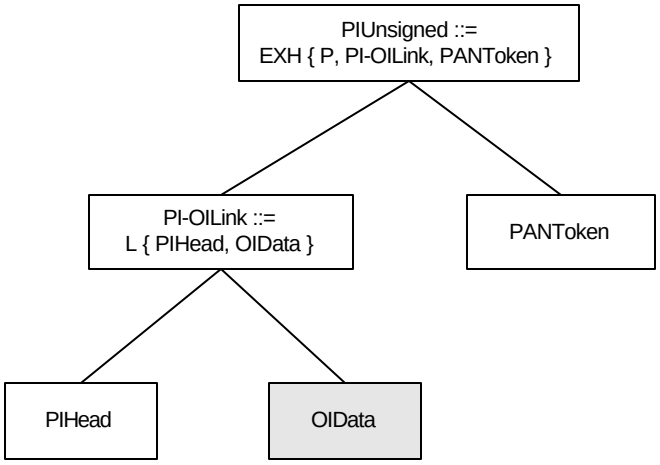
PIUnsigned	Dibuat oleh *cardholder* yang tidak memiliki sertifikat tanda tangan. Digunakan dalam pesan PReqUnsigned. Integritas data dijamin dengan menaruh *hash* dari PIData di dalam blok OAEP yang terenkripsi.

[5] Si pembobol pesan dalam *dictionary attack* memiliki banyak contoh-contoh data yang akan dibobol, dalam hal ini ia banyak mempunyai contoh PANData yang dapat dicobakan satu persatu.

	Tidak ada jaminan autentisitas pada mekanisme ini.
PIDualSigned	Dibuat oleh *cardholder* yang memiliki sertifikat tanda tangan. Autentisitas dan Integritas data terjamin oleh tanda tangan *cardholder*.
AuthToken	Dibuat oleh *payment gateway* dan hanya bisa dibaca oleh *payment gateway*. AuthToken bertujuan untuk otorisasi berulang-ulang.

Tabel III-11 Tiga Macam Payment Instruction (PI)

PIUnsigned merupakan EXH {P, PI-OILink, PANToken} seperti yang digambarkan pada *tree* berikut ini.

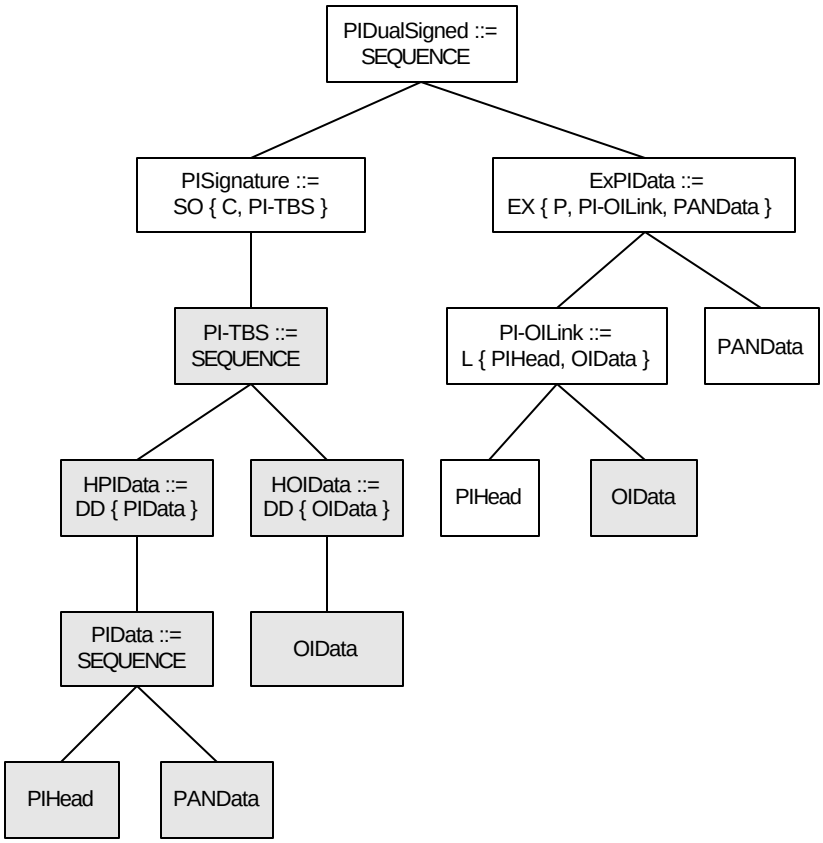


Gambar III-33 Tree PIUnsigned

Kalau kita tinjau kembali EXH pada pembahasan sebelumnya, *hash* dari PI-OILink dan PANToken akan tersimpan dalam blok OAEP yang terenkripsi. *Hash* dari PI-OILink ini yang dikatakan menjamin integritas data.

Kalau kita juga masih ingat OI (*Order Information*) pada bab dua, ia sebenarnya adalah OIData, yang di dalam PI jenis apapun tersimpan dalam bentuk *hash*-nya. OIData digambarkan berwarna abu-abu karena ia dalam bentuk ter-*hash* sehingga tidak dapat dibaca isinya. Pembahasan tentang PIHead dan OIData menyusul setelah PIDualSigned.

PIDualSigned merupakan SEQUENCE dari PISignature dan ExPIData, seperti yang terlihat pada gambar.

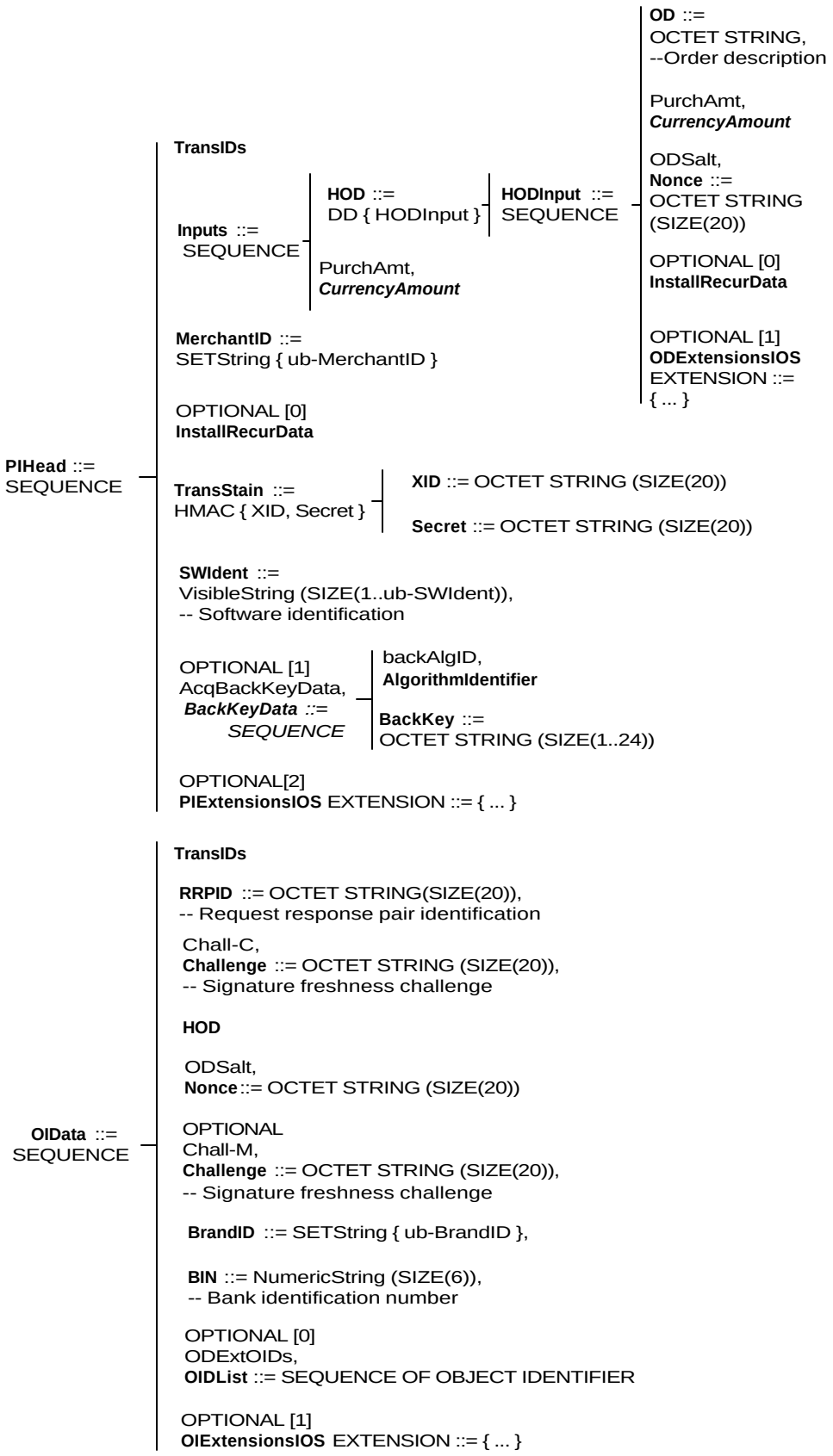


Gambar III-34 Tree PIDualSigned

Bentuk PIDualSigned dan PIUnsigned agak mirip. Kalau kita perhatikan, ExPIData sama dengan PIUnsigned, bedanya *hash* dari PI-OILink tidak disertakan pada blok OAEP dan yang menjadi ekstra parameter adalah PANData bukan PANToken. Mengapa PANData bukan PANToken, karena pada PIUnsigned *cardholder* tidak memiliki hubungan dengan CA yang bersama-sama menyimpan PANSecret yang ada pada PANData.

Dapat kita perhatikan juga, PISignature adalah tanda tangan digital ganda yang menghubungkan antara PI (PIData) dan OI(OIData). Mulai PI-TBS ke bawah semua berwarna abu-abu, karena semuanya dalam bentuk ter-*hash*, tidak dapat dibaca.

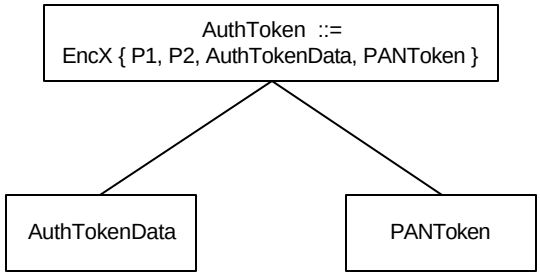
PIUnsigned dan PIDualSigned memakai struktur data umum OIData dan PIHead, keduanya dapat kita lihat dalam representasi *tree* berikut ini.



Gambar III-35 Tree ASN.1 PIHead dan OIData

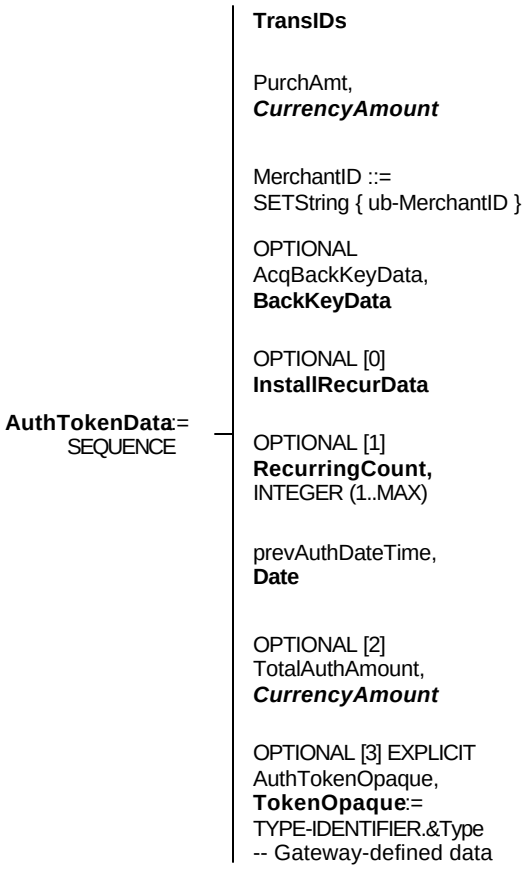
Dari gambar *tree* di atas, kita juga dapat melihat hubungan PI(PIHead) dan OI(OIData) bahwa keduanya memiliki TransIDs dan HOD yang sama.

AuthToken (Authorization Token) merupakan EncX{P1, P2, AuthTokenData, PANToken}.



Gambar III-36 Tree AuthToken

AuthToken dipergunakan untuk otorisasi yang berulang-ulang, untuk angsuran, langganan, pengiriman barang yang terpisah, dsb. Tujuannya agar pada otorisasi yang berulang-ulang *merchant* tidak perlu mengirimkan PI dalam bentuk yang besar (PIUnsigned dan PIDualSigned) untuk transaksi yang telah terotorisasi sebelumnya.



Gambar III-37 Tree ASN.1 AuthTokenData

7. AcqCardMsg

```
AcqCardMsg ::= EncK { AcqBackKey, P, AcqCardCodeMsg }

AcqBackKey ::= BackKeyData
BackKey ::= OCTET STRING (SIZE(1..24))                    -- Secret

AcqCardCodeMsg ::= SEQUENCE {
   acqCardCode     AcqCardCode,
   acqCardMsgData  AcqCardMsgData
}

AcqCardCode ::= ENUMERATED {
   messageOfDay        (0),
   accountInfo         (1),
   callCustomerService (2)
}

AcqCardMsgData ::= SEQUENCE {
   acqCardText   [0] EXPLICIT SETString { ub-acqCardText }  OPTIONAL,
   acqCardURL    [1] URL  OPTIONAL,
   acqCardPhone  [2] EXPLICIT SETString { ub-acqCardPhone }  OPTIONAL
}
```

Gambar III-38 AcqCardMsg

AcqCardMsg berguna dalam mekanisme di mana *acquirer* dapat mengirimkan pesan kembali ke *cardholder* melalui *merchant* tanpa diketahui oleh *merchant. Cardholder* terlebih dahulu mengirimkan kunci simetris (lihat AcqBackKeyData pada PIHead), kemudian *acquirer* memberikan balasannya melalui AuthRes (*Authorization Response*) dan *merchant* meneruskannya pada *cardholder* melalui PRes (*Purchase Response*).

8. CapToken

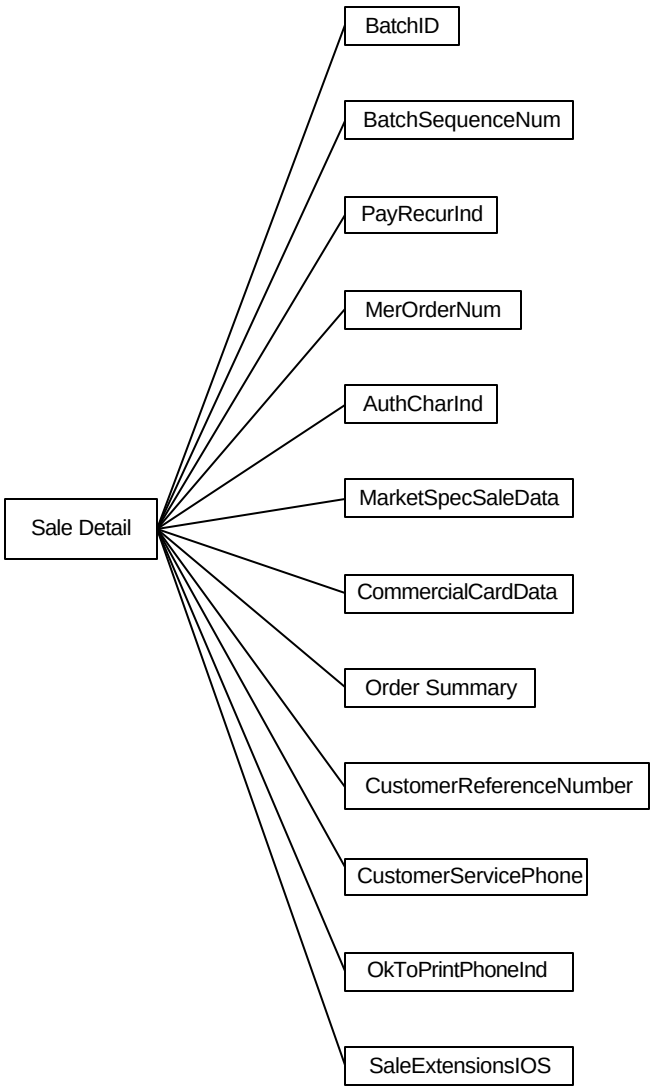
```
CapToken ::= CHOICE {
   encX  [0] EXPLICIT EncX { P1, P2, CapTokenData, PANToken },
   enc   [1] EXPLICIT Enc { P1, P2, CapTokenData },
   null  [2] EXPLICIT NULL
}

CapTokenData ::= SEQUENCE {
   authRRPID     RRPID,
   authAmt       CurrencyAmount,
   tokenOpaque   TokenOpaque
}
```

Gambar III-39 CapToken

CapToken (*Capture Token*) merepresentasikan data yang dibutuhkan oleh *payment gateway* untuk melakukan proses *capture* untuk suatu transaksi. CapToken dibuat untuk otorisasi yang tidak langsung meminta proses *capture* (*Capture Now*).

9. SaleDetail

SaleDetail bertujuan mengumpulkan semua data yang berhubungan dengan transaksi pembelian yang menggunakan suatu kartu pembayaran. Data-data yang ada dalam SaleDetail dikelompokkan seperti yang terlihat pada gambar berikut ini.



Gambar III-40 SaleDetail

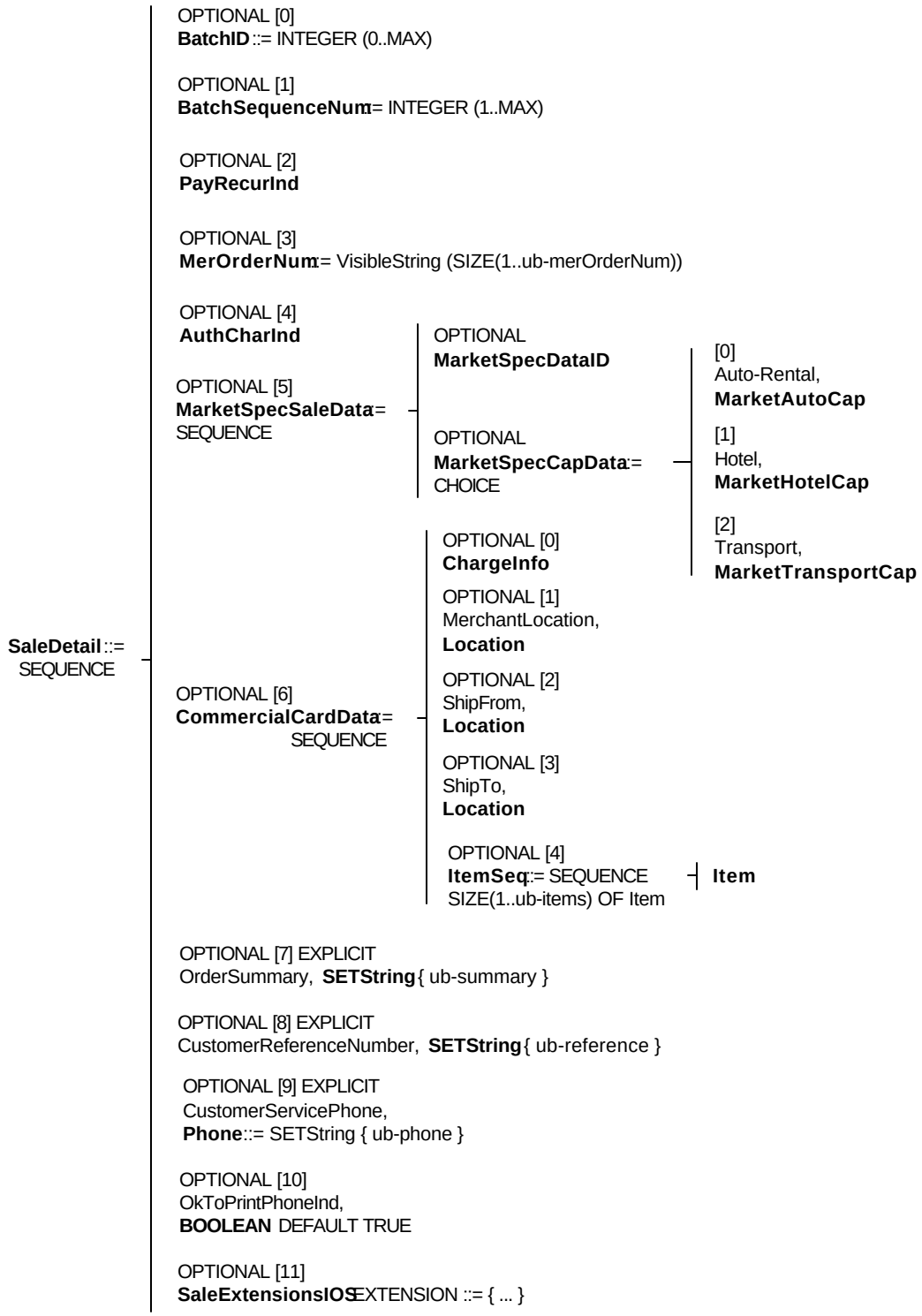
Semua kelompok data tersebut bersifat *optional.* Keterangan mengenai kelompok-kelompok data tersebut dapat kita lihat pada tabel berikut ini.

Data	Keterangan
BatchID	Identifikasi *batch,* kesepakatan administrasi antara *merchant* dan *payment gateway*.

<table>
<tr><td>BatchSequenceNum</td><td>Nomor urut transaksi ini dalam batch.</td></tr>
<tr><td>PayRecurInd</td><td>merupakan tipe-tipe transaksi, seperti tabel dibawah ini.
<table>
<tr><th>Tipe transaksi</th><th>Keterangan</th></tr>
<tr><td>unknown</td><td>tipe transaksi tidak diketahui</td></tr>
<tr><td>single transaction</td><td>transaksi hanya ter-diri dari sebuah otorisasi dan capture</td></tr>
<tr><td>recurring transaction</td><td>transaksi terdiri dari beberapa otorisasi dan capture dengan fre-kuensi yang teratur</td></tr>
<tr><td>installment payment</td><td>transaksi terdiri dari beberapa otorisasi dan capture dengan jumlah yang terbatas</td></tr>
<tr><td>other mail order</td><td>transaksi mail order yang lain.</td></tr>
</table></td></tr>
<tr><td>MerOrderNum</td><td>Nomor pesanan merchant</td></tr>
<tr><td>AuthCharInd</td><td>Nilai yang menyatakan kondisi ketika terjadi proses otorisasi (direct marketing, recurring payment, address verification, preferred customer, incremental authorization).</td></tr>
<tr><td>MarketSpecSale Data</td><td>Data yang menerangkan secara spesifik tentang pasar (merchant).</td></tr>
<tr><td>CommercialCard Data</td><td>Deskripsi barang-barang untuk proses capture. Biasanya informasi ini hanya diikutsertakan untuk produk kartu komersil di bawah pengaturan khusus antara merchant dan cardholder.</td></tr>
<tr><td>OrderSummary</td><td>Ringkasan deskripsi pesanan</td></tr>
<tr><td>CustomerReference Number</td><td>Nomor tertentu yang diberikan pada pesanan cardholder.</td></tr>
<tr><td>CustomerService Phone</td><td>Nomor telepon customer service milik merchant.</td></tr>
<tr><td>OkToPrintPhoneInd</td><td>Nilai boolean yang menyatakan apakah suatu issuer boleh mencetak nomor telepon customer service sesuai dengan pernyataan cardholder.</td></tr>
<tr><td>SaleExtensionsIOs</td><td>Data finansial tertentu yang berguna pada proses capture oleh payment gateway, jaringan finansial, atau oleh issuer.</td></tr>
</table>

Tabel III-12 Keterangan SaleDetail

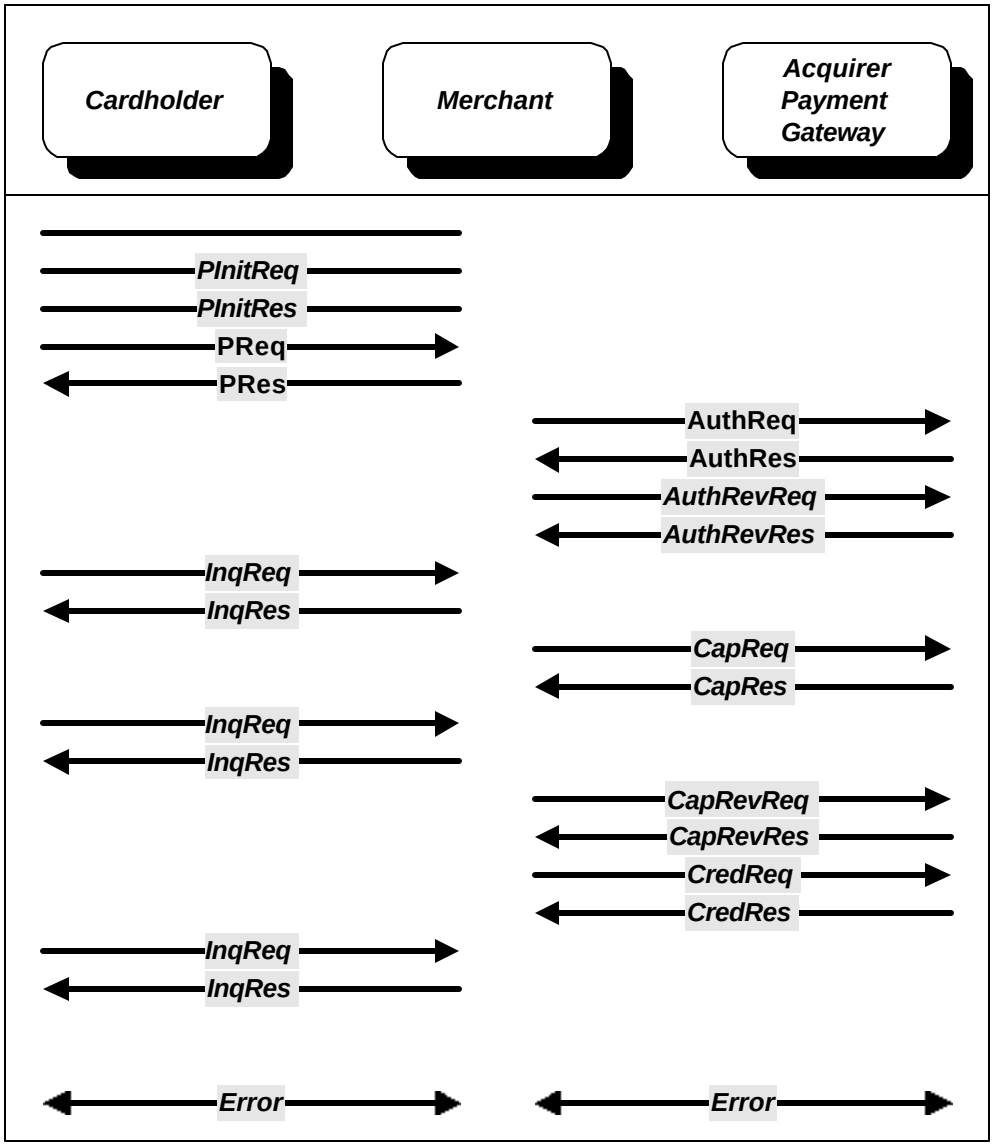
Spesifikasi SaleDetail dalam ASN.1 secara garis besar digambarkan pada tree berikut ini, untuk lengkapnya silakan mengacu pada Spesifikasi Formal Protokol (SET buku 3).



Gambar III-41 Tree ASN.1 SaleDetail

1.10.6 Pasangan Pesan Otorisasi (*Authorization Pair)*

Pada tahap ini kita telah memasuki lapisan pesan (*message*). Ada banyak jenis pesan dalam SET, sebagian sudah kita jumpai pada bab dua. Dalam gambar, pesan-pesan yang tercetak miring adalah pesan yang bersifat *optional.*

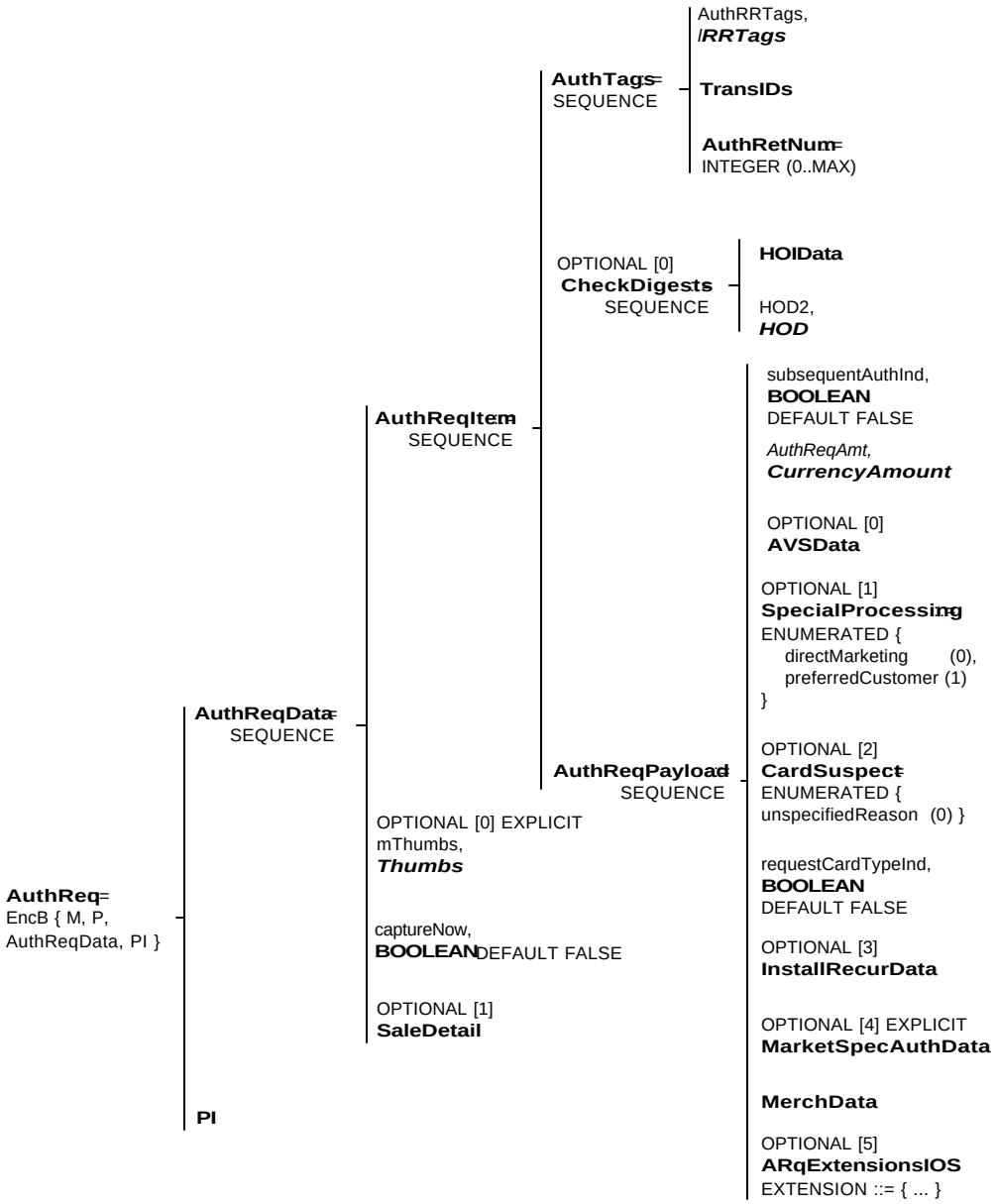


Gambar III-42 Pesan-Pesan dalam SET

Dalam ruang lingkup penelitian ini yaitu dalam proses otorisasi, hanya mengenal dua buah pesan yaitu *Authorization Request* (AuthReq) dan *Authorization Response* (AuthRes). Pesan-pesan ini dikirimkan ketika *merchant* meminta otorisasi atas kartu pembayaran milik *cardholder* (lihat bab dua).

1. Authorization Request

Authorization request adalah pesan yang dikirimkan oleh *merchant* kepada *payment gateway* yang berisi informasi-informasi transaksi yang akan diotorisasi saja atau sekaligus akan di-*capture*. Berikut ini adalah spesifikasi ASN.1 yang ditampilkan dalam bentuk *tree*.



Gambar III-43 Authorization Request (AuthReq)

Penjelasan tentang komponen-komponen penyusun *authorization request* ada dalam tabel berikut ini.

AuthReq	EncB(M,P, AuthReqData, PI)
AuthReqData	{AuthReqItem, [Mthumbs], CaptureNow, [SaleDetail]}
PI	<PIUnsigned, PIDualSigned, AuthToken>
AuthReqItem	{AuthTags,[CheckDigests],AuthReqPayload}
MThumbs	Kumpulan *digest* dari sertifikat-sertifikat dan daftar-daftar sertifikat yang tidak berlaku.
CaptureNow	Nilai *boolean* yang menandakan apakah *merchant* ingin langsung melakukan *capture* dalam proses ini.
SaleDetail	Data-data yang berhubungan dengan transaksi dan *merchant.*
AuthTags	{AuthRRTags, TransIDs, [AuthRetNum]}
CheckDigests	{HOIData, HOD2} digunakan oleh *payment gateway* untuk mengautentikasi PI, diabaikan apabila PI adalah AuthToken.
AuthReqPayload	Dijelaskan dalam tabel tersendiri.
AuthRRTags	RRTags, lihat pada penjelasan komponen pesan. Cat: RRPID diperlukan di sini karena bisa terdapat beberapa pasangan otorisasi untuk setiap *purchase request*.
TransIDs	Didapat dari OIData yang terasosiasi dengan transaksi ini.
AuthRetNum	Identifikasi dari *authorization request* yang sudah masuk dalam jaringan finansial/perbankan.
HOIData	DD(OIData) OIData diperoleh *merchant* dalam *purchase request*. *Payment gateway* akan membandingkan HOIData buatan *merchant* ini dengan HOIData buatan *cardholder* yang terdapat dalam PI.
HOD2	DD(HODInput) HODInput diperoleh *merchant* pada fase belanja, sebelum bermain dalam protokol SET. *Payment gateway* akan membandingkan HOD2 ini dengan HOD yang terdapat dalam PI.

Tabel III-13 Keterangan AuthReq

AuthReqPayload	{SubsequentAuthInd, AuthReqAmt, [AVSData], [SpecialProcessing], [CardSuspect], RequestCardTypeInd, [InstallRecurData], [MarketSpecAuthData], MerchData, [ArqExtensions]}
SubsequentAuthInd	Nilai *boolean* yang menandakan *merchant* meminta otorisasi tambahan karena terjadi pengiriman terpisah.

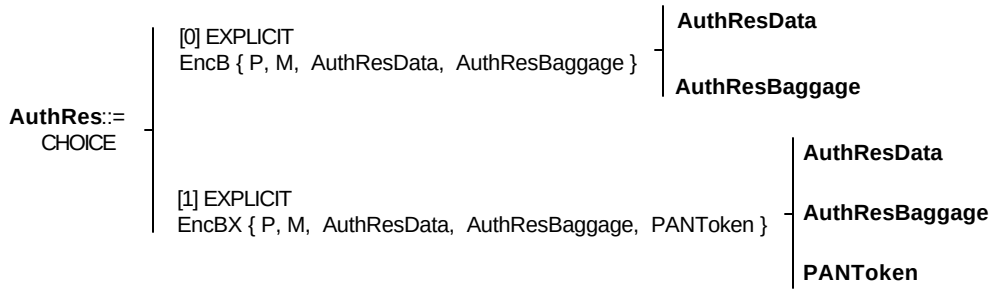
AuthReqAmt	Nilai otorisasi, boleh berbeda dari nilai pembelian (PurchAmt) tergantung kebijakan *acquirer*.
AVSData	{[StreetAddress], Location} Alamat penagihan *cardholder*, didapat melalui mekanisme di luar SET.
SpecialProcessing	*Field* terenumerasi yang menandakan tipe dari pemrosesan khusus yang diminta.
CardSuspect	*Field* terenumerasi yang menandakan *merchant* mencurigai *cardholder* dan penyebab dari kecurigaan tersebut.
RequestCardTypeInd	Menandakan bahwa *merchant* meminta keterangan tipe dari kartu milik *cardholder*.
InstallRecurData	Data-data yang diperlukan apabila terjadi pembayaran secara angsuran atau berlangganan.
MarketSpecAuthData	<MarketAuthoAuth, MarketHotelAuth, MarketTransportAuth> Ketentuan otorisasi pasar.
MerchData	{[MerchCatCode], [MerchGroup]}
ARqExtensions	Data finansial yang berhubungan dengan proses otorisasi ini.
StreetAddress	Nama jalan alamat *cardholder*
Location	Alamat lengkap *cardholder*
MarketAutoAuth	{Duration}
MarketHotelAuth	{Duration, [Prestige]}
MarketTransportAuth	{} Belum ada ketentuan otorisasi untuk segmen pasar ini.
MerchCatCode	Kode 4 *byte* (ANSI X9.10) yang menandakan tipe bisnis, produk dan jasa.
MerchGroup	Kode terenumerasi yang menandakan kategori umum dari *merchant*.
Duration	Batas waktu antara otorisasi dan *capture* (dalam hari).
Prestige	Nilai terenumerasi yang menandakan tingkat kemewahan *merchant*. Tingkatan ini didefinisikan oleh pemegang merek kartu.

Tabel III-14 Keterangan AuthReqPayload

2. Authorization Response

Authorization response merupakan pesan balasan atas pesan *authorization request* yang dikirimkan oleh *merchant* pada awal proses otorisasi. Pesan ini juga merupakan jawaban atas permintaan otorisasi yang dilakukan oleh *acquirer* kepada *issuer* lewat jaringan perbankan yang sudah ada di luar lingkungan SET.

Terdapat dua macam pesan *authorization response*, yaitu *authorization response* yang menyertakan PANToken milik *payment gateway* dan *authorization response* yang tidak menyertakan PANToken, tergantung regulasi dan kebiasaan.



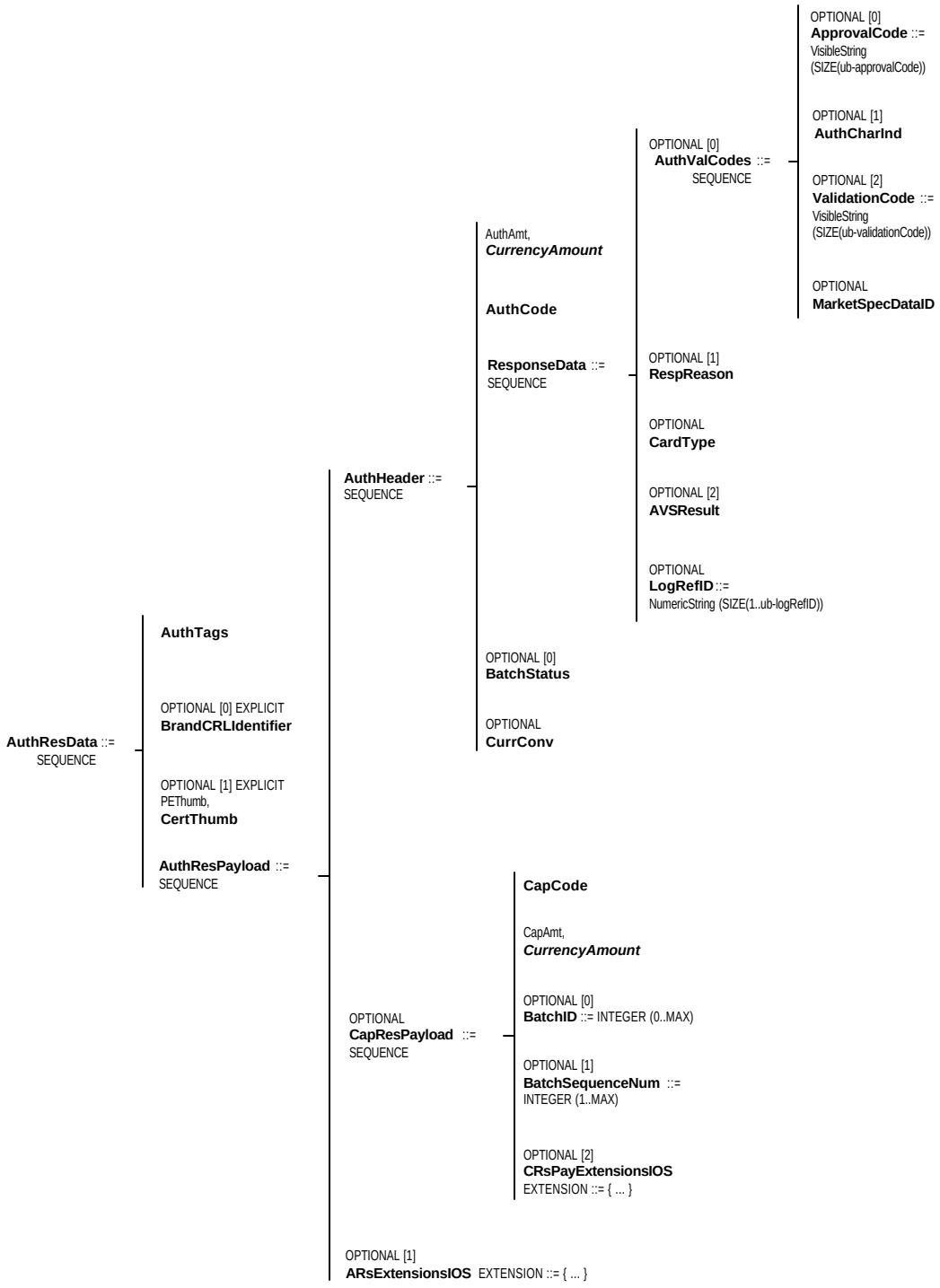
Gambar III-44 Authorization Response (AuthRes)

Tabel-tabel berikut ini berisi keterangan tentang komponen-komponen penyusun *authorization response* (AuthRes).

AuthRes	< EncB(P, M, AuthResData, AuthResBaggage), EncBX(P, M, AuthResData, AuthResBaggage, PANToken) >
AuthResData	Diterangkan dalam tabel keterangan AuthResData
AuthResBaggage	Diterangkan dalam tabel keterangan AuthResBaggage

Tabel III-15 Keterangan AuthRes

AuthResData dan komponen-komponen penyusunnya digambarkan dalam *tree* ASN.1 berikut ini. Tabel keterangan mengenai komponen penyusunnya menyusul dibawahnya.



Gambar III-45 AuthResData

AuthResData	{AuthTags, [BrandCRLIdentifier], [PEThumb], AuthResPayload}
AuthTags	Disalin dari AuthReq.TransIDs yang bersangkutan, dan AuthRetNum di dalamnya disesuaikan dengan informasi saat ini.
BrandCRLIdentifier	Daftar CRL (daftar sertifikat yang tidak berlaku) untuk setiap CA dibawah Brand CA.
PEThumb	Kumpulan *digest* dari sertifikat-sertifikat *payment gateway,* diberikan apabila diketahui *merchant* membutuhkannya.
AuthResPayload	{AuthHeader, [CapResPayload], [ArsExtensions]
AuthHeader	Diterangkan dalam tabel keterangan AuthHeader.
CapResPayload	Diterangkan dalam tabel keterangan CapResPayload. Data ini dibuat jika *merchant* meminta *capture* bersamaan dengan proses otorisasi ini.
ArsExtensions	Data finansial yang berhubungan dengan proses otorisasi atau *capture* yang berguna bagi *payment gateway,* jaringan perbankan, atau *issuer*.

Tabel III-16 Keterangan AuthResData

AuthHeader	{AuthAmt, AuthCode, ResponseData, [BatchStatus], [CurrConv]}
AuthAmt	Disalin dari AuthReqPayload.AuthReqAmt.
AuthCode	Kode terenumerasi yang menyatakan hasil dari proses otorisasi pembayaran.
ResponseData	{[AuthValCodes], [RespReason], [CardType], [AVSResult], [LogRefID]}
BatchStatus	Informasi status mengenai transaksi-transaksi yang telah diotorisasi. Bagian ini tidak masuk dalam ruang lingkup penelitian ini.
CurrConv	{CurrConvRate, CardCurr}
AuthValCodes	{[ApprovalCode], [AuthCharInd], [ValidationCode]. [MarketSpecDataID]}
RespReason	Kode Terenumerasi yang menandakan entitas yang memberi pelayanan dan (bila sesuai) alasan penundaaannya.
CardType	Kode terenumerasi yang menandakan tipe kartu yang digunakan untuk transaksi.
AVSResult	Kode terenumerasi yang merupakan jawaban atas verifikasi alamat *cardholder.*

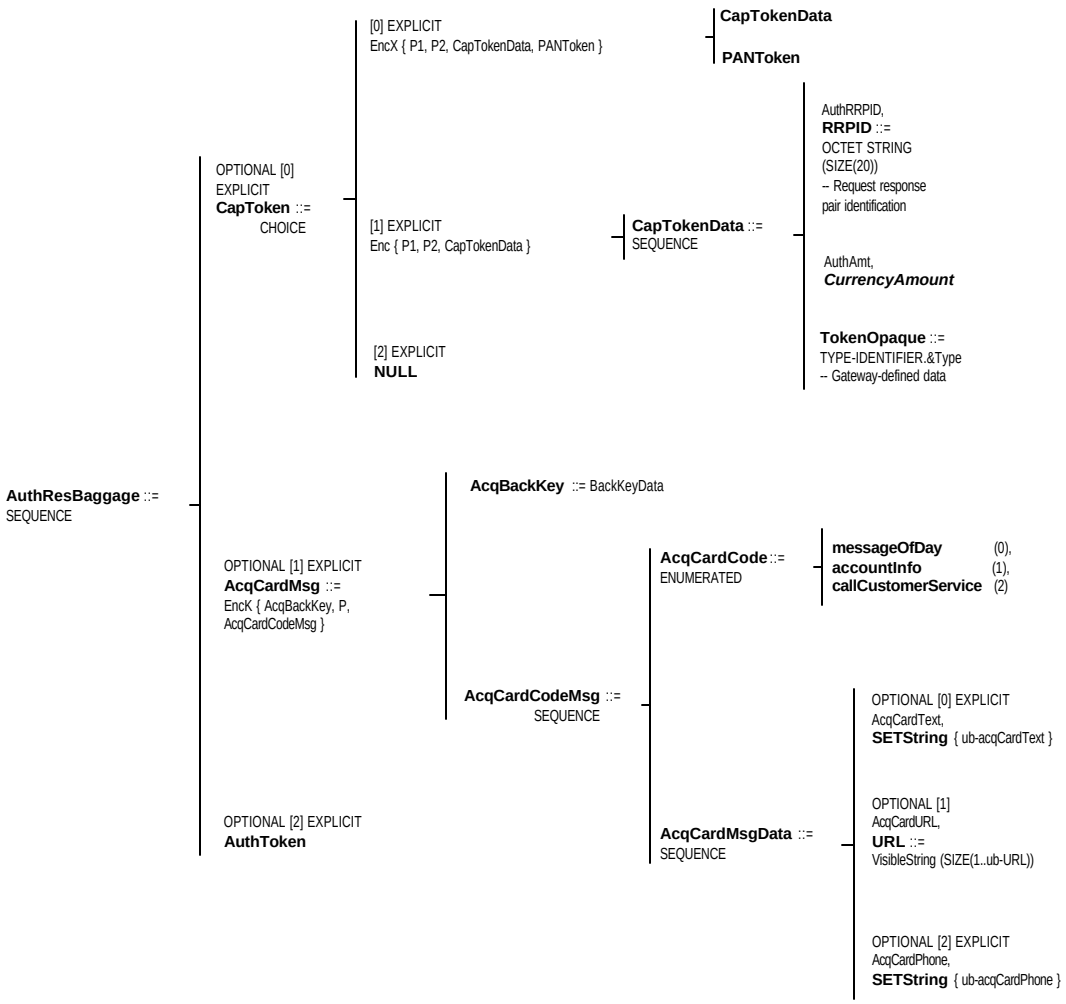
LogRefID	Nilai *alphanumeric* yang diberikan pada suatu transaksi (digunakan untuk pencocokan pada pengembalian transaksi atau *reversal*).
CurrConvRate	Nilai konversi mata uang.
CardCurr	Kode mata uang *cardholder* yang didefinisikan dalam ISO 4127.
ApprovalCode	Kode persetujuan yang diberikan *issuer* pada transaksi.
AuthCharInd	Nilai terenumerasi yang menandakan kondisi yang terjadi ketika otorisasi terjadi.
ValidationCode	4 *byte* kode *alphanumeric* yang dikalkulasi untuk memastikan *field-field* yang dibutuhkan dalam pesan otorisasi juga ada dalam pesan untuk proses kliringnya.
MarketSpecDataID	Kode terenumerasi yang mengidentifikasikan tipe dari data pasar tertentu yang diberikan pada waktu otorisasi (sesuai yang didefinisikan oleh jaringan finansial).

Tabel III-17 Keterangan AuthHeader

CapResPayload	{CapCode, CapAmt, [BatchID], [BatchSequenceNum], [CrsPayExtensions]}
CapCode	Kode terenumerasi yang menandakan status dari *capture.*
CapAmt	Harga *capture*, biasanya disalin dari CapReq yang bersangkutan.
BatchID	Identifikasi dari persetujuan *batch* untuk administrasi *merchant-acquirer*
BatchSequenceNum	Nomor urut transaksi ini dalam *batch*.
CrSPayExtensions	Data finansial yang berhubungan dengan proses *capture*.

Tabel III-18 Keterangan CapResPayload

Berikut ini adalah *tree* ASN.1 dari AuthResBaggage yang merupakan bagasi yang mengikuti *authorization response*. AuthResBaggage berisi data-data yang berguna untuk diberikan kepada *cardholder* (AcqCardMsg), untuk otorisasi transaksi ini berikutnya (AuthToken) dan untuk melakukan proses *capture* transaksi ini (CapToken).



Gambar III-46 AuthResBaggage

AuthResBaggage	{[CapToken], [AcqCardMsg], [AuthToken]}
CapToken	Lihat pada pembahasan sebelumnya di bab ini
AcqCardMsg	Lihat pada pembahasan sebelumnya di bab ini
AuthToken	Lihat pada pembahasan sebelumnya di bab ini

Tabel III-19 Keterangan AuthResBaggage

1.10.7 Pembungkus Pesan (*Message Wrapper*)

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

MessageHeader ::= SEQUENCE {
    version     INTEGER { setVer1(1) } (setVer1),
    revision    INTEGER (0) DEFAULT 0,   -- This is version 1.0
    date        Date,
    messageIDs  [0] MessageIDs  OPTIONAL,
    rrpid       [1] RRPID  OPTIONAL,
    swIdent     SWIdent
}

MessageIDs ::= SEQUENCE {
    lid-C  [0] LocalID  OPTIONAL,
    lid-M  [1] LocalID  OPTIONAL,
    xID    [2] XID  OPTIONAL
}

SWIdent ::= VisibleString (SIZE(1..ub-SWIdent))    -- Software identification

MWExtensionsIOS EXTENSION ::= { ... }
```

Gambar III-47 Pembungkus Pesan (*Message Wrapper*)

MessageWrapper berfungsi untuk membungkus pesan-pesan SET sebelum dikirimkan. MessageWrapper terdiri dari tiga bagian yaitu MessageHeader, Message dan MWExtensions.

Beberapa identitas pesan disimpan dalam MessageHeader yaitu dalam MessageID dan RRPID. MessageID berisi beberapa komponen dari TransIDs yaitu LID-C, LID-M dan XID. Adapun RRPID merupakan identifikasi untuk setiap pasangan pesan, seperti *purchase request* dan *purchase response* memiliki satu RRPID yang unik.

Pesan-pesan SET seperti *purchase request, purchase response, authorization request, authorization response, capture request, capture response*, dsb; tersimpan dalam komponen Message. Message merupakan CHOICE dari seluruh tipe pesan SET. Masing-masing tipe pesan memiliki *tag* EXPLICIT seperti yang didefinisikan dalam ASN.1 di bawah ini.

```
Message ::= CHOICE {

    purchaseInitRequest          [ 0] EXPLICIT PInitReq,
    purchaseInitResponse         [ 1] EXPLICIT PInitRes,

    purchaseRequest              [ 2] EXPLICIT PReq,
    purchaseResponse             [ 3] EXPLICIT PRes,

    inquiryRequest               [ 4] EXPLICIT InqReq,
    inquiryResponse              [ 5] EXPLICIT InqRes,

    authorizationRequest         [ 6] EXPLICIT AuthReq,
    authorizationResponse        [ 7] EXPLICIT AuthRes,

    authReversalRequest          [ 8] EXPLICIT AuthRevReq,
    authReversalResponse         [ 9] EXPLICIT AuthRevRes,

    captureRequest               [10] EXPLICIT CapReq,
    captureResponse              [11] EXPLICIT CapRes,

    captureReversalRequest       [12] EXPLICIT CapRevReq,
    captureReversalResponse      [13] EXPLICIT CapRevRes,

    creditRequest                [14] EXPLICIT CredReq,
    creditResponse               [15] EXPLICIT CredRes,

   creditReversalRequest         [16] EXPLICIT CredRevReq,
   creditReversalResponse        [17] EXPLICIT CredRevRes,

   pCertificateRequest           [18] EXPLICIT PCertReq,
   pCertificateResponse          [19] EXPLICIT PCertRes,

   batchAdministrationRequest    [20] EXPLICIT BatchAdminReq,
   batchAdministrationResponse   [21] EXPLICIT BatchAdminRes,

   cardholderCInitRequest        [22] EXPLICIT CardCInitReq,
   cardholderCInitResponse       [23] EXPLICIT CardCInitRes,

   meAqCInitRequest              [24] EXPLICIT Me-AqCInitReq,
   meAqCInitResponse             [25] EXPLICIT Me-AqCInitRes,

   registrationFormRequest       [26] EXPLICIT RegFormReq,
   registrationFormResponse      [27] EXPLICIT RegFormRes,

   certificateRequest            [28] EXPLICIT CertReq,
   certificateResponse           [29] EXPLICIT CertRes,

   certificateInquiryRequest     [30] EXPLICIT CertInqReq,
   certificateInquiryResponse    [31] EXPLICIT CertInqRes,

   error                    [999] EXPLICIT Error
}
```

Gambar III-48 Pesan (*Message*)

Tag EXPLICIT pada tipe Message ini berarti bahwa tipe ini menambahkan nilai *tag* dan nilai panjang di depan kode DER dari setiap pesan. Untuk setiap pesan nilai *tag*-nya berbeda-beda sehingga kalau kita menerima suatu objek Message dalam bentuk kode DER, kita dapat membedakan jenis pesan direpresentasikan oleh objek Message ini.

1.10.8 Pesan Kesalahan (*Error*)

```
Error ::= CHOICE {
    SignedError    [0] EXPLICIT SignedError,
    UnsignedError  [1] EXPLICIT ErrorTBS
}

SignedError ::= S {EE, ErrorTBS}

ErrorTBS ::= SEQUENCE {
    ErrorCode   ErrorCode,
    ErrorNonce  Nonce,
    ErrorOID    [0] OBJECT IDENTIFIER  OPTIONAL,
    ErrorThumb  [1] EXPLICIT CertThumb  OPTIONAL,
    ErrorMsg    [2] EXPLICIT ErrorMsg
}

ErrorMsg ::= CHOICE {                          -- Either the
    MessageHeader  [0] EXPLICIT MessageHeader,      -- MessageHeader or a
    BadWrapper     [1] OCTET STRING (SIZE(1..20000)) -- copy of the message
}
```

Gambar III-49 Pesan Kesalahan (Error)

Pesan kesalahan (Error) adalah pesan yang setara dengan pesan-pesan SET lain, namun berfungsi untuk memberitahu suatu kesalahan. Error ini pada waktu akan dikirimkan juga dibungkus dalam objek MessageWrapper.

Error terdiri dari dua macam tipe yaitu SignedError dan UnsignedError, perbedaannya hanya pada tanda tangan digital. Kedua Error tersebut dalam spesifikasi ASN.1 di atas berisi ErrorTBS.

Dalam ErrorTBS terdapat beberapa komponen, diantaranya adalah ErrorCode yaitu suatu kode terenumerasi yang menjelaskan keadaan kesalahan tersebut; dan ErrorMsg yaitu pesan yang mengalami kesalahan.

ErrorMsg bisa dikirimkan dalam dua bentuk, yaitu MessageHeader dari pesan yang mengalami kesalahan atau MessageWrapper dari pesan tersebut. Biasanya jika yang dikirimkan adalah objek MessageWrapper-nya , kesalahan terjadi karena tidak bisa membuka MessageWrapper tersebut.

Di bawah ini adalah spesifikasi ASN.1 untuk kode kesalahan atau ErrorCode.

```
ErrorCode ::= ENUMERATED {
    UnspecifiedFailure      (1),
    MessageNotSupported     (2),
    DecodingFailure         (3),
    InvalidCertificate      (4),
    ExpiredCertificate      (5),
    RevokedCertificate      (6),
    MissingCertificate      (7),
    SignatureFailure        (8),
    BadMessageHeader        (9),
    WrapperMsgMismatch      (10),
    VersionTooOld           (11),
    VersionTooNew           (12),
    UnrecognizedExtension   (13),
    MessageTooBig           (14),
    SignatureRequired       (15),
```

```
    MessageTooOld        (16),
    MessageTooNew        (17),
    ThumbsMismatch       (18),
    UnknownRRPID         (19),
    UnknownXID           (20),
    UnknownLID           (21),
    ChallengeMismatch    (22)
}
```

Gambar III-50 Kode Kesalahan (ErrorCode)

Keterangan untuk masing-masing kesalahan terdapat dalam tabel berikut ini.

unspecifiedFailure	Alasan kesalahan tidak terdapat dalam daftar kesalahan ini.
messageNotSupported	Pesan ini tidak dapat didukung oleh si penerima.
decodingFailure	Kesalahan terdapat pada proses pengkodean DER pada pesan.
invalidCertificate	Sertifikat yang diperlukan untuk memproses pesan tidak cocok. *Field* ErrorThumb pada ErrorTBS menyimpan sertifikat yang salah ini.
expiredCertificate	Sertifikat yang diperlukan untuk memproses pesan telah kadaluarsa. *Field* ErrorThumb pada ErrorTBS menyimpan sertifikat yang salah ini.
revokedCertificate	Sertifikat yang diperlukan untuk memproses pesan telah ditarik/ditolak. *Field* ErrorThumb pada ErrorTBS menyimpan sertifikat yang salah ini.
missingCertificate	Sertifikat yang diperlukan untuk memproses pesan tidak ditemukan dalam pesan tersebut maupun dalam tempat penyimpanan (*cache)* si penerima.
signatureFailure	Tanda tangan digital pada pesan tidak dapat diverifikasi.
badMessageHeader	MessageHeader pada pesan tidak dapat diproses.
wrapperMsgMismatch	Isi dari MessageWrapper tidak konsisten dengan isi di dalam pesan, contohnya RRPID tidak cocok.
versionTooOld	Versi pesan yang diterima terlalu kuno untuk diproses oleh si penerima.
versionTooNew	Versi pesan yang diterima terlalu baru untuk diproses oleh si penerima.

unrecognizedExtension	Pesan atau sertifikat yang memiliki ekstension yang tidak dapat diproses oleh si penerima. *Field* ErrorOID pada ErrorTBS mengidentifikasikan OID ekstension tersebut sedangkan ErrorThumb mengidentifikasikan sertifikat yang mengalami kesalahan ekstension tersebut.
messageTooBig	Pesan terlalu besar untuk diproses oleh si penerima.
signatureRequired	Pesan memerlukan tanda tangan digital.
messageTooOld	Pesan terlalu tua untuk diproses.
messageTooNew	Pesan terlalu baru untuk diproses.
thumbsMismatch	*thumb* yang dikirimkan pada pesan yang tidak tertandatangani tidak cocok dengan *thumb* yang diterima kembali.
unknownRRPID	RRPID tidak dikenali.
unknownXID	XID tidak dikenali.
unknownLID	LID tidak dikenali.
challengeMismatch	*challenge* (nilai acak) yang dikirimkan pada *request* tidak sama dengan yang dikirimkan pada *response*.

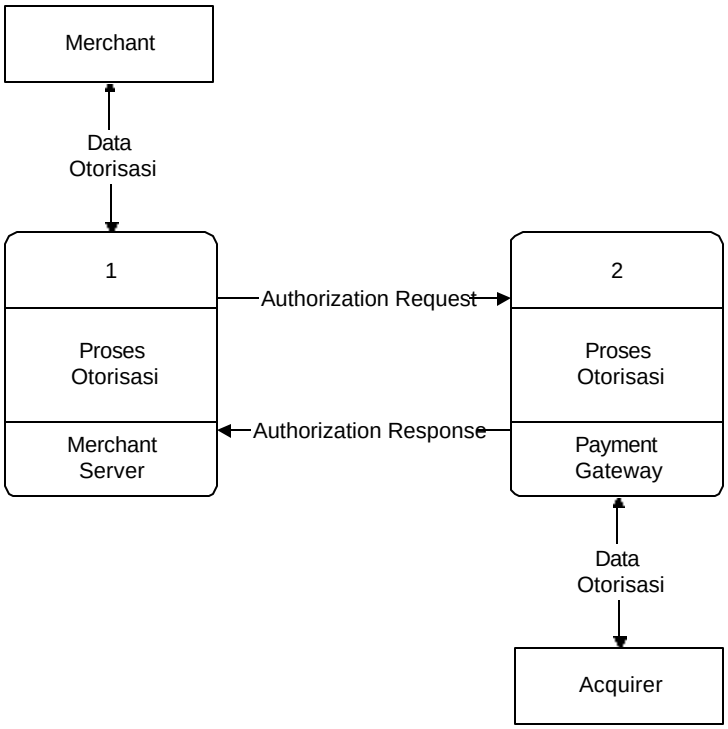
Tabel III-20 Keterangan ErrorCode

1.11 PROSES

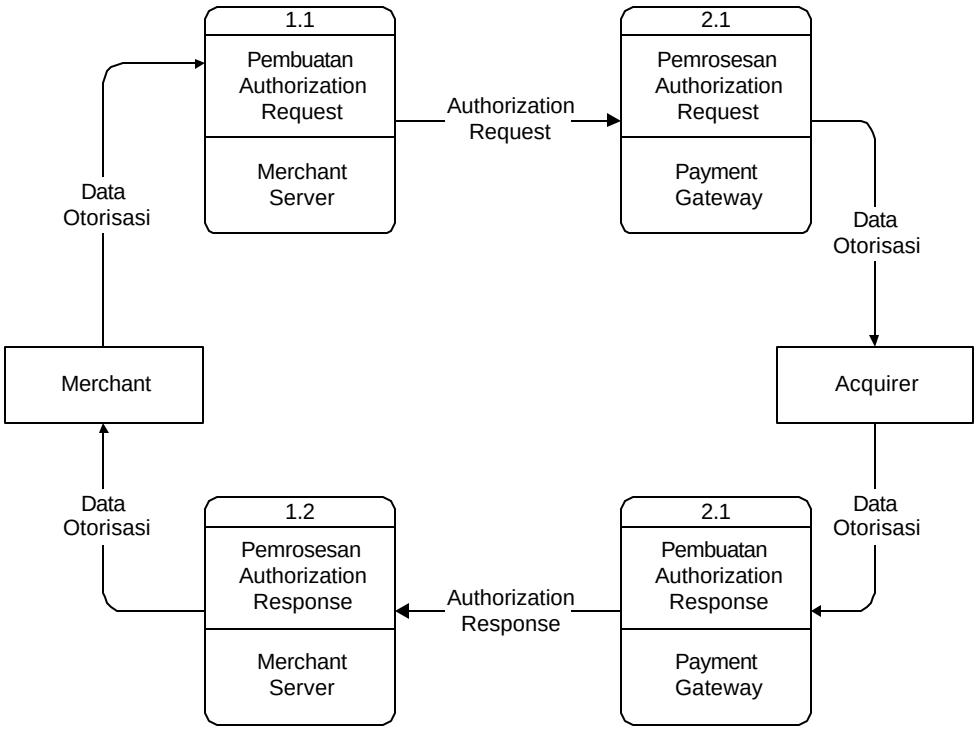
Pembahasan proses langsung berfokus pada ruang lingkup penelitian ini yaitu pada proses otorisasi antara *merchant* dan *payment gateway.* Pembahasan juga dilengkapi dengan Diagram Alur Data (DAD). Untuk lebih mengerti DAD proses ini, sebaiknya kita juga mengacu pada *tree-tree* ASN.1 untuk *authorization request* dan *authorization response*.

Pada DAD tingkat 0 atau kita kenal dengan sebutan diagram kontekstual, terdapat dua proses yaitu proses otorisasi di *merchant* dan proses otorisasi di *payment gateway.* Pada tingkat ini data-data yang tercantum belum spesifik dijelaskan.

Kedua proses tersebut dipecah menjadi empat proses pada DAD tingkat 1. Proses otorisasi di *merchant* dipecah menjadi proses pembuatan *authorization request* dan proses pemrosesan *authorization response*; sedangkan pada proses otorisasi di *payment gateway* dipecah menjadi proses pemrosesan *authorization request* dan pembuatan *authorization response*. Data-data yang tercantum pada DAD tingkat 1 ini juga belum spesifik dijelaskan. Data –data tersebut akan dicantumkan secara spesifik pada DAD tingkat 2.



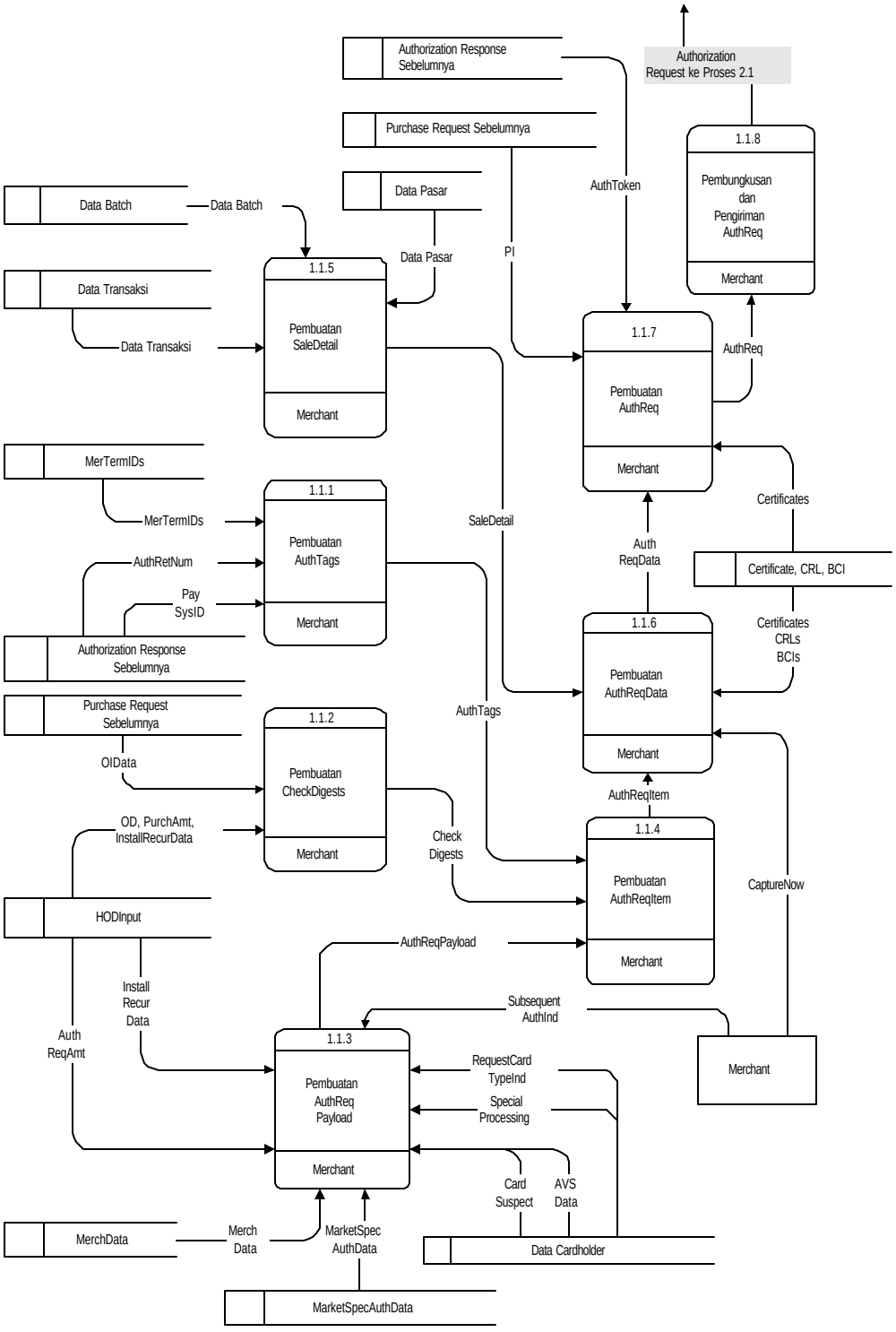
Gambar III-51 Diagram Konteks Proses Otorisasi



Gambar III-52 DAD Tingkat 1 - Proses Otorisasi

Keempat proses pada DAD tingkat 1 ini akan dijelaskan masing-masing secara terpisah pada empat sub-bab beerikut ini.

1.11.1 Pembuatan *Authorization Request*



Gambar III-53 Pembuatan *Authorization Request*

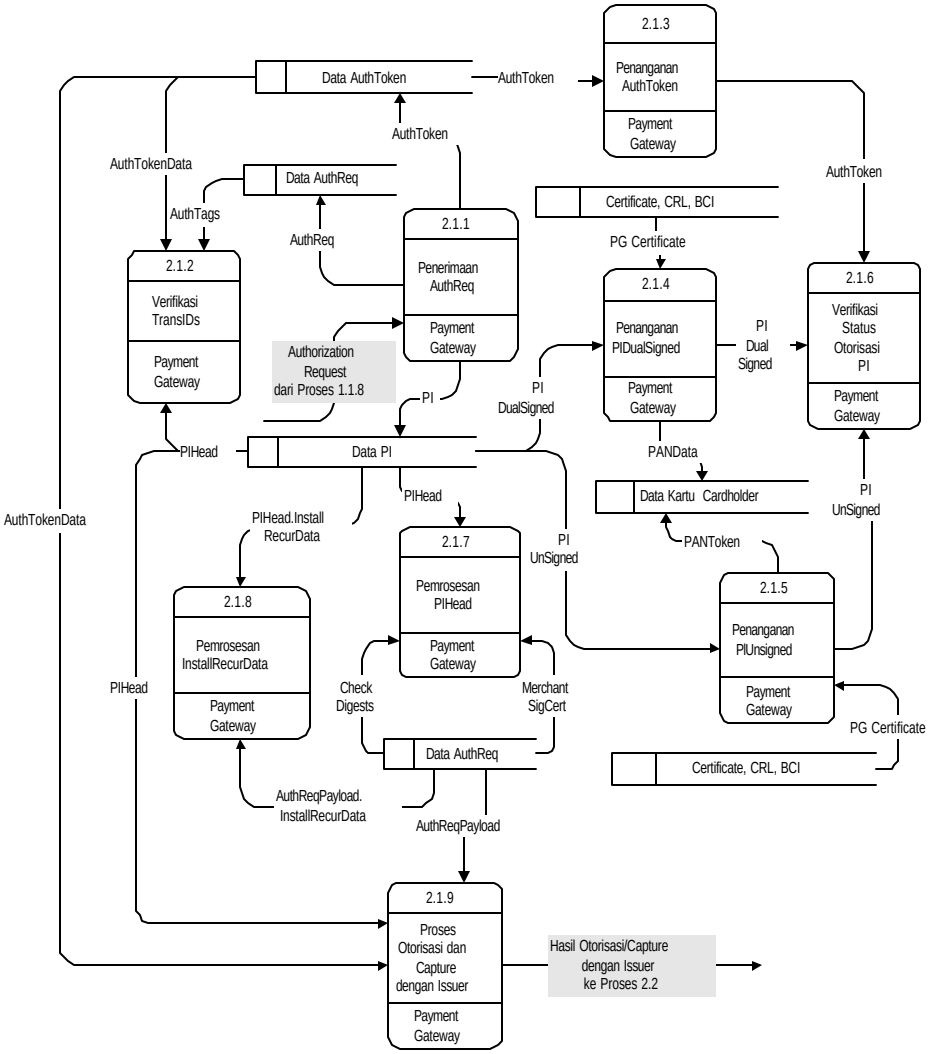
Urutan proses pembuatan *authorization request* dan keterangan untuk setiap proses terdapat dalam tabel berikut ini.

Pembuatan AuthTags	• Isi AuthRRTags *lihat halaman 54* • Isi TransIDs, apabila ini adalah otorisasi berantai dan PaySysID telah didefinisikan, isi PaySysID. • Apabila ini adalah transaksi dengan pembayaran terpisah atau berulang-ulang dan *acquirer* telah memeberikan AuthRetNum, isi AuthRetNum berdasarkan AuthRes sebelumnya.
Pembuatan CheckDigests	• Buat HOD2 dengan meng-*hash* OD, PurchAmt, ODSalt, ODExtensions dan InstallRecurData. *lihat penjelasan pada sub-bab 1.10.61* • Buat HOIData dengan meng-*hash* OIData yang didapat dari *purchase request* (Preq). *lihat penjelasan pada sub-bab 1.10.61*
Pembuatan AuthReqPayload	Isi *field-field* AuthReqPayload sesuai dengan penjelasan AuthReqPayload *lihat penjelasan pada sub-bab 1.10.61*
Pembuatan AuthReqItem	Gabungkan AuthTags, CheckDigests, dan AuthReqPayload.
Pembuatan SaleDetail	Isi *field-field* SaleDetail sesuai dengan penjelasan SaleDetail. *lihat penjelasan sub-bab 1.10.59*
Pembuatan AuthReqData	• Isi MThumbs dengan membuat *digest* dari sertifikat-sertifikat dan daftar-daftar sertifikat yang tidak berlaku. • Isi *field* CaptureNow sesuai kehendak *merchant*. • Gabungkan AuthReqItem, Mthumbs, CaptureNow dan SaleDetail.
Pembuatan AuthReq	Lakukan proses EncB(M,P,AuthReqData,PI). Cat: PI didapat dari Preq atau AuthRes sebelumnya.

Pembungkusan dan Pengiriman AuthReq	Bungkus AuthReq dengan MessageWrapper dan kirimkan melalui jaringan kepada *payment gateway*. *Lihat penjelasan pada sub-bab 1.10.7*

Tabel III-21 Keterangan Proses Pembuatan AuthReq

1.11.2 Pemrosesan *Authorization Request*



Gambar III-54 Pemrosesan *Authorization Request*

Urutan proses pemrosesan *authorization request* dan keterangan untuk setiap proses terdapat dalam tabel berikut ini.

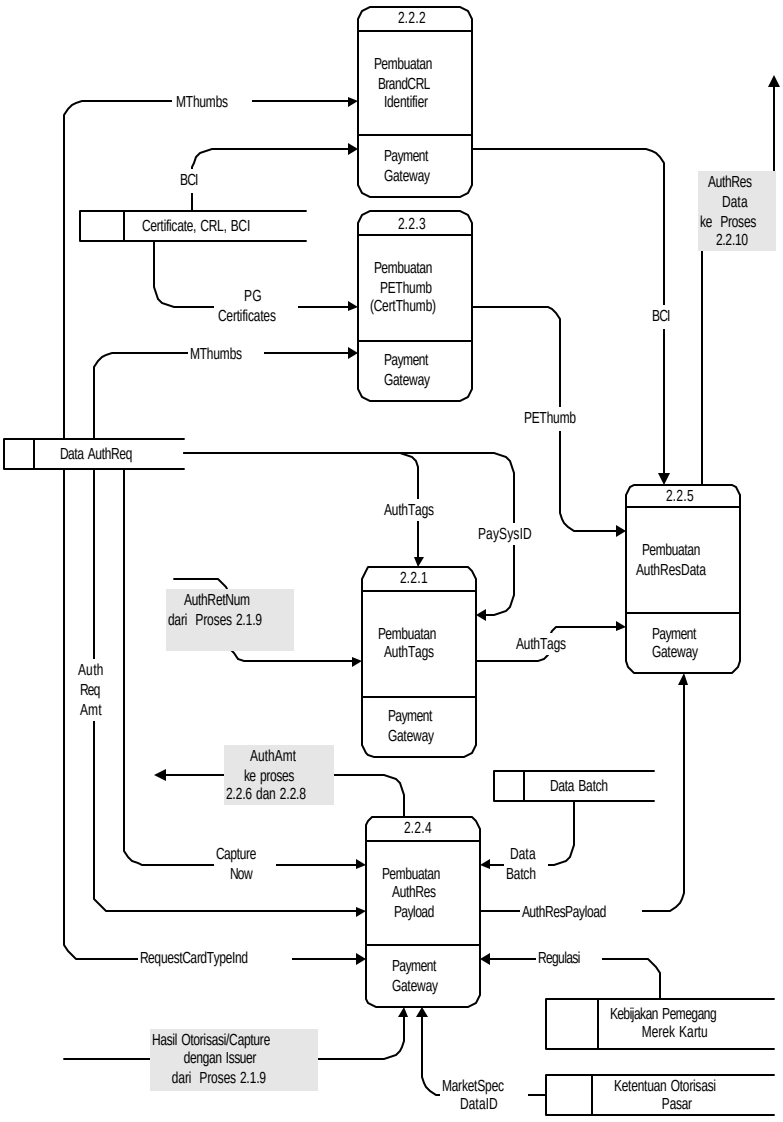
Penerimaan AuthReq	• Menerima AuthReq dari jaringan. • Membuka dan memverifikasi MessageWrapper dan Message.
Verifikasi TransIDs	Bandingkan TransIDs pada AuthTags dengan TransIDs pada PIHead atau AuthTokenData. Apabila terdapat perbedaan, kirim AuthRes dengan AuthCode “piAuthMismatch”. Proses selanjutnya, pilih salah satu dari ketiga proses di bawah ini sesuai dengan jenis PI yang diterima.
Penanganan AuthToken	Apabila PI yang diterima berupa AuthToken, berarti sebelumnya sudah terjadi proses otorisasi. Apabila terdapat InstallRecurData dan berisi *field* Recurring: • Periksa apakah tanggal RecurringExpiry sudah lewat dari tanggal sekarang. Jika sudah, isi AuthCode pada AuthRes dengan “recurringExpired”. • Periksa apakah tanggal sekarang sudah lewat dari tanggal yang tertera pada *field* PrevAuthDate ditambah jumlah hari yang tertera pada *field* RecurringFrequency, jika belum isi AuthCode pada AuthRes dengan “recurringToSoon”. • Jika pada InstallRecurData terdapat informasi tambahan pada IRExtensions, lakukan pemrosesan sesuai dengan kebijakan pemegang merek kartu pembayaran.
Penanganan PIDualSigned	• Verifikasi apakah merek yang tertera pada sertifikat tanda tangan *cardholder* sesuai dengan merek yang tertera pada sertifikat pertukaran pesan *payment gateway*. Jika tidak isi AuthCode dengan “CardMerchBrandMismatch”. • Verifikasi PANData • Verifikasi identitas *cardholder* menggunakan PAN, CardExpiry, dan PANSecret dari PANData dan bandingkan dengan identitas *cardholder* pada *field* CommonName yang terdapat pada sertifikat *cardholder*. Jika verifikasi gagal, kirimkan

	pesan Error dengan pesan “SignatureFailure”. • Verifikasi SO. • Simpan data dari PANData.
Penanganan PIUnsigned	• Periksa apakah sertifikat *payment gateway* mengindikasikan bahwa ia tidak memerlukan sertifikat *cardholder*. Jika tidak, isi AuthCode dengan “signatureRequired”. • Verifikasi *hash* dalam EXH. • Simpan data dari PANToken.
Verifikasi Status Otorisasi PI	Verifikasi apakah PI pernah diproses dan tidak pernah mengalami penundaan atau pengembalian. Jika ya, tolak otorisasi dengan AuthCode berisi “piPreviouslyUsed”.
Pemrosesan PIHead	• Verifikasi apakah PIHead.MerchantID sesuai dengan *field* merID pada ekstension merchantData di dalam sertifikat tanda tangan *merchant* yang digunakan untuk memverifikasi tanda tangan *merchant* pada pesan AuthReq. Jika tidak sesuai, tolak otorisasi dengan AuthCode berisi “piAuthMismatch”. • Kirimkan TransStain kepada *issuer* melalui jaringan finansial untuk verifikasi atau simpan TransStain untuk verifikasi secara *off-line*. • Verifikasi HOIData yang diterima dari PI dan CheckDigests, apabila tidak sesuai, tolak otorisasi dengan AuthCode berisi “piAuthMismatch”. • Verifikasi HOD dari PIHead dan HOD2 dari CheckDigests, apabila tidak sesuai, tolak otorisasi dengan AuthCode berisi “piAuthMismatch”. • Proses PIExtensions. Apabila ditemukan ekstension yang tidak dapat dilayani, tolak otorisasi dengan mengirimkan pesan Error yang berisi “unrecognizedExtensions”.
Pemrosesan InstallRecurData	Apabila InstallRecurData terdapat dalam AuthReq, verifikasi apakah InstallRecurData dalam

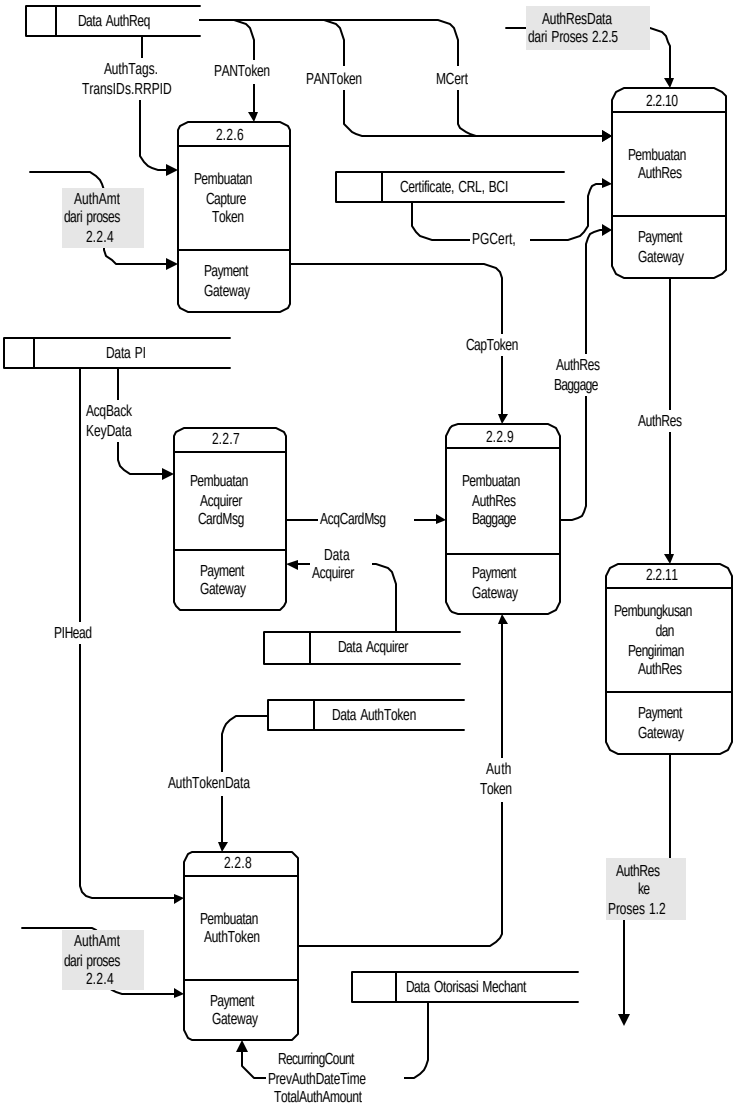
	AuthReqPayload sesuai dengan yang terdapat dalam PIHead. Jika mereka tidak sesuai, tolak otorisasi dengan AuthCode berisi "InstallRecurMismatch".
Proses Otorisasi dan *Capture* dengan *Issuer*	• Periksa apakah CaptureNow adalah *true*. Jika sistem pembayaran ini tidak dapat melayani *capture*, kirimkan AuthRes dengan AuthCode berisi "capturedNotSupport". • Lakukan proses otorisasi dan *capture* (jika diminta) melalui jaringan finansial dengan *issuer*.

Tabel III-22 Keterangan Proses Pemrosesan AuthReq

1.11.3 Pembuatan Authorization Response



Gambar III-55 Pembuatan *Authorization Response* (1)



Gambar III-56 Pembuatan *Authorization Response* (2)

Urutan proses pembuatan *authorization response* dan keterangan untuk setiap proses terdapat dalam tabel berikut ini.

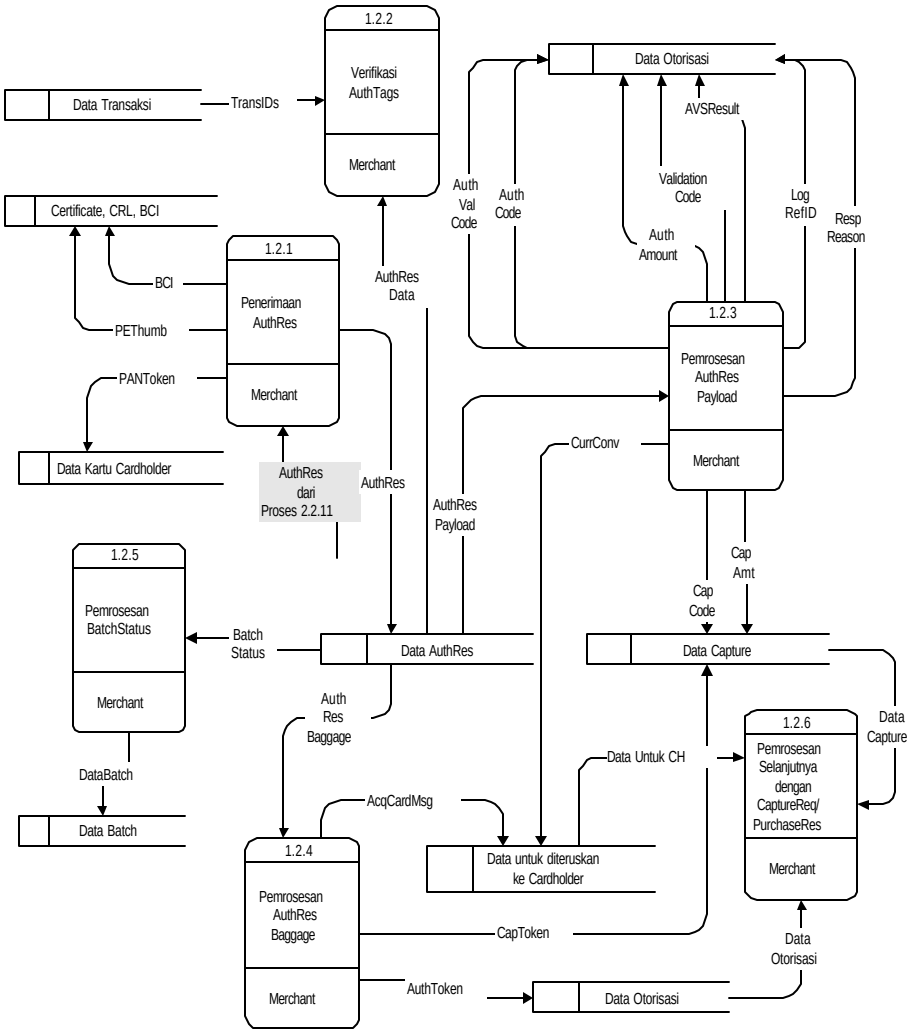
Pembuatan AuthTags	Isi AuthTags sesuai dengan yang diterima pada AuthReq. Apabila diperlukan bagi jaringan merek kartu tertentu, isi AuthRetNum yang diterima dalam proses otorisasi dengan *issuer*.
Pembuatan BrandCRLIdentifier	Isi BrandCRLIdentifier dengan yang dimiliki oleh *payment gateway* apabila Thumbs untuk BrandCRLIdentifier yang dikirimkan *merchant* pada AuthReq tidak ada atau sudah ketinggalan.

Pembuatan PEThumb	Berisi *digest* dari sertifikat-sertifikat *payment gateway* yang diperlukan oleh *merchant*.
Pembuatan AuthResPayload	Isi *field-field* dalam AuthResPayload, sesuai dengan penjelasan AuthRes. *lihat sub-bab 1.10.62*
Pembuatan AuthResData	Gabungkan AuthTags, BrandCRLIdentifier, PEThumb dan AuthResPayload.
Pembuatan CaptureToken	• Isi *field* AuthAmt pada *CapTokenData* sesuai dengan AuthAmt dalam AuthRes. • Isi TokenOpaque pada CapTokenData dengan data khusus yang diperlukan untuk proses kliring. • Jika *merchant* biasanya menerima PANToken dari *acquirer*, isi PANToken yang didapat dari PI dan gunakan enkapsulasi EncX untuk membentuk CapToken. Jika tidak, gunakan enkapsulasi Enc.
Pembuatan AcquirerCardMsg	Isi *field* AcqCardMsg pada AuthResBaggage apabila diperbolehkan dalam kebijakan pemegang merek kartu pembayaran untuk menerima kunci dari *cardholder* dan mengirimkan pesan kepadanya.
Pembuatan AuthToken	√ Apabila ini adalah otorisasi yang pertama: • Isi PANToken, TransIDs, PurchAmt, MerchID, AcqBackInfo dan InstallRecurData dari PIHead. • Isi RecurringCount dengan 1. • Isi PrevAuthDate dengan tanggal sekarang. • Isi TotalAuthAmount dengan AuthAmt dari AuthRes yang akan mengikutsertakan AuthToken ini. √ Apabila ini adalah otorisasi lanjutan: • Isi PANToken, TransIDs, PurchAmt, MerchID, AcqBackInfo dan InstallRecurData dari AuthToken sebelumnya. • Tambahkan RecurringCount dengan 1. • Isi PrevAuthDate dengan tanggal sekarang. • Tambahkan TotalAuthAmount dengan AuthAmt

	dari AuthRes yang akan mengikutsertakan AuthToken ini. √ Langkah selanjutnya dari dua langkah di atas, adalah membuat PANToken dari data *cardholder* pada PI sebelumnya (termasuk AuthToken). √ Buat EncX {P1, P2, AuthTokenData, PANToken} P1 dan P2 adalah *payment gateway*.
Pembuatan AuthResBaggage	Gabungkan CapToken, AcqCardMsg, dan AuthToken
Pembuatan AuthRes	Apabila PANToken *cardholder* diikutsertakan, gunakan enkapsulasi EncBX jika tidak gunakan enkapsulasi EncB untuk membungkus AuthResData dan AuthResBaggage.
Pembungkusan dan pengiriman AuthRes	Bungkus AuthRes dengan MessageWrapper dan kirimkan melalui jaringan kepada *merchant*.

Tabel III-23 Keterangan Proses Pembuatan AuthRes

1.11.4 Pemrosesan *Authorization Response*



Gambar III-57 Pemrosesan *Authorization Response*

Urutan proses pembuatan *authorization response* dan keterangan untuk setiap proses terdapat dalam tabel berikut ini.

Penerimaan AuthRes	• Menerima AuthRes dari jaringan. • Membuka dan memverifikasi MessageWrapper dan Message. • Bila PANToken ada, simpan dalam basis data lokal yang aman.
Verifikasi AuthTags	Bandingkan AuthTags dengan data transaksi yang terdapat di *merchant*, bandingkan komponen TransIDs-nya. • Jika XID tidak sesuai, catat pada *log* kesalahan "unknown XID". • Jika LID-C atau LID-M tidak sesuai, catat pada *log* kesalahan dengan *payment gateway* "unknown LID".
Pemrosesan AuthResPayload	√ Proses ArsExtensions bila ada. Apabila ekstension ini penting dan tidak dikenal, catat pada *log* kesalahan dengan *payment gateway* "unrecognizedExtensions". √ Proses CapResPayload bila ada: • Proses CrsExtensions bila ada. Apabila ekstension ini penting dan tidak dikenal, catat pada *log* kesalahan dengan *payment gateway* "unrecognizedExtensions". • Proses CapCode untuk menjelaskan hasilnya. • Proses SaleDetail sesuai dengan kebijakan pemegang merek kartu pembayaran. • Untuk *capture* yang berhasil, simpan CapCode dan CapAmt √ Apabila CurrConv ada, simpan untuk diteruskan ke *cardholder*. √ Proses AuthCode, AuthAmt dan Response Data: • Proses AuthCode untuk menjelaskan hasilnya. • Simpan AuthCode dan AuthAmt untuk otorisasi

	yang berhasil. • Simpan ValidationCode untuk otorisasi yang berhasil. • Simpan AuthValCodes apabila ada. • Proses RespReason apabila ada. • Simpan AVSResult apabila ada. • Simpan LogRefID apabila ada.
Pemrosesan AuthResBaggage	• Simpan CapToken apabila ada. • Apabila AcqCardMsg ada, simpan untuk diteruskan ke *cardholder*. • Simpan AuthToken untuk otorisasi berikutnya.
Pemrosesan BatchStatus	Proses BatchStatus dan simpan datanya apabila ada.
Pemrosesan Selanjutnya dengan *Capture Request* atau *Purchase Response.*	Kemungkinan proses lanjutnya dari *merchant* adalah mengirimkan: • *capture request* • *authorization request* • *purchase response*

Tabel III-24 Keterangan Proses Pemrosesan AuthRes

BAB IV
IMPLEMENTASI

1.12 PEMILIHAN BAHASA PEMROGRAMAN

Pada penelitan ini, penulis menggunakan bahasa Java sebagai bahasa pemrograman yang dipakai dalam pengimplementasian *library* objek SET ini. Adapun alasan pemilihan bahasa Java ini adalah:

1. Bahasa Java memiliki kelas-kelas (*class)* struktur data dan kelas bantu (*utility*) yang lengkap dan mudah pemakaiannya.
2. Pemrogram tidak perlu khawatir dengan manajemen *memory* dan pemakaian *pointer*, bahasa Java menyediakan banyak fasilitas untuk manajemen *memory* seperti *garbage collection*[6], *array* yang bisa berkembang sendiri, dsb; berbeda dengan bahasa C yang menuntut pemrogram harus teliti dengan masalah pemakaian *memory*.
3. *Class library* dalam Java sangat lengkap, terutama untuk kriptografi. Banyak *vendor* yang telah menyediakan *library* kriptografi dalam bahasa Java seperti Sun Microsystem, Australian Business Association (ABA) dan Cryptix.
4. Bahasa Java adalah bahasa yang berorientasi objek (*object oriented*), sesuai dengan spesifikasi SET yang juga berorientasi objek.

Software Development Kid untuk bahasa Java yang digunakan adalah *Java Development Kid* (JDK) versi 1.2.

JDK 1.2 digunakan karena pada saat implementasi dimulai versi inilah yang terbaru dan memiliki kelengkapan-kelengkapan yang mendukung untuk keamanan dan sertifikat.

---

[6] *Garbage collection* membebaskan semua blok *memory* yang tidak terpakai dan tidak ada pemiliknya.

1.13 TAHAPAN IMPLEMENTASI

Implementasi prototipe proses otorisasi kartu pembayaran antara *merchant* dan *payment gateway* dibagi dalam dua tahapan, yaitu implementasi pesan dan implementasi proses.

Implementasi pesan juga dibagi dalam beberapa tahapan. Pembagian itu berdasarkan struktur lapisan objek SET yang digambarkan pada bab analisa dan perancangan (hal 39). Berdasarkan struktur lapisan objek SET tersebut, maka tahapan implementasi *library* objek SET adalah sebagai berikut:

1. Implementasi objek pembantu yang berfungsi mengkodekan (*encode*) objek-objek di lapisan atasnya ke dalam kumpulan *byte* DER dan menerjemahkan (*decode*) kumpulan *byte* DER tersebut ke dalam objek semula.
2. Implementasi objek-objek yang didefinisikan oleh PKCS, X.509 dan OAEP.
3. Implementasi objek-objek operator kriptografi SET.
4. Implementasi objek-objek komponen pesan.
5. Implementasi objek-objek pesan (*message*) dan pesan kesalahan (*error*).
6. Implementasi objek pembungkus pesan (*message wrapper*).

Beberapa tahap dari tahapan implementasi tersebut tidak murni dilakukan, karena penulis menggunakan beberapa objek yang sudah diimplementasikan oleh *vendor-vendor* lain. Penjelasan lebih lanjut dapat dilihat pada penjelasan objek.

Implementasi proses hanya terdiri empat bagian yaitu implementasi proses pembuatan AuthReq, pemrosesan AuthReq, pembuatan AuthRes dan pemrosesan AuthRes sesuai dengan DAD pada sub-bab 1.11.

1.14 IMPLEMENTASI PESAN

Pesan SET disusun atas objek-objek. Implementasi pesan SET merupakan implementasi objek-objek yang terkait dalam penyusunan sebuah pesan. Seluruh objek tersebut untuk lebih mudahnya kita sebut saja objek-objek pesan SET.

1.14.1 Karakteristik Objek Pesan SET

Ada dua macam objek pesan SET yang akan diimplementasikan pada penelitan ini. Kedua macam objek tersebut adalah objek bantu (*tools*) DER dan objek penyusun. Keduanya memiliki karakteristik yang berbeda.

1. Objek Bantu DER *(DER Tools)*

Objek bantu DER yang diperlukan dalam implementasi ini adalah suatu objek yang berfungsi untuk menerjemahkan suatu nilai dalam tipe data ASN.1 yang dipadankan dengan tipe data Java ke dalam kumpulan *byte* Java dalam representasi DER, dan juga dapat menerjemahkan kembali kumpulan *byte* tersebut ke dalam nilai semula dalam tipe data Java.

Pada penelitian ini, penulis tidak mengimplementasi objek bantu DER tersebut, namun memakai implementasi dari Sun Microsystem yang disertakan dalam paket JDK1.2..

Objek-objek bantu DER tersebut adalah `sun.security.util.DerOutputStream`, `sun.security.util.DerInputStream` dan `sun.security.util.DerValue`[7].

DEROutputStream

Objek DerOutputStream berfungsi sebagai tempat untuk menuliskan data dalam tipe data Java (primitif maupun objek) yang sepadan dengan tipe data ASN.1 dalam bentuk DER. Objek ini menampung kumpulan *byte* (byte []) hasilnya.

```
Constructor
 DerOutputStream()
 DerOutputStream(int i)

 Method
  void derEncode(java.io.OutputStream outputstream)
  void putBitString(byte[] abyte0)
  void putBoolean(boolean flag)
  void putDerValue(DerValue dervalue)
  void putEnumerated(BigInt bigint)
  void putGeneralizedTime(java.util.Date date)
  void putIA5String(java.lang.String s)
  void putInteger(BigInt bigint)
  void putLength(int i)
  void putNull()
  void putOctetString(byte[] abyte0)
  void putOID(ObjectIdentifier objectidentifier)
```

[7] sun.security.util adalah nama sebuah paket dalam Java, sedangkan DerInputStream , DerOutputStream, DerValue adalah nama kelasnya.

```
  void putOrderedSet(byte byte0, DerEncoder[] aderencoder)
  void putOrderedSetOf(byte byte0, DerEncoder[] aderencoder)
  void putPrintableString(java.lang.String s)
  void putSequence(DerValue[] adervalue)
  void putSet(DerValue[] adervalue)
  void putTag(byte byte0, boolean flag, byte byte1)
  void putUnalignedBitString(BitArray bitarray)
  void putUTCTime(java.util.Date date)
  void write(byte byte0, byte[] abyte0)
  void write(byte byte0, DerOutputStream deroutputstream)
  void writeImplicit(byte byte0, DerOutputStream deroutputstream)
  byte[] toByteArray()
```

Gambar IV-1 Ringkasan Objek DerOutputStream

Objek DerOutputStream ini menyediakan *method-method* untuk meletakkan berbagai jenis data dalam tipe data Java ke dalam objek ini (semua *method* yang berawalan "put"). Data yang diletakkan dalam objek ini otomatis dikodekan dengan aturan DER.

Setelah seluruh penulisan selesai dilakukan, hasil yang ditampung sementara dalam objek ini dapat disalin ke tipe data *byte-array* (byte []) yang merupakan tipe data primitif pada Java. *Method* yang digunakan untuk melakukan hal ini adalah *method* `toByteArray()`. *Byte-array* merupakan tipe data yang digunakan Java untuk pengiriman pesan lewat `java.io.Socket` maupun `java.io.ServerSocket`.

DerInputStream

DerInputStream merupakan objek yang dapat menerima *byte-array* dalam representasi DER dan menerjemahkannya dalam tipe-tipe data Java yang sesuai dengan kode DER pada *byte-array* tersebut.

```
Constructor
 DerInputStream(byte[] abyte0)
 DerInputStream(byte[] abyte0, int i, int j)

Method Summary
 int available()
 byte[] getBitString()
 void getBytes(byte[] abyte0)
 DerValue getDerValue()
```

```
BigInt getEnumerated()
java.util.Date getGeneralizedTime()
BigInt getInteger()
void getNull()
byte[] getOctetString()
ObjectIdentifier getOID()
DerValue[] getSequence(int i)
DerValue[] getSet(int i)
DerValue[] getSet(int i, boolean flag)
BitArray getUnalignedBitString()
java.util.Date getUTCTime()
void mark(int i)
int peekByte()
void reset()
DerInputStream subStream(int i, boolean flag)
byte[] toByteArray()
```

Gambar IV-2 Ringkasan Objek DerInputStream

Constructor pada DerInputStream menerima *byte-array* dalam representasi DER dan menyimpan *byte-byte* tersebut di dalam objek ini.

Data-data dalam *byte-array* tersebut dapat terdiri atas satu atau beberapa blok nilai DER, setiap blok merepresentasikan satu nilai dalam tipe ASN.1. DerInputStream menyediakan *method-method* untuk mengambil setiap blok nilai DER tersebut dan menerjemahkannya ke dalam suatu nilai dalam tipe data Java.

DerInputStream memiliki *method-method* yang tersedia sesuai dengan tipe data Java yang diinginkan. Contohnya apabila ia ingin mengambil dan menerima suatu nilai dalam tipe data Date, ia dapat memakai *method* `getGeneralizedTime()`.

DerValue

Objek DerValue adalah objek pelengkap bagi objek DerOutputStream dan objek DerInputStream.

```
Constructor
 DerValue(byte[] abyte0)
 DerValue(byte[] abyte0, int i, int j)
 DerValue(byte byte0, byte[] abyte0)
 DerValue(java.io.InputStream inputstream)
 DerValue(java.lang.String s)
```

```
Method
 static byte createTag(byte byte0, boolean flag, byte byte1)
 void encode(DerOutputStream deroutputstream)
 boolean equals(DerValue dervalue)
 boolean equals(java.lang.Object obj)
 java.lang.String getAsString()
 byte[] getBitString()
 byte[] getBitString(boolean flag)
 boolean getBoolean()
 BigInt getEnumerated()
 java.lang.String getIA5String()
 BigInt getInteger()
 BigInt getInteger(boolean flag)
 byte[] getOctetString()
 ObjectIdentifier getOID()
 java.lang.String getPrintableString()
 java.lang.String getT61String()
 BitArray getUnalignedBitString()
 BitArray getUnalignedBitString(boolean flag)
 boolean isConstructed()
 boolean isContextSpecific()
 boolean isContextSpecific(byte byte0)
 int length()
 void resetTag(byte byte0)
 byte[] toByteArray()
 DerInputStream toDerInputStream()
 java.lang.String toString()
```

Gambar IV-3 Ringkasan Objek DerValue

Objek DerValue ini menyediakan *method-method* "get" yang tidak tersedia dalam objek DerInputStream.

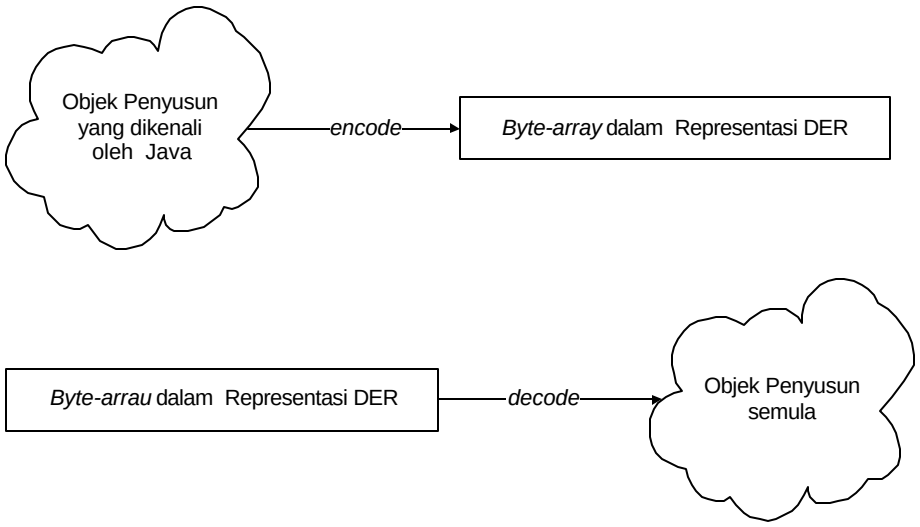
2. Objek Penyusun Pesan SET

Objek penyusun adalah objek-objek yang dapat diikutsertakan untuk menyusun objek penyusun di atasnya, termasuk objek yang paling atas.

Setiap objek penyusun pada implementasi ini memiliki kesamaan sifat seperti yang dijelaskan berikut ini:

1. Memiliki *method* untuk mengkodekan (*encode*) objek ini ke dalam nilai *byte-array* dalam representasi DER dan menerjemahkan (decode) nilai *byte-array*

DER objek tersebut ke dalam bentuk objek mula-mula yang dimengerti oleh Java.



Gambar IV-4 *Encode* dan *Decode* Objek

Suatu objek yang dibentuk oleh beberapa objek di lapisan bawahnya, ketika hendak meng-*encode* dirinya sendiri melakukan penggabungan hasil *encode* dari objek-objek penyusunnya.

2. Memiliki *field-field* sesuai dengan spesifikasi ASN.1 dan memastikan *field-field* tersebut dapat diakses.
3. Untuk setiap objek-objek operator kriptografi SET, dilengkapi dengan modul verifikasi sesuai dengan analisa pada bab tiga.

1.14.2 Contoh Implementasi Sebuah Objek Pesan

Berikut ini akan ditampilkan sebuah contoh pengimplementasian objek TransIDs berdasarkan kedua sifat di atas.

```
/**
 *          <pre>
 *          337 TransIDs ::= SEQUENCE {
 *          338    lid-C      LocalID,
 *          339    lid-M      [0] LocalID  OPTIONAL,
 *          340    xid        XID,
 *          341    pReqDate   Date,
 *          342    paySysID   [1] PaySysID  OPTIONAL,
 *          343    language   Language         -- Cardholder requested session language
 *          344 }
 *
 *          348 XID ::= OCTET STRING (SIZE(20))
 *
 *          320 PaySysID ::= VisibleString (SIZE(1..ub-paySysID=64))
 *
 *          282 Language ::= VisibleString (SIZE(1..ub-RFC1766-language=35))
 *
 *          284 LocalID ::= OCTET STRING (SIZE(1..20))
 *
 *          265 Date ::= GeneralizedTime
```

```
*          </pre>
*/
```

Gambar IV-5 Komentar Program tentang Spesifikasi TransIDs dalam ASN.1

Potongan program di atas merupakan bagian komentar program yang menerangkan tentang spesifikasi TransIDs dalam ASN.1.

TransIDs merupakan SEQUENCE dari:

1. lid-c dengan tipe LocalID
2. lid-m dengan tipe LocalID namun *field* ini bersifat *optional* dan bila ada , memiliki *tag Context Specific* [0] dan bersifat IMPLICIT. Karena LocalID adalah OCTET STRING dan merupakan tipe primitif, maka nilai *tag*-nya adalah `0x80`.
3. xid dengan tipe XID, XID adalah OCTET STRING.
4. pReqDate dengan tipe Date, Date adalah GENERALIZED TIME.
5. paySysId dengan tipe PaySysID namun *field* ini bersifat *optional* dan bila ada, memiliki *tag ContextSpecific* [1] dan bersifat IMPLICIT. Karena PaySysID bertipe VISIBLE STRING dan merupakan tipe primitif, maka nilai *tag*-nya adalah `0x81`.
6. language dengan tipe Language, Language adalah VISIBLE STRING

```
package digsec.set.payment.component;

import sun.security.util.*;
import java.util.Date;
import java.security.SecureRandom;
import sun.misc.HexDumpEncoder;
import java.io.*;

public class TransIDs implements DerEncoder {

        //fields, ada penyesuaian tipe antara ASN.1 dan Java
        public byte[]       lid_C;                  // OCTET STRING = byte []
        public byte[]       lid_M;                  // OCTET STRING = byte []
        public byte[]       xid;                    // OCTET STRING = byte []
        public Date         pReqDate;               // GENERALIZED TIME = Date
        public String       paySysId;               // VISIBLE STRING = String
        public String       language="English";      // VISIBLE STRING = String

        //Constructor
        public TransIDs(byte[] lid_C,
                        byte[] lid_M,
                        byte[] xid,
                        Date   pReqDate,
                        String paySysId,
                        String language) throws Exception {

                if(lid_C.length<1 || lid_C.length>20)
                        throw new IllegalArgumentException("TransIDs::lid-C");
                this.lid_C=lid_C;

                if(lid_M!=null && (lid_M.length<1 || lid_M.length>20))
                        throw new IllegalArgumentException("TransIDs::lid-M");
                this.lid_M=lid_M;

                if(xid!=null && xid.length != 20)
                        throw new IllegalArgumentException("TransIDs::XID");
                this.xid = xid;

                this.pReqDate = pReqDate;

                this.paySysId=paySysId;

                this.language=language;
        }
```

Gambar IV-6 Awal program TransIDs

. Pada potongan program di atas terlihat bahwa jika dalam ASN.1 suatu komponen bertipe OCTET STRING, maka dalam implementasi di Java bertipe `byte []`. Jika dalam ASN.1 bertipe VISIBLE STRING, maka dalam Java bertipe String. Jika dalam ASN.1 bertipe GENERALIZED TIME, dalam Java bertipe Date. Ini semua menggambarkan bahwa ada penyesuaian tipe data dari ASN.1 ke Java. Disinilah DerOutputStream, DerInputStream dan DerValue berperan, akan lebih jelas terlihat pada potongan program di bawah ini.

```
public TransIDs(DerInputStream din) throws Exception {

                    // din berisi array of byte DER dari TransIDs
                    din = din.getDerValue().toDerInputStream();

                    // dapatkan lid_C dari tipe OCTET STRING
                    lid_C=din.getOctetString();

                    //apabila terdapat byte dengan nilai 0x80,
                    //dapatkan lid_M dari tipe OCTET STRING
                    if((byte) din.peekByte() == (byte) 0x80) {
                            DerValue dv1 = din.getDerValue();
                            dv1.resetTag((byte)0x04);
                            lid_M=dv1.getOctetString();
                    }

                    // dapatkan xid dari OCTET STRING
                    xid = din.getOctetString();
                    pReqDate = din.getGeneralizedTime();

                    //apabila terdapat byte dengan nilai 0x81,
                    //dapatkan paySysID dari VISIBLE STRING
                    if((byte) din.peekByte() == (byte) 0x81) {
                            DerValue dv2 = din.getDerValue();
                            dv2.resetTag((byte)0x13);
                            paySysId = dv2.getVisibleString();
                    }

                    // dapatkan language dari VISIBLE STRING
                    language = din.getDerValue().getbleString();
            }

            public void derEncode(OutputStream out) throws IOException {
                    DerOutputStream dout  = new DerOutputStream();
                    DerOutputStream dout2 = new DerOutputStream();

                    // menuliskan lid_C ke dalam bentuk OCTET STRING
                    dout.putOctetString(lid_C);

                    // apabila lid_M tersedia
                    if(lid_M!=null) {
                          // menuliskan lid_M ke dalam bentuk OCTET STRING yang tag-nya diganti
                          // 0x80
                          dout2.putOctetString(lid_M);
                          dout.writeImplicit((byte)0x80,dout2);
                    }

                    // menuliskan xid ke dalam bentuk OCTET STRING
                    dout.putOctetString(xid);
                    // menuliskan pReqDate ke dalam bentuk GENERALIZED TIME
                    dout.putGeneralizedTime(pReqDate);

                    //apabila paySysId tersedia
                    if(paySysId!=null) {
                            // menuliskan paySysId ke dalam bentuk VISIBLE STRING yang
                            // tag-nya diganti 0x81
                            dout2.reset();
                            dout2.putVisibleString(paySysId);
                            dout.writeImplicit((byte)0x81,dout2);
                    }

                    // menuliskan language ke dalam bentuk VISIBLE STRING
                    dout.putVisibleString(language);

                    dout2.reset();
                    dout2.write((byte)0x30,dout);
                    out.write(dout2.toByteArray());
            }

            public void encode(DerOutputStream dout) throws Exception {
```

```
                derEncode(dout);
        }

        public String toString() {
                HexDumpEncoder hde = new HexDumpEncoder();
                String s = "<TransIDs>\n";
                s += "lid-C   :\n" + hde.encodeBuffer(lid_C) + "\n";
                if(lid_M!=null)
                s += "lid-M   :\n" + hde.encodeBuffer(lid_M) + "\n";
                s += "xid     :\n" + hde.encodeBuffer(xid) + "\n";
                s += "pReqDate:" + pReqDate + "\n";
                if(paySysId!=null)
                s += "paySysId:" + paySysId + "\n";
                s += "language:" + language + "\n";
                return s;
        }
}
```

Gambar IV-7 *Encode-Decode* TransIDs

Pada potongan program di atas, terlihat bahwa objek TransIDs ini memiliki *method* untuk *encode* maupun *decode*. Method untuk melakukan *encode* adalah `void derEncode(OutputStream out) throws IOException`, sedangkan untuk melakukan *decode* adalah melalui *constructor* `TransIDs (DerInputStream din) throws Exception`. *Constructor* tersebut menerima sebuah objek DerInputStream yang di dalamnya terdapat *byte-array* DER. Kemudian *constructor* tersebut mengubah *byte-array* DER yang terdapat dalam objek DerInputStream tersebut menjadi objek TransIDs yang baru.

1.14.3 Pemaketan Objek Pesan

Tabel berikut ini objek-objek pesan SET yang telah diimplementasikan. Objek-objek tersebut juga telah dikelompokan dalam paket-paket berdasarkan karakteristiknya.

Penulis mengimplementasikan objek-objek ini bersama rekan peneliti, Dwinanda Prayudi (Iduy).

Paket	Kelas	Pembuat
digsec.set.core.pkcs Ket: berisi objek-objek yang didefinisikan dalam PKCS	AlgorithmId.java	Sun
	ContentInfo.java	
	DigestedData.java	Arif
	EncryptedContentInfo.java	
	EncryptedData.java	Iduy
	EnvelopedData.java	Arif
	PKCS7.java	
	PKCS9Attribute.java	
	PKCS9Attributes.java	
	RecipientInfo.java	Arif

	SignerInfo.java	Sun
	ParsingException.java	
digsec.set.core.crypto	DetachedDigest.java	Arif
	E.java	
Keterangan:	EH.java	
Objek-objek operator	EK.java	Iduy
kriptografi SET	EX.java	Arif
	EXH.java	
	HMAC.java	Iduy
	Linkage.java	Arif
	OAEP.java	
	OAEPEncoder.java	
	S.java	Iduy
	SO.java	
digsec.set.core.encapsulation	Enc.java	Arif
	EncB.java	
Keterangan:	EncBX.java	
Objek-objek operator krip-	EncK.java	Iduy
tografi untuk enkapsulasi.	EncX.java	Arif
digsec.set.payment.component	AuthToken.java	Arif
	AuthTokenData.java	
Keterangan:	BackKeyData.java	Iduy
Objek-objek komponen pesan.	CountryCode.java	Arif
	CurrencyAmount.java	Iduy
	HOD.java	
	HODInput.java	
	HOIData.java	
	HPIData.java	
	Inputs.java	
	InstallRecurData.java	Arif
	InstallRecurInd.java	
	Location.java	
	MerTermIDs.java	
	OIData.java	Iduy
	PANData.java	Arif
	PANToken.java	
	PI.java	Iduy
	PIData.java	
	PIDataUnsigned.java	
	PIDualSigned.java	
	PIHead.java	
	PIOILink.java	

	PISignature.java	
	PITBS.java	
	PIUnsigned.java	
	Recurring.java	Arif
	RRTags.java	
	TransIDs.java	
digsec.set.payment.component.SaleDetail Keterangan: Objek-objek komponen pesan khusus untuk SaleDetail	AuthCharInd.java	Arif
	AutoApplicableRate.java	
	AutoCharges.java	
	AutoNoShow.java	
	AutoRateInfo.java	
	BatchId.java	
	BatchSequenceNum.java	
	ChargeInfo.java	
	CommercialCardData.java	
	DateTime.java	
	Distance.java	
	DistanceScale.java	
	HotelCharges.java	
	HotelNoShow.java	
	HotelRateInfo.java	
	Item.java	
	ItemSeq.java	
	MarketAutoCap.java	
	MarketHotelCap.java	
	MarketSpecCapData.java	
	MarketSpecDataID.java	
	MarketSpecSaleData.java	
	MarketTransportCap.java	
	MerOrderNum.java	
	PayRecurInd.java	
	Phone.java	
	Restrictions.java	
	SaleDetail.java	
	StopOverCode.java	
	TripLeg.java	
	TripLegSeq.java	
digsec.set.payment.message.AuthReqRes Keterangan: Objek-objek pesan otorisasi	AuthCode.java	Arif
	AuthHeader.java	
	AuthReq.java	
	AuthReqData.java	
	AuthReqItem.java	

	AuthReqPayload.java	
	AuthRes.java	
	AuthResBaggage.java	
	AuthResData.java	
	AuthResPayload.java	
	AuthTags.java	
	AuthValCodes.java	
	AVSData.java	
	BatchDetails.java	
	BatchStatus.java	
	BatchTotals.java	
	BrandBatchDetails.java	
	BrandBatchDetailsSeq.java	
	CapResPayload.java	
	CapToken.java	
	CapTokenData.java	
	CheckDigests.java	
	ClosedWhen.java	
	MarketSpecAuthData.java	
	MerchData.java	
	ResponseData.java	
digsec.set.payment.message.wrapper Keterangan: Objek-objek pembungkus pesan	Message.java	Iduy
	MessageHeader.java	
	MessageIDs.java	
	MessageWrapper.java	
digsec.set.payment.message.error Keterangan: Objek-objek pesan kesalahan	Error.java	Iduy
	ErrorCode.java	
	ErrorMsg.java	
	ErrorTBS.java	

Tabel IV-1 Tabel Objek-Objek Pesan SET

1.15 IMPLEMENTASI PROSES

Implementasi prototipe proses otorisasi antara *merchant* dan *payment gateway* dibagi dalam dua modul. Modul pertama adalah proses otorisasi di *merchant* dan modul kedua adalah proses otorisasi di *payment gateway.*

Sesuai dengan diagram alur data yang terdapat pada analisa proses di bab tiga, implementasi ini dilakukan. Sebelumnya perlu dijelaskan bahwa apa yang

diimplementasikan ini hanyalah prototipe saja. Adapun keterbatasan dari protipe proses ini adalah :

1. Tidak memiliki basis data transaksi. Data transaksi yang digunakan diasumsikan. sudah ada dan tersimpan dalam *file*.
2. Tidak memiliki modul pemrosesan sertifikat.
3. Tidak memiliki modul antar muka dengan pemakai.

Sesuai dengan DAD Proses Otorisasi pada sub-bab 1.11, terdapat dua bagian proses di *merchant,* yaitu pembuatan *authorization request* dan pemrosesan *authorization response*; dan dua bagian proses di *payment gateway*, yaitu pemrosesan *authorization request* dan pembuatan *authorization response.* Seluruh proses diimplementasikan berdasarkan DAD tersebut.

Pengiriman pesan antara *merchant* dan *payment gateway* menggunakan *socket* TCP/IP dengan memanfaatkan *library* yang sudah ada di Java.

Berikut ini adalah potongan program proses pembuatan AuthReq.

```
package digsec.set.application;

import java.io.*;
import java.net.*;
import java.util.*;
import sun.security.util.*;
import java.security.*;
import java.security.cert.*;
import digsec.set.payment.component.*;
import digsec.set.payment.component.SaleDetail.*;
import digsec.set.core.pkcs.*;
import digsec.set.core.crypto.*;
import digsec.set.payment.message.AuthReqRes.*;
import au.net.aba.crypto.provider.*;
import cryptix.provider.Cryptix;
import sun.misc.HexDumpEncoder;
import digsec.set.payment.wrapper.*;

public class Merchant {

        //------------------ network ----------------------------
        Socket                  mySock;
        DataInputStream         input;
        DataOutputStream        output;
        InetAddress             gatewayAddress;
        int                     port;

        //-----------Certicate and PrivateKey---------------------
        X509Certificate         m_cert;
```

```
X509Certificate          p_cert;
PrivateKey               m_priv;

//----------Cardholder Data-------------------------
AVSData                  avsData;
BigInt                   specialProcessing = new BigInt(0);
BigInt                   cardSuspect = new BigInt(0);
boolean                  requestCardTypeInd = false;

//----------Merchant Data---------------------------
byte[]                   lid_M = {2,1,0,3,1,7};
MerTermIDs               merTermIDs;
MerchData                merchData;
MarketSpecAuthData       marketSpecAuthData;

//---------- Earlier Transaction ----------
byte[]                   pi;
OIData                   oiData;
HODInput                 hodInput;
TransIDs                 transIDs;
BigInt                   authRetNum=null;
String                   paySysID = null;

//---------- Merchant's Options ----------
boolean                  subsequentAuthInd = false;
 boolean                 captureNow=true;

 //---------- Current Transaction ----------
CurrencyAmount           authReqAmt;
SaleDetail               saleDetail=null;

//----------flag----------
boolean                  splitRecurringPayment=false;
```

Gambar IV-8 *Field-field* pada AuthReq

Pada potongan program di atas terlihat sejumlah *field* yang sesuai dengan basis data yang dilambangkan pada DAD di bab tiga.

```
public MerchantP (InetAddress addr, int port) throws Exception {
        Security.addProvider(new ABAProvider());
        Security.addProvider(new Cryptix());

        gatewayAddress = addr;
        this.port = port;
```

```
            initPrivateKeyCert();
            initCardholderData();
            initMerchantData();
            initPreliminaryTransaction();
            initCurrentTransaction();
            initConnection();
            run();
    }

    private void initCurrentTransaction () throws Exception {
            authReqAmt = hodInput.purchAmt;
    }

    private void initPreliminaryTransaction () throws Exception {
            FileInputStream fi = new FileInputStream("HODInput.dat");
            byte[] input = new byte[fi.available()];
            fi.read(input);
            hodInput = new HODInput(new DerInputStream(input));

            fi = new FileInputStream("OIData.dat");
            input = new byte[fi.available()];
            fi.read(input);
            oiData = new OIData(new DerInputStream(input));

            fi = new FileInputStream("PIDualSigned.dat");
            pi = new byte[fi.available()];
            fi.read(pi);

            transIDs = oiData.transIDs;

    }

    private void initMerchantData() throws Exception {
            FileInputStream fi = new FileInputStream("MerTermIDs.dat");
            byte[] input = new byte[fi.available()];
            fi.read(input);
            fi.close();
            merTermIDs = new MerTermIDs(new DerInputStream(input));

            fi = new FileInputStream("MerchData.dat");
            input = new byte[fi.available()];
            fi.read(input);
            fi.close();
            merchData = new MerchData(new DerInputStream(input));

            marketSpecAuthData = null;
    }
```

```
private void initCardholderData() throws Exception {
        FileInputStream fi = new FileInputStream(
                        "d:\\digsec\\progie\\arif\\source\\digsec\\set\\application\\avsData.dat");
        byte[] input = new byte[fi.available()];
        fi.read(input);
        fi.close();
        avsData = new AVSData(new DerInputStream(input));
}

private void initPrivateKeyCert() throws Exception {
        //get payment-gateway's certificate
        FileInputStream inStream = new   FileInputStream(
                        "d:\\digsec\\progie\\arif\\source\\digsec\\set\\application\\keystore\\pg.cert");
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        p_cert = (X509Certificate)cf.generateCertificate(inStream);
        inStream.close();

        //get merchant's certificate
        inStream = new FileInputStream(
                        "d:\\digsec\\progie\\arif\\source\\digsec\\set\\application\\keystore\\m.cert");
        cf = CertificateFactory.getInstance("X.509");
        m_cert = (X509Certificate)cf.generateCertificate(inStream);
        inStream.close();

        //get merchant's private key
        inStream = new FileInputStream(
                        "d:\\digsec\\progie\\arif\\source\\digsec\\set\\application\\keystore\\m.jks");
        java.security.KeyStore ks = java.security.KeyStore.getInstance("JKS", "SUN");
        ks.load(inStream,"Merchant".toCharArray());
        m_priv = (PrivateKey) ks.getKey("M", "Merchant".toCharArray());
}

private v oid initConnection() throws IOException {
        mySock = new Socket(gatewayAddress,port);
        input = new DataInputStream(mySock.getInputStream());
        output = new DataOutputStream(mySock.getOutputStream());
}
```

Gambar IV-9 Inisialisasi Data-Data yang Sebelumnya Diterima dari Cardholder

Potongan program di atas adalah tahap inisialisi, data-data yang diasumsikan diterima dari *cardholder* sebelumnya diambil dari file-file yang sudah dipersiapkan terlebih dahulu.

```
public void run() throws Exception {
        Message msg = makeAuthReq();
        output.write(composeWrapper(msg));
    }

    private byte[] composeWrapper(Message msg) throws Exception {
        MessageIDs messageIDs = new MessageIDs(transIDs.lid_C,
                                               transIDs.lid_M,
                                               transIDs.xid);
        MessageHeader mh = new MessageHeader(new Date(), messageIDs,
                                             oiData.rrpid, "DIGSEC");
        MessageWrapper mw = new MessageWrapper(mh,msg);
        DerOutputStream dout = new DerOutputStream();
        mw.encode(dout);
        return dout.toByteArray();
    }

    private AuthTags makeAuthTags() throws Exception {
        RRTags rrTags = new RRTags(oiData.rrpid, merTermIDs);
        return new AuthTags(rrTags,transIDs,authRetNum);
    }

    private AuthReq makeAuthReq() throws Exception {
        //->->->->->->->->->-> AuthReqItem <-<-<-<-<-<-<-<-<-<-
        //make AuthTags
        AuthTags authTags = makeAuthTags();
        //make AuthReqPayload
        AuthReqPayload authReqPayload = new AuthReqPayload(subsequentAuthInd,
                                                           authReqAmt,
                                                           avsData,
                                                           specialProcessing,
                                                           cardSuspect,
                                                           requestCardTypeInd,
                                                           hodInput.installRecurData,
                                                           marketSpecAuthData,
                                                           merchData);
        //make CheckDigest
        HOD hod2 = new HOD(hodInput);
        HOIData hoiData = new HOIData(oiData);
        CheckDigests checkDigest = new CheckDigests(hoiData,hod2);
        //make AuthReqItem
        AuthReqItem authReqItem = new AuthReqItem(authTags,
                                                  checkDigest,
                                                  authReqPayload);

        //->->->->->->->->->-> Make SaleDetail <-<-<-<-<-<-<-<-<-<-
        //->->->->->->->->->-> Make AuthReq <-<-<-<-<-<-<-<-<-<-<
        AuthReqData authReqData = new AuthReqData(authReqItem,
```

```
                                                captureNow,
                                                saleDetail);
            AuthReq ar = new AuthReq(m_cert,m_priv,p_cert,authReqData,pi);
            return ar;
    }
```

Gambar IV-10 Proses-proses yang dibuat mendekati DAD Pembuatan AuthReq

Potongan program di atas dibuat sesuai dengan proses-proses yang ada pada proses pembuatan AuthReq.

BAB V
PENGUJIAN

1.16 SKENARIO PENGUJIAN

Pada pengujian ini, terdapat dua bagian skenario pengujian. Skenario pertama adalah pengujian objek pesan dan skenario kedua adalah pengujian proses.

1.16.1 Pengujian Objek Pesan

Secara umum setiap objek baik dari lapisan terbawah hingga teratas harus memiliki kemampuan untuk meng-*encode* dirinya ke dalam *byte-array* DER dan membentuk dirinya kembali dari *byte-array* DER tersebut. Tapi pada pengujian ini, cukup objek-objek pada lapisan atas (mulai dari lapisan komponen pesan) yang diuji. Pengujian untuk objek-objek lapisan bawah telah dilakukan selama tahap pengembangan. Daftar objek-objek yang diuji dapat dilihat pada sub-bab hasil pengujian.

1.16.2 Pengujian Proses

Pengujian proses dilakukan dengan memberikan kasus-kasus terhadap proses-proses otorisasi baik di *merchant* maupun di *payment gateway*. Kasus-kasus yang diberikan meliputi kasus normal dan kasus khusus. Kasus-kasus tersebut adalah:

1. *Payment gateway* menerima pesan AuthReq dengan kondisi normal (tidak terjadi kesalahan). Keadaan yang diharapkan dari kasus ini adalah *merchant* menerima AuthRes kembali dengan AuthCode “approve”.
2. *Payment gateway* menerima pesan AuthReq dengan bentuk *byte-array* DER yang salah. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan pesan kesalahan kepada *merchant* dengan kode “decoding failure”.
3. *Payment gateway* menemukan TransIDs pada AuthTags tidak sesuai dengan TransIDs pada PI. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “piAuthMismatch”.
4. Dalam kasus PI yang diterima *payment gateway* adalah AuthToken, Recurring.RecurringExpiry pada InstallRecurData telah lewat dari tanggal

sekarang. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “recurringExpired”.

5. Dalam kasus PI yang diterima *payment gateway* adalah AuthToken, tanggal sekarang belum melewati tanggal PrevAuthDate ditambah Recurring.RecurringFrequency. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “recurringTooSoon”.
6. Dalam kasus PI yang diterima *payment gateway* adalah PIDualSigned, PANData dalam parameter PI tidak sesuai dengan PANData dalam PI-TBS. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “piAuthMismatch”.
7. Dalam kasus PI yang diterima *payment gateway* adalah PIUnSigned, *Hash* dari PI-OILink pada blok OAEP tidak sesuai. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “piAuthMismatch”.
8. *Payment gateway* menerima PIHead, HOIData pada CheckDigests tidak sesuai dengan HOIData pada PIHead Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “piAuthMismatch”.
9. Bila *payment gateway* menerima PIHead, HOD2 pada CheckDigests tidak sesuai dengan HOD pada PIHead. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “piAuthMismatch”.
10. InstallRecurData yang terdapat dalam AuthReqPayload yang diterima *payment gateway* tidak sesuai dengan InstallRecurData yang terdapat dalam PIHead. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes kepada *merchant* dengan AuthCode “installRecurMismatch”.
11. *Payment gateway* menerima AuthReq yang menandakan bahwa *merchant* ingin langsung melakukan proses *capture* (captureNow), namun *payment gateway* tidak menyediakan jenis pelayanan tersebut. Keadaan yang diharapkan dari kasus ini adalah *payment gateway* mengirimkan AuthRes dengan AuthCode “captureNotSupport”.
12. *Merchant* Menerima pesan AuthRes dengan bentuk *byte-array* yang salah. Keadaan yang diharapkan dari kasus ini adalah *merchant* mengirimkan pesan kesalahan kepada *payment gateway* dengan kode “decoding failure”.

13. AuthTags.XID yang diterima *merchant* tidak sesuai dengan AuthReq yang sebelumnya dikirimkan oleh *merchant.* Keadaan yang diharapkan dari kasus ini adalah *merchant* mencatat pada log kesalahan dengan kode “unknown XID”.
14. AuthTags.LID-C atau AuthTags.LID-M tidak sesuai dengan AuthReq yang sebelumnya dikirimkan oleh *merchant.* Keadaan yang diharapkan dari kasus ini adalah *merchant* mencatat pada log kesalahan dengan kode “unknown LID”.

1.17 HASIL PENGUJIAN

1.17.1 Pengujian Objek Pesan Umum

Paket	Objek Penyusun	Uji *Encode-Decode*
digsec.set.payment.component	AuthToken.java	√
	AuthTokenData.java	√
	BackKeyData.java	❖
	HOD.java	❖
	HODInput.java	❖
	HOIData.java	❖
	HPIData.java	❖
	Inputs.java	❖
	InstallRecurData.java	√
	InstallRecurInd.java	√
	MerTermIDs.java	√
	OIData.java	❖
	PANData.java	√
	PANToken.java	√
	PI.java	❖
	PIData.java	❖
	PIDataUnsigned.java	❖
	PIDualSigned.java	❖
	PIHead.java	❖
	PIOILink.java	❖
	PISignature.java	❖
	PITBS.java	❖
	PIUnsigned.java	❖
	Recurring.java	√

	RRTags.java	√
	TransIDs.java	√
digsec.set.payment.component.SaleDetail	AuthCharInd.java	√
	AutoApplicableRate.java	√
	AutoCharges.java	√
	AutoNoShow.java	√
	AutoRateInfo.java	√
	BatchId.java	√
	BatchSequenceNum.java	√
	ChargeInfo.java	√
	CommercialCardData.java	√
	DateTime.java	√
	Distance.java	√
	DistanceScale.java	√
	HotelCharges.java	√
	HotelNoShow.java	√
	HotelRateInfo.java	√
	Item.java	√
	ItemSeq.java	√
	MarketAutoCap.java	√
	MarketHotelCap.java	√
	MarketSpecCapData.java	√
	MarketSpecDataID.java	√
	MarketSpecSaleData.java	√
	MarketTransportCap.java	√
	MerOrderNum.java	√
	PayRecurInd.java	√
	Phone.java	√
	Restrictions.java	√
	SaleDetail.java	√
	StopOverCode.java	√
	TripLeg.java	√
	TripLegSeq.java	√
digsec.set.payment.message.AuthReqRes	AuthCode.java	√
	AuthHeader.java	√
	AuthReq.java	√
	AuthReqData.java	√
	AuthReqItem.java	√

	AuthReqPayload.java	√
	AuthRes.java	√
	AuthResBaggage.java	√
	AuthResData.java	√
	AuthResPayload.java	√
	AuthTags.java	√
	AuthValCodes.java	√
	AVSData.java	√
	BatchDetails.java	√
	BatchStatus.java	√
	BatchTotals.java	√
	BrandBatchDetails.java	√
	BrandBatchDetailsSeq.java	√
	CapResPayload.java	√
	CapToken.java	√
	CapTokenData.java	√
	CheckDigests.java	√
	ClosedWhen.java	√
	MarketSpecAuthData.java	√
	MerchData.java	√
	ResponseData.java	√
digsec.set.payment. message.wrapper	Message.java	√
	MessageHeader.java	√
	MessageIDs.java	√
	MessageWrapper.java	√
digsec.set.payment. message.error.	Error.java	√
	ErrorCode.java	√
	ErrorMsg.java	√
	ErrorTBS.java	√

Tabel V-1 Hasil Pengujian Objek Pesan

Keterangan:

- Tanda di sebelah kiri ini pada tabel menandakan pengujian dilakukan oleh Dwinanda Prayudi.

1.17.2 Pengujian Proses

Kasus	Kesesuaian
1	√
2	√
3	√
4	√
5	√
6	√
7	√
8	√
9	√
10	√
11	√
12	√
13	√
14	√
15	√

Tabel V-2 Hasil Pengujian Proses

BAB VI
KESIMPULAN DAN SARAN

1.18 KESIMPULAN

Dari penelitian yang dilakukan oleh penulis ini dapat ditarik beberapa kesimpulan, yaitu:

1. Proses otorisasi kartu pembayaran antara *merchant* dan *payment gateway* dapat diimplementasikan.
2. Pemahaman terhadap ASN.1 dan DER diperlukan untuk mengerti spesifikasi SET.
3. Untuk dapat mengimplementasikan proses otorisasi ini juga diperlukan objek-objek kriptografi dan objek-objek penyusun pesan SET dimana setiap objek memiliki fasilitas untuk mengkodekan dirinya kedalam kumpulan *byte* DER dan menerjemahkan *byte* DER tersebut menjadi objek itu sendiri.
4. Implementasi dapat dilakukan dengan bahasa pemrograman Java dengan resiko proses berjalan lambat, namun kemudahan bagi pemrogramnya.
5. Proses otorisasi kartu pembayaran antara *merchant* dan *payment gateway* dapat memenuhi isu autentisitas, kerahasiaan dan integritas penyampaian data transaksi dengan kombinasi teknik kriptografi dan kombinasi informasi transaksi yang diatur sedimikian rupa untuk menjamin keamanannya.

1.19 SARAN PENGEMBANGAN

Prototipe proses otorisasi antara *merchant* dan *payment gateway* ini dapat dikembangkan lagi dengan cara:

1. Menambah modul basis data pada server *merchant* dan *payment gateway* yang berhubungan dengan transaksi SET ini.
2. Menembah modul pemrosesan sertifikat yang berhubungan dengan CA yang ada.
3. Integrasi dengan sistem *merchant* pada sisi yang berhubungan dengan *cardholder*.

LAMPIRAN A
KODE OTORISASI

Berikut ini adalah tabel kode otorisasi (AuthCode) yang dikembalikan kepada *merchant* oleh *payment gateway* melalui *authorization response* (AuthRes).

approved	Permintaan otorisasi telah disetujui.
unspecifiedFailure	Permintaan otorisasi tidak dapat diproses dengan alasan yang tidak didefinisikan dalam tabel ini.
declined	Permintaan otorisasi ditolak.
noReply	*Issuer* tidak menjawab permintaan otorisasi ini. Nilai ini sering menandakan fasilitas pemrosesan data *issuer* tidak berfungsi sementara.
callIssuer	*Issuer* meminta panggilan telepon oleh *merchant*.
amountError	Nilai transaksi tidak dapat diproses oleh sistem di lapisan atas (*acquirer,* jaringan perbankan, *issuer*).
expiredCard	Kartu pembayaran sudah kadaluarsa.
invalidTransaction	Permintaan otorisasi tidak dapat diproses oleh sistem di lapisan atas (*acquirer,* jaringan perbankan, *issuer*) karena tipe transaksi tidak diperbolehkan.
systemError	Permintaan otorisasi tidak dapat diproses oleh sistem di lapisan atas (*acquirer,* jaringan perbankan, *issuer*) karena data pada permintaan tidak sesuai.
piPreviouslyUsed	*Payment Instructions* pada permintaan otorisasi telah dipergunakan untuk permintaan otorisasi sebelumnya.
recurringTooSoon	Waktu minimum antara dua permintaan otorisasi belum dilalui untuk transaksi berlangganan.
recurringExpired	Tanggal kadaluarsa untuk transaksi berlangganan telah lewat.
piAuthMismatch	Data pada PI dari *cardholder* tidak berhubungan dengan data pada OD dari *merchant*.
installRecur Mismatch	InstallRecurData pada PI dari *cardholder* tidak berhubungan dengan InstallRecurData pada OD dari *merchant*.
captureNotSupported	*Payment gateway* tidak menyediakan pelayanan untuk *capture*.
signatureRequired	PIUnsigned tidak dapat diterima oleh *payment gateway* untuk merek kartu pembayaran ini.
cardMerchBrand Mismatch	Merek kartu pembayaran yang tertera pada sertifikat tanda tangan *cardholder* tidak cocok dengan merek yang terdapat pada sertifikat pertukaran pesan milik *payment gateway.*

DAFTAR PUSTAKA

[Kali93] Burton S. Kaliski Jr.: *A Layman's Guide to a Subset of ASN.1, BER, and DER.;* RSA Laboratories, 1993.

[Nua99] Nua Internet Surveys: *32 Million Household to Bank Online by 2003*; Nua Ltd., Irlandia, 1999

[PKCS #7] RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard.* Version 1.5, November 1993.

[PKCS #9] RSA Laboratories. *PKCS #9: Selected Attribute Types.* Version 1.1, November 1993.

[Rfc1766] H. Alvestrand: *Tags for the Identification of Languages*, Request for Comments: 1766; UNINNET, Maret 1995.

[Schn96] Bruce Schneier: *Applied Cryptography, 2nd ed.;* John Wiley & Sons, Inc., New York 1996.

[Set97a] Visa dan Mastercard: *SET Secure Electronic Transaction Specification, Book 1: Business Description*; 1997.

[Set97b] Visa dan Mastercard: *SET Secure Electronic Transaction Specification, Book 2: Programmer's Guide*; 1997.

[Set97c] Visa dan Mastercard: *SET Secure Electronic Transaction Specification, Book 3: Formal Protocol Definition*; 1997.

[Setco99] SETCo: Hompage, 1999. http://www.setco.org