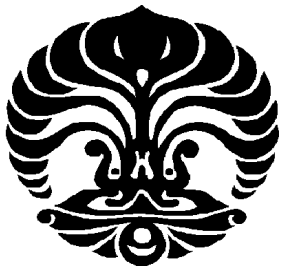
SMARTWALLET – JAVA WALLET BERBASIS SMARTCARD DAN PROTOKOL SET

Disusun sebagai Laporan Tugas Akhir

Oleh :

Iman Budi Setiawan - 1295000261



Fakultas Ilmu Komputer

Universitas Indonesia

Depok

1999

LEMBAR PERSETUJUAN

SKRIPSI : SMARTWALLET – JAVA WALLET BERBASIS SMARTCARD DAN PROTOKOL SECURE ELECTRONIC TRANSACTION (SET).

NAMA : IMAN BUDI SETIAWAN

NPM : 1295000261

SKRIPSI INI TELAH DIPERIKSA DAN DISETUJUI

DEPOK, …….

Menyetujui,

_____

Pembimbing Tugas Akhir

KATA PENGANTAR

Puji dan syukur saya panjatkan kepada Allah Yang Maha Kuasa, karena berkat rahmat dan karunia-Nya sehingga saya dapat menyelesaikan tugas akhir dengan judul "*SmartWallet – Java Wallet Berbasis SmartCard dan Protokol SET*". Tugas akhir ini merupakan salah satu syarat untuk memperoleh kelulusan untuk mahasiswa Fakultas Ilmu Komputer UI pada jenjang pendidikan tingkat S1.

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

1. Ibu dan bapak, yang telah mendidik dan membesarkan saya dengan penuh pengorbanan dan kasih sayang.
2. Bapak Nursalim Hadi PH.d, selaku pembimbing awal tugas akhir yang telah memberikan arahan dalam mengerjakan tugas akhir ini.
3. Bapak Bob Hardian M.Kom, selaku pembimbing tugas akhir yang banyak memberikan arahan dalam mengerjakan tugas akhir ini.
4. Arianto Mukti Wibowo SKom, selaku asisten pembimbing tugas akhir yang telah membimbing saya dalam mengerjakan tugas akhir ini.
5. Bapak Machmudin Junus PH.d, selaku pembimbing akademis sejak saya masuk dan menuntut ilmu di Fakultas Ilmu Komputer UI.
6. Bapak T. Basaruddin PH.d, selaku pengganti pembimbing akademis pada akhir masa perkuliahan saya.
7. Segenap staf, dosen, karyawan dan rekan-rekan mahasiswa baik dilingkungan Fasilkom UI maupun lingkungan UI yang tidak dapat saya sebutkan satu persatu, atas segala sumbangsih, perhatian dan dukungannya.

Saya 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

Internet yang berkembang dengan pesat kini telah dimanfaatkan orang untuk melakukan transaksi perdagangan. Transaksi yang dilakukan harus dapat menjamin keamanan data-data yang dipertukarkan antar pihak-pihak yang berkepentingan. Masalah-masalah umum yang ada pada Sistem Perdagangan di Internet (SPI) antara lain kerahasiaan pesan (*confidentiality*), keutuhan pesan (*integrity*), keabsahan pesan (*authenticity*), dan keaslian pesan (*non repudiation*).

Enkripsi data merupakan solusi yang tepat untuk melindungi data dari usaha-usaha pencurian dan pemalsuan data. Fasilitas enkripsi di Internet yang banyak digunakan orang saat ini adalah *Secure Socket Layer* (SSL) yang didukung oleh dua *browser* terkemuka yaitu Microsoft Internet Explorer dan Netscape Navigator. Namun pada kenyataannya, SSL mempunyai kelemahan yang membuatnya kurang potensial untuk digunakan dalam transaksi perdagangan. Oleh karena itu, Visa dan Mastercard mengeluarkan protokol standar yang aman untuk transaksi perdagangan yang disebut *Secure Electronic Transaction* (SET).

Tugas akhir ini mengimplementasikan SmartWallet, yaitu Java Wallet berbasis SmartCard dan protokol SET. SmartWallet adalah suatu aplikasi berbasis Java Applet yang digunakan untuk melakukan transaksi perdagangan di Internet. SmartWallet digunakan sebagai aplikasi utama untuk melakukan penyimpanan data, autentikasi, permintaan sertifikat ke *Certificate Authority* (CA), dan transaksi dengan pedagang (*merchant*). Smartcard digunakan konsumen sebagai tempat penyimpanan dan autentikasi data saat melakukan transaksi dengan pedagang. Semua proses di atas menggunakan protokol SET untuk menjamin keamanan data-data selama transaksi.

xiv + 1 hlm.; 55 gbr.; 8 tbl.;
Bilbiografi: 28 (1994-1999)

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Bab ini membahas tentang latar belakang masalah, tujuan penelitian, pembatasan masalah, metodologi penelitian, serta sistematika penulisan dari penelitian tugas akhir ini.

1.1 LATAR BELAKANG MASALAH

Perkembangan Internet yang demikian pesat telah dimanfaatkan orang sebagai sarana untuk melakukan transaksi perdagangan. Transaksi yang dilakukan secara *online* melalui Internet ini biasa disebut sebagai *Internet Commerce*. Transaksi tersebut dilakukan melalui jaringan publik sehingga masalah terbesar terletak pada sistem keamanannya. Masalah-masalah tersebut antara lain kerahasiaan pesan (*confidentiality*), keutuhan pesan (*integrity*), keabsahan pesan (*authenticity*), dan keaslian pesan (*non repudiation*). Masalah-masalah ini menyebabkan banyak pedagang yang khawatir untuk memanfaatkan *Internet Commerce* dalam sistem perdagangan mereka.

Transaksi *online* yang digunakan pertama kali hanya berbasis form html biasa. Saat itu Internet hanya digunakan untuk menjual barang-barang dagangan saja sedangkan pembeliannya dapat dilakukan melalui *mail order / telephone order* (MOTO). Transaksi seperti ini tidak efektif karena pembeli tidak dapat memesan secara langsung barang-barang yang akan dibelinya. Kemudian orang mulai mencoba menggunakan Internet untuk melakukan pembelian secara *online*. Seorang pembeli dapat melakukan transaksi dengan memilih barang yang akan dibelinya kemudian memasukkan nomor kartu kredit miliknya untuk di-*charge* oleh pedagang sebagai pembayarannya. Namun cara seperti ini tidak cukup aman untuk transaksi yang penting dan membutuhkan kerahasiaan data.

Dua *browser* terkemuka yaitu Netscape Navigator dan Microsoft Internet Explorer kemudian mendukung teknologi *SSL (Secure Socket Layer*) yang bertujuan untuk menjamin keamanan transaksi. Protokol SSL diimplementasikan pada level *transport* dan digunakan secara Internasional menggunakan teknologi enkripsi 40-bit dari RSA. Hal ini berdasarkan undang-undang regulasi Amerika Serikat, yaitu hanya melakukan ekspor teknologi enkripsi sebatas 40-bit, sedangkan untuk domestik boleh sampai 128-bit. SSL yang memanfaatkan teknologi kunci publik 40-bit dari RSA ternyata dapat dijebol dalam waktu 1,3 hari dengan 100 komputer

menggunakan *brute-force attack* [DHMM96]. Ini tidak berarti teknologi SSL tidak bermanfaat. Tujuannya jelas, yakni agar biaya yang dikeluarkan oleh pihak penyerang lebih besar daripada 'harga' informasi informasi yang dienkripsi melalui SSL, sehingga secara ekonomi tidak menguntungkan. Sedangkan kunci enkripsi simetris yang digunakan adalah 128-bit (40-bit dienkripsi dan 80-bit tidak dienkripsi saat kunci dipertukarkan). Dengan *clear-text attack*, masih sangat sulit untuk memecahkannya. Kelemahan utama dari protokol SSL ini sebenarnya terletak pada informasi kartu kredit. Seorang pedagang dapat mengetahui informasi kartu kredit milik konsumennya sehingga ia dengan mudah dapat menggunakan informasi kartu kredit tersebut untuk kepentingannya.

Visa dan MasterCard mengeluarkan protokol standar yang disebut sebagai SET (*Secure Electronic Transaction*) untuk mengatasi kelemahan-kelemahan yang dimiliki protokol SSL. Protokol SET menjamin keamanan pengiriman informasi pemesan dan pembayaran serta integritas data dalam setiap transaksi. Pada protokol ini, seorang konsumen dapat diautentikasi sebagai pemegang kartu yang sah pada perusahaan pembayaran tertentu dan seorang pedagang memang benar-benar bisa menerima jenis pembayaran tersebut. Disamping itu, sistem pembayarannya tidak terikat kepada suatu protokol perangkat keras atau perangkat lunak tertentu, dengan kata lain dapat bekerja dengan berbagai macam perangkat lunak dan berbagai penyedia jasa. Saat ini sudah banyak vendor yang menyatakan kesediaannya untuk menerapkan protokol SET pada sistem perdagangannya, antara lain Microsoft, IBM, Netscape, GTE, Open Market, CyberCash, Terisa System, dan Verisign. Bahkan perusahaan penyelenggara *charge card* seperti American Express akhirnya menyatakan dukungannya untuk SET. Kelebihan-kelebihan yang ditawarkan oleh protokol SET menyebabkan penulis menggunakannya dalam implementasi ini.

Protokol SET tidak diimplementasikan pad*a level transpo*rt seperti protokol sebelumnya yaitu protokol SSL tetapi pada level aplikasi. Kedua protokol ini membutuhkan aplikasi yang dapat digunakan untuk melakukan autentikasi dan *customize* data-data transaksi. Aplikasi yang digunakan untuk melakukan tugas ini yang selanjutnya disebut *wallet*. *Browser* akan membuka sebuah aplikasi *wallet* kemudian mengirimkan data-data transaksi ke *wallet* tersebut. *Wallet* inilah yang digunakan oleh konsumen untuk mengatur data-data transaksi yang akan dikirim ke pedagang. Aplikasi *wallet* telah banyak dikembangkan oleh vendor-vendor terkemuka di dunia, misalnya Microsoft Wallet, vWallet oleh Verisign, MasterCard Wallet, dan Java Wallet oleh Sun Microsystem. Implementasi yang dilakukan adalah SmartWallet, yaitu Java Wallet yang berbasis SmartCard dan protokol SET. *Wallet* ini akan disimpan di sebuah *web server* yang terletak di lokasi pedagang, dan konsumen dapat menggunakannya melalui *browser* dari alamatnya masing-masing.

Masalah lain dalam melakukan transaksi adalah autentikasi data transaksi. Konsumen harus dapat diidentifikasikan sebagai konsumen yang sah, demikian juga halnya dengan pedagang. Autentikasi ini dilakukan dengan menggunakan teknologi kunci publik dan menggunakan jasa *Certificate Authority* (CA) untuk menyebarkan kunci publik tersebut. Selain itu, konsumen dan pedagang harus menggunakan identitasnya untuk melakukan transaksi. Identitas ini dapat disimpan dalam media penyimpanan masing-masing. Media penyimpanan yang umumnya digunakan saat ini adalah hardisk, tetapi hardisk memiliki banyak kekurangan sehingga penulis juga menggunakan *smartcard* sebagai alternatif lain untuk media penyimpanan.

1.2 TUJUAN PENELITIAN

Tujuan utama penelitian tugas akhir ini adalah mengimplementasikan SmartWallet, yaitu Java Wallet berbasis Smartcard dan protokol SET. Secara lebih spesifik, tujuan penelitian adalah:

- Melakukan studi perbandingan terhadap teknologi *wallet* yang ada saat ini.
- Mengimplementasikan sebuah *wallet* berbasis Java Applet yang akan digunakan dalam transaksi perdagangan.
- Mengimplementasikan sebuah *website* yang menggunakan aplikasi *wallet* yang telah diimplementasi tersebut.
- Menggunakan protokol SET pada implementasi *wallet* tersebut.
- Penyimpanan dan autentikasi data di *wallet* menggunakan *hardisk* sebagaimana wallet-wallet yang sudah ada.
- Penyimpanan dan autentikasi data di *wallet* menggunakan *smartcard* sebagai alternatif lain yang lebih aman.

1.3 PEMBATASAN MASALAH

Implementasi yang dilakukan pada penelitian tugas akhir ini menggunakan protokol SET dan algoritma enkripsi yang sudah ada. Disamping itu, identitas pribadi yang disimpan dalam *smartcard* cukup data-data yang penting untuk autentikasi saja, mengingat ruang penyimpanan *smartcard* saat ini yang relatif kecil. Untuk memperlihatkan penggunaan SmartWallet maka diimplementasi juga sebuah *website* untuk pemesanan kamar hotel secara *online*.

1.4 METODOLOGI PENELITIAN

Penelitian dimulai dengan pembahasan mengenai analisis kebutuhan umum suatu transaksi perdagangan. Kemudian mencoba melakukan analisis lebih lanjut mengenai kemampuan sistem yang akan diimplementasikan dalam memenuhi kebutuhan-kebutuhan tersebut serta melihat keunggulannya dibandingkan sistem perdagangan yang ada saat ini.

Penelitian dilanjutkan dengan mempelajari protokol SET (Secure Electronic Transaction) yang merupakan protokol standar perdagangan yang dikeluarkan oleh Visa dan MasterCard.

Kemudian melakukan analisis mengenai penggunaan *smartcard* sebagai tempat penyimpanan informasi dan media autentikasi. Disamping itu, juga mempelajari tentang ISO/IEC-7816, *Public Key Cryptography Standard* (PKCS), *PC/SC*, *Open Card*, *Open Visa*, *Java Card*, dan literatur-literatur lain yang bermanfaat. Selanjutnya, untuk lebih memahami tentang penggunaan *smartcard*, penulis melakukan studi ke perusahaan-perusahaan asing yang memproduksi *smartcard* yaitu Gemplus, Siemens Nixdorf, dan Schlumberger.

Penelitian selanjutnya mempelajari pemrograman pada *smartcard* dengan mengikuti standar ISO/IEC-7816. Kemudian dipelajari juga teori-teori kriptografi dan teknik-teknik pemrograman pada java yang digunakan dalam implementasi ini.

Kemudian melakukan desain dari sistem yang akan diimplementasikan. Pada desain ini, *smartcard* mempunyai peran yang sangat penting dan digunakan sebagai media autentikasi. Tiap proses transaksi diperlihatkan sejelas mungkin untuk memperlihatkan alur proses dari transaksi perdagangan tersebut.

Setelah melakukan desain dari sistem yang akan dibuat, maka dimulailah tahap implementasi yang mengacu pada tahap desain. Pada tahap ini, diimplementasikan sebuah Java Wallet yang berbasis applet, menggunakan smartcard sebagai media autentikasi, dan protokol SET sebagai protokol komunikasi. Kemudian melakukan implementasi sistem pemesanan fasilitas hotel secara *online* melalui Internet. Tahap selanjutnya yaitu melakukan pengujian terhadap implementasi yang dilakukan kemudian mengambil kesimpulan dari sistem perdagangan tersebut.

1.5 SISTEMATIKA PENULISAN

Bab pertama diawali dengan penjelasan mengenai latar belakang masalah, kemudian disambung dengan tujuan penelitian, ruang lingkup permasalahan, metodologi penelitian, dan

sistematika penulisan.

Bab 2 membahas lebih dalam mengenai analisis kebutuhan sistem perdagangan secara umum serta analisis mengenai syarat-syarat tambahan yang harus dimiliki oleh suatu SPI. Setelah itu dianalisis kebutuhan-kebutuhan yang belum dipenuhi oleh sistem perdagangan yang sudah ada saat ini.

Selanjutnya Bab 3 membahas landasan teori yang dapat membantu pembaca dalam memahami implementasi yang dilakukan. Landasan teori tersebut mencakup tentang *Secure Socket Laye*r (SSL), *Secure Electronic Transaction* (SET), kriptografi, *smartcard*, dan *Java Native Interface* (JNI).

Bab 4 membahas tentang perbandingan teknologi yang digunakan pada aplikasi *wallet* yang ada saat ini.

Selanjutnya Bab 5 membahas mengenai perancangan dari sistem yang diimplementasi, disini diperlihatkan alur proses transaksi dari awal permintaan sampai selesainya transaksi tersebut serta struktur penyimpanan data dan proses yang terjadi pada *smartcard*.

Setelah itu Bab 6 membahas tentang implementasi yang dilakukan, dengan sistem perdagangan *bank card* tersebut diharapkan dapat memberi keamanan yang lebih terjamin. Implementasi ini menggunakan smartcard sebagai media autentikasi dan protokol SET sebagai media komunikasi pada transaksi perdagangan.

Kemudian Bab 7 menjelaskan pengujian terhadap sistem yang sudah diimplementasikan sebelumnya. Tahap ini memperlihatkan proses transaksi dari awal permintaan sampai selesainya transaksi, selain itu juga menganalisis keamanan dari transaksi yang sedang dilakukan tersebut.

Bab 8 sebagai bab terakhir diisi dengan kesimpulan dan saran dari penelitian tugas akhir ini.

BAB II
ANALISIS KEBUTUHAN

Bab ini membahas mengenai analisis kebutuhan dari sistem perdagangan yang diimplementasi. Analisis ini dimulai dengan menganalisis kebutuhan umum dari Sistem Perdagangan di Internet (SPI) saat ini. Kemudian dilakukan analisis mengenai syarat-syarat tambahan yang harus dimiliki oleh suatu SPI. Setelah itu dianalisis kebutuhan-kebutuhan yang ada berdasarkan kinerja dari sistem perdagangan yang banyak digunakan saat ini. Analisis-analisis tersebut menjadi dasar dalam melakukan implementasi selanjutnya.

2.1. KEBUTUHAN UMUM

Secara umum, suatu transaksi perdagangan diharapkan dapat menjamin :

- Kerahasiaan (*confidentiality*), yaitu data transaksi harus dapat disampaikan secara rahasia, sehingga tidak dapat dibaca oleh pihak-pihak yang tidak diinginkan.
- Keutuhan (*integrity*): Data setiap transaksi tidak boleh berubah saat disampaikan melalui suatu saluran komunikasi.
- Keabsahan atau keotentikan (*authenticity*), meliputi:
  - Keabsahan pihak-pihak yang melakukan transaksi: Bahwa sang konsumen adalah seorang pelanggan yang sah pada suatu perusahaan penyelenggara sistem pembayaran tertentu (misalnya kartu kredit Visa dan MasterCard, atau kartu debit seperti Kualiva dan StarCard) dan keabsahan keberadaan pedagang itu sendiri.
  - Keabsahan data transaksi: Data transaksi itu oleh penerima diyakini dibuat oleh pihak yang mengaku membuatnya (biasanya sang pembuat data tersebut membubuhkan tanda tangannya). Hal ini termasuk pula jaminan bahwa tanda tangan dalam dokumen tersebut tidak bisa dipalsukan atau diubah.
- Dapat dijadikan bukti / tak dapat disangkal (*non-repudiation*): catatan mengenai transaksi yang telah dilakukan dapat dijadikan barang bukti di suatu saat jika ada perselisihan.

2.2. BEBERAPA SYARAT TAMBAHAN UNTUK SPI

Ada beberapa syarat lagi dari sistem perdagangan di Internet [ARRI97] – meskipun tidak mutlak – hal ini sangat tergantung dari berbagai faktor lain yang ikut dipertimbangkan oleh

perusahaan pengembang SPI, sehingga dapat berbeda-beda untuk setiap SPI:

1. Jika menggunakan protokol HTTP, dapat berjalan dengan baik pada *web browser* yang populer seperti Netscape Navigator atau Microsoft Internet Explorer.
2. *Open*, artinya perangkat lunak ataupun perangkat keras sistem perdagangan di Internet tersebut tidak dibuat hanya untuk kepentingan satu pedagang saja, namun sistem perdagangan di Internet tersebut dapat dipergunakan oleh berbagai pedagang.
3. Pada SPI yang membutuhkan perangkat lunak klien khusus, perangkat lunak tersebut harus dibuat pada banyak *platform* agar pemakaiannya dapat meluas.
4. Sistem perdagangan di Internet tersebut sebaiknya dapat menerima sebanyak mungkin cara pembayaran.
5. Jika proses pembayaran dilakukan secara interaktif (yang tidak/kurang interaktif biasanya menggunakan surat elektronik), proses pengolahan data secara aman tersebut dapat berlangsung dalam waktu yang dapat ditolerir. Jika proses untuk melakukan pengamanan data transaksi yang dikirim saja sudah sangat lama, mungkin metoda yang diterapkan tidak cocok.
6. Ada beberapa SPI yang berusaha untuk tidak terikat pada sebuah protokol perangkat lunak maupun perangkat keras tertentu. Memang pada akhirnya harus ada implementasi lebih kongkrit yang harus dijabarkan. Di sisi lain, ada pula beberapa SPI yang memang dirancang sedemikian rupa untuk memanfaatkan protokol-protokal yang sudah ada.
7. Dalam kasus tertentu, transaksi diharapkan supaya anonim dan tidak dapat dilacak. Sedangkan pada kasus lainnya, justru transaksi tersebut diharapkan jati diri pihak-pihak yang bertransaksi dapat diidentifikasi dan dapat dilacak dengan baik.

2.3. ANALISIS SISTEM PERDAGANGAN SAAT INI

Saat ini sudah banyak sistem perdagangan yang telah diimplementasi dan sebagian besar sudah digunakan dalam transaksi perdagangan, misalnya CyberTrade, Open Market, Cybercash, First Virtual, NetChex dan Digicash Ecash. Sistem perdagangan tersebut masing-masing mempunyai kelebihan dan kekurangan terutama pada masalah keamanan dalam melakukan transaksi [ARRI97].

Sistem Perdagangan di Internet (SPI) saat ini sudah ada yang menggunakan aplikasi *wallet* untuk melakukan transaksi. Sementara itu, SPI-SPI lain hanya mengimplementasikan toko elektronik sederhana yang menggunakan form HTML. Aplikasi *wallet* yang cukup terkenal saat ini antara lain Microsoft Wallet yang dikeluarkan oleh Microsoft dan Java Wallet yang

dikeluarkan oleh Sun Microsystem. Aplikasi-aplikasi *wallet* tersebut mempunyai ciri-ciri sebagai berikut:

1. Aplikasi *wallet* harus di*download* dan di*install* terlebih dulu di komputer konsumen sebelum dapat digunakan dalam transaksi perdagangan.
2. Aplikasi *wallet* perlu melakukan *update* terus menerus jika dikeluarkan versi *wallet* yang lebih baru.
3. Aplikasi *wallet* yang digunakan untuk vendor yang satu tidak sama dengan vendor yang lain sehingga konsumen harus menginstall aplikasi *wallet* yang lain untuk melakukan transaksi dengan pedagang yang menggunakan jenis *wallet* yang berbeda.
4. Aplikasi *wallet* diletakkan di komputer konsumen, hal ini akan mengurangi keamanan transaksi karena seorang konsumen yang ahli dapat mengubah aplikasi *wallet* tersebut.
5. Aplikasi *wallet* menyimpan informasi tentang penggunanya di *hardisk client.* Metode ini dapat dimanfaatkan oleh orang-orang yang tidak berhak dengan mencuri data-data yang ada di *hardisk* tersebut. Selain itu, sulit bagi konsumen untuk melakukan transaksi dari komputer lain karena data-datanya ada di komputer miliknya.
6. Protokol pembayaran SSL (*Secure Socket Layer*) merupakan protokol standar yang paling banyak diimplementasikan oleh SPI-SPI saat ini. SSL merupakan protokol pada lapisan *transport*, yang didukung oleh dua *browser* utama yaitu Netscape Navigator dan Microsoft Internet Explorer. SSL yang banyak digunakan secara Internasional menggunakan teknologi 40-bit dari RSA. Hal ini berdasarkan undang-undang regulasi Amerika Serikat, yaitu hanya melakukan ekspor teknologi enkripsi sebatas 40-bit, sedangkan untuk domestik boleh sampai 128-bit. SSL yang memanfaatkan teknologi kunci publik 40-bit dari RSA ternyata dapat dijebol dalam waktu 1,3 hari dengan 100 komputer menggunakan *brute-force attack* [DHMM96]. Ini tidak berarti teknologi SSL tidak bermanfaat. Tujuannya jelas, yakni agar biaya yang dikeluarkan oleh pihak penyerang lebih besar daripada ‘harga’ informasi informasi yang dienkripsi melalui SSL, sehingga secara ekonomi tidak menguntungkan. Sedangkan kunci enkripsi simetris yang digunakan adalah 128-bit (40-bit dienkripsi dan 80-bit tidak dienkripsi saat kunci dipertukarkan). Dengan *clear-text attack*, masih sangat sulit untuk memecahkannya. Kelemahan utama dari protokol SSL ini sebenarnya terletak pada informasi kartu kredit. Seorang pedagang dapat mengetahui informasi kartu kredit milik konsumennya sehingga ia dengan mudah dapat menggunakan informasi kartu kredit tersebut untuk kepentingannya.

2.4. ANALISIS SISTEM PERDAGANGAN YANG DIKEMBANGKAN

Berikut ini dianalisis kebutuhan-kebutuhan sistem perdagangan yang dikembangkan sebagai perbaikan dari sistem perdagangan yang ada saat ini :

1. Sistem perdagangan harus memenuhi kebutuhan-kebutuhan umum untuk SPI dan sebaiknya memenuhi syarat-syarat tambahan yang harus dimiliki SPI seperti yang telah diuraikan sebelumnya.
2. Konsumen dapat dengan mudah melakukan transaksi tanpa perlu melakukan *download* dan instalasi *wallet* untuk setiap vendor yang berbeda.
3. Konsumen seharusnya tidak dapat mengubah program *wallet* yang digunakan dalam transaksi.
4. Informasi tentang identitas pengguna transaksi harus aman dan tidak dapat dicuri oleh orang-orang yang tidak berhak.
5. Sistem perdagangan harus menggunakan protokol yang aman sehingga orang lain (selain otoritas yang mengeluarkan kartu kredit) tidak dapat mengetahui nomor kartu kredit konsumen.
6. Konsumen dapat melakukan transaksi kapan saja, di mana saja, dan di komputer apa saja.

BAB III
LANDASAN TEORI

Bab ini membahas mengenai landasan teori yang menjadi dasar pengetahuan untuk tahap implementasi. Teori-teori yang diuraikan di sini adalah *Secure Socket Layer* (SSL), *Secure Electronic Transaction* (SET), kriptografi, smartcard, *Java Native Interface* (JNI), dan standar pengkodean ASN.1, BER dan DER.

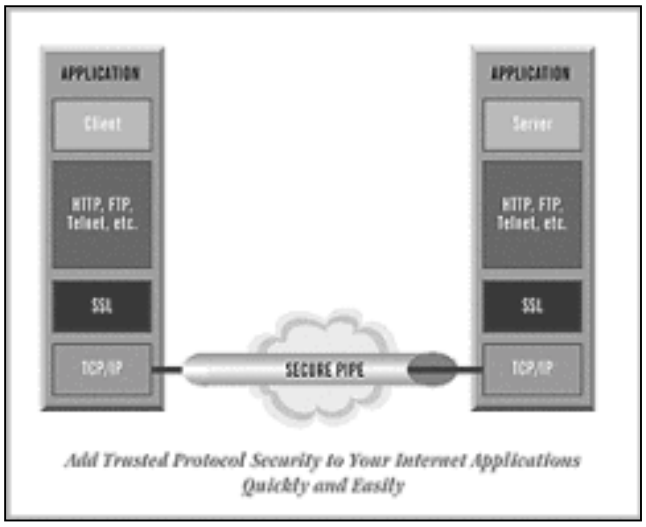
3.1 SECURE SOCKET LAYER (SSL)

3.1.1 Dekripsi

Secure Socket Layer (SSL) adalah suatu protokol komunikasi untuk hubungan *point to point* pada Internet [RSID99]. SSL menyediakan fasilitas keamanan seperti kerahasiaan, keutuhan dan keabsahan. Protokol ini bebas dipergunakan siapa saja, bahkan didukung oleh dua *browser* utama, yaitu Netscape Navigator dan Microsoft Internet Explorer. SSL juga tidak mengkhususkan diri untuk hanya mendukung protokol tertentu seperti HTTP, karenanya SSL menggunakan *port* 443 untuk berhubungan dengan pelayan internet yang juga memiliki fasilitas SSL. Lapisan aplikasi diatasnya dapat memanfaatkan kunci yang telah dinegosiasikan oleh SSL. SSL dirancang agar fasilitas keamanan pada aplikasi yang memanfaatkan SSL tidak merepotkan pemakainya. SSL telah banyak mengalami perkembangan, bahkan saat ini telah dikeluarkan teknologi baru yang disebut *Transport Layer Security* (TLS) sebagai pengembangan lebih lanjut dari SSL.

Dengan memanfaatkan SSL, aplikasi internet dapat melakukan komunikasi yang aman melalui fasilitas yang disediakan oleh SSL [ELTC96]:

- Kerahasiaan pesan, sehingga tidak bisa dibaca oleh pihak yang tidak diinginkan
- Keutuhan pesan, sehingga tidak bisa diubah-ubah di tengah jalan
- Keabsahan, sehingga meyakinkan pihak-pihak yang berkomunikasi mengenai keabsahan pesan dan keabsahan jati diri lawan bicaranya.



Gambar III.1. Secure Socket Layer

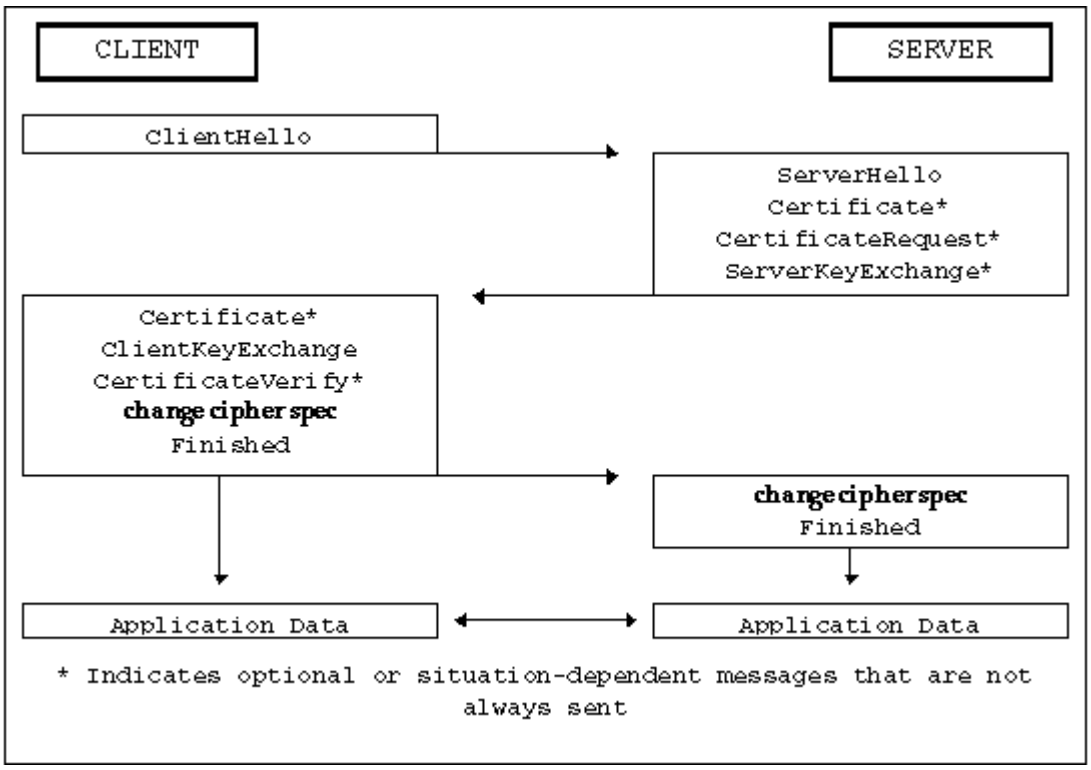
3.1.2 Algoritma

SSL dapat menjaga kerahasiaan (*confidentiality*) dari informasi yang dikirim karena menggunakan teknologi enkripsi yang maju dan dapat di-*update* jika ada teknologi baru yang lebih bagus. Dengan penggunaan sertifikat digital, SSL menyediakan autentikasi yang transparan antara *client* dengan *server*. SSL menggunakan algoritma RSA untuk membuat tanda tangan digital (*digital signature*) dan amplop digital (*digital envelope*). Selain itu, untuk melakukan enkripsi dan dekripsi data setelah koneksi dilakukan, SSL menggunakan RC4 sebagai algoritma standar untuk enkripsi kunci simetri.

Saat aplikasi menggunakan SSL, sebenarnya terjadi dua sesi, yakni sesi *handshake* dan sesi pertukaran informasi. Berikut dijabarkan sebuah skenario yang aman dari sesi *handshake* SSL versi 3.0 [IETC96]:

- Klien mengirimkan *client hello* yang harus dijawab dengan *server hello*. Tahap ini terjadi kesepakatan atas pengunaan versi protokol, *session ID*, perangkat kriptografi, metoda kompresi.
- Pelayan kemudian dapat mengirim sertifikat kepada klien. Selain itu pelayan bisa meminta klien untuk menunjukkan sertifikatnya – namun tidak harus. Pelayan lantas mengirimkan pesan *server hello done*, lalu menunggu jawaban dari klien.

- Jika pelayan meminta sertifikat dengan pesan *certificate request*, maka klien harus mengirimkan pesan *certificate message* atau *no certificate*.



Gambar III.2. Security Handshake

- Pesan *client key exchange* kini dikirim, dimana pesan yang disandikan itu tergantung dari algoritma kriptografi kunci publik yang disepakati pada tahap pertama. Pesan itu berisi kunci-kunci yang dibuat secara acak oleh klien untuk keperluan enkripsi dan perhitungan sidik jari (*hash*). Jika memungkinkan, dapat pula disertai tanda tangan digital melalui pengiriman pesan *certificate verify*.
- Lalu pesan *change cipher spec* dikirimkan oleh klien sambil mengaktifkan spesifikasi *cipher* yang telah disepakati. Hal ini dilakukan dengan mengkopi *pending cipher spec* ke *current cipher spec*. Segera setelah itu, klien mengirimkan pesan *finished* guna mengakhiri *handshake*. Hal serupa dilakukan pula oleh pelayan.
- Akhirnya pelayan dan klien dapat bertukar pesan dengan menyandikannya dengan kunci dan algoritma yang telah disepakati bersama pada level aplikasi.
- Guna mencegah serangan yang dilakukan terhadap pesan yang disandikan, pelayan dan klien dapat melakukan *handshake* beberapa kali pada session ID yang sama guna mengubah kunci,

namun mereka tidak perlu mengubah parameter komunikasi yang telah disepakati sebelumnya.

- Patut juga dicatat bahwa klien perlu memeriksa sertifikat yang diterimanya agar lebih yakin bahwa dia sedang berkomunikasi dengan pelayan yang diinginkan. Jika klien tidak memeriksanya, masih ada kesempatan bagi seseorang untuk menyamar menjadi pelayan yang seharusnya diajak bicara (masih termasuk serangan *man-in-the-middle*).
- Klien memeriksa sertifikat digital itu dengan membandingkan tanda tangan OS (otoritas sertifikat) pada sertifikat digital itu dengan daftar OS yang dimiliki. Biasanya, *browser-browser* seperti Netscape Navigator atau Microsoft Internet Explorer sudah menyertakan sertifikat digital dari OS utama yang terkenal, sehingga memudahkan pemeriksaan sertifikat digital pada koneksi SSL. Penyertaan serfikat digital OS utama pada *browser* akan menghindarkan klien dari pemalsuan sertifikat OS utama.

3.1.3 Keamanan

Pada sesi handshake di atas, ada serangan yang berusaha menipu pelayan dengan merekam pembicaraan antara klien dan pelayan sebelumnya. Dalam skenario SSL tanpa sertifikat klien, hal ini dapat dicegah dengan penggunaan angka random yang dipertukarkan setiap terjadi pertukaran pesan [IETC96]. *ClientHello.random* dan *Serverhello.random* dapat dibuat dan dipertukarkan saat tahap pertama.

SSL yang digunakan secara Internasional menggunakan teknologi kunci publik 40-bit dari RSA, sedangkan kunci publik yang digunakan di Amerika Serikat sepanjang 128-bit. Hal ini berdasarkan peraturan regulasi dari pemerintah Amerika Serikat yang melarang ekspor kunci enkripsi melebihi 40-bit. Kunci publik 40-bit ini ternyata dapat dijebol dalam waktu 1,3 hari dengan 100 komputer menggunakan *brute-force attack* [DHMM96]. Ini tidak berarti teknologi SSL tidak bermanfaat. Tujuannya jelas, yakni agar biaya yang dikeluarkan oleh pihak penyerang lebih besar dari pada ‘harga’ informasi yang dienkripsi melalui SSL, sehingga secara ekonomi tidak menguntungkan. Sedangkan kunci enkripsi simetris yang dipergunakan adalah 128-bit (40-bit dienkripsi dan 80-bit tidak dienkripsi saat kunci dipertukarkan). Dengan *clear-text attack*, masih sangat sulit untuk memecahkannya.

3.2 SECURE ELECTRONIC TRANSACTION (SET)

3.2.1 Deskripsi

Dua raksasa kartu kredit dunia, Visa dan MasterCard, bekerja sama membuat suatu standar pembayaran pada saluran tak aman, yang diberi nama *Secure Electronic Transaction* (SET) [VIMA97]. Kini, sebagian besar penyedia jasa pelayanan pembayaran di Internet telah setuju untuk mengikuti standar SET. Menurut spesifikasi SET, ada beberapa kebutuhan bisnis yang perlu ditangani:

1. Keamanan pengiriman informasi pemesanan dan pembayaran.
2. Integritas data dalam setiap transaksi.
3. Otentikasi bahwa seorang konsumen adalah seorang pemegang kartu (*cardholder*) yang valid pada suatu perusahaan penyelenggara pembayaran tertentu (misalnya: Visa atau MasterCard).
4. Otentikasi bahwa seorang pedagang memang benar-benar bisa menerima jenis pembayaran tersebut.
5. Menyediakan suatu sistem pembayaran yang tidak terikat kepada suatu protokol perangkat keras atau perangkat lunak tertentu, dengan kata lain dapat bekerja dengan berbagai macam perangkat lunak dan berbagai penyedia jasa.

Banyak *developer* yang sudah menyatakan dukungannya terhadap SET bagi produk/produk penunjang sistem perdagangan Internet mereka, seperti Microsoft, IBM, Netscape, SAIC, GTE, Open Market, CyberCash, Terisa Systems and VeriSign. Bahkan kini perusahaan penyelenggara *charge card* seperti American Express, akhirnya menyatakan dukungannya untuk SET. Penulis merasa cukup yakin bahwa SET, ataupun derivasinya akan menjadi standar di masa depan.

3.2.2 Pihak-pihak yang terlibat

Dalam skema SET, pihak-pihak yang terlibat adalah:

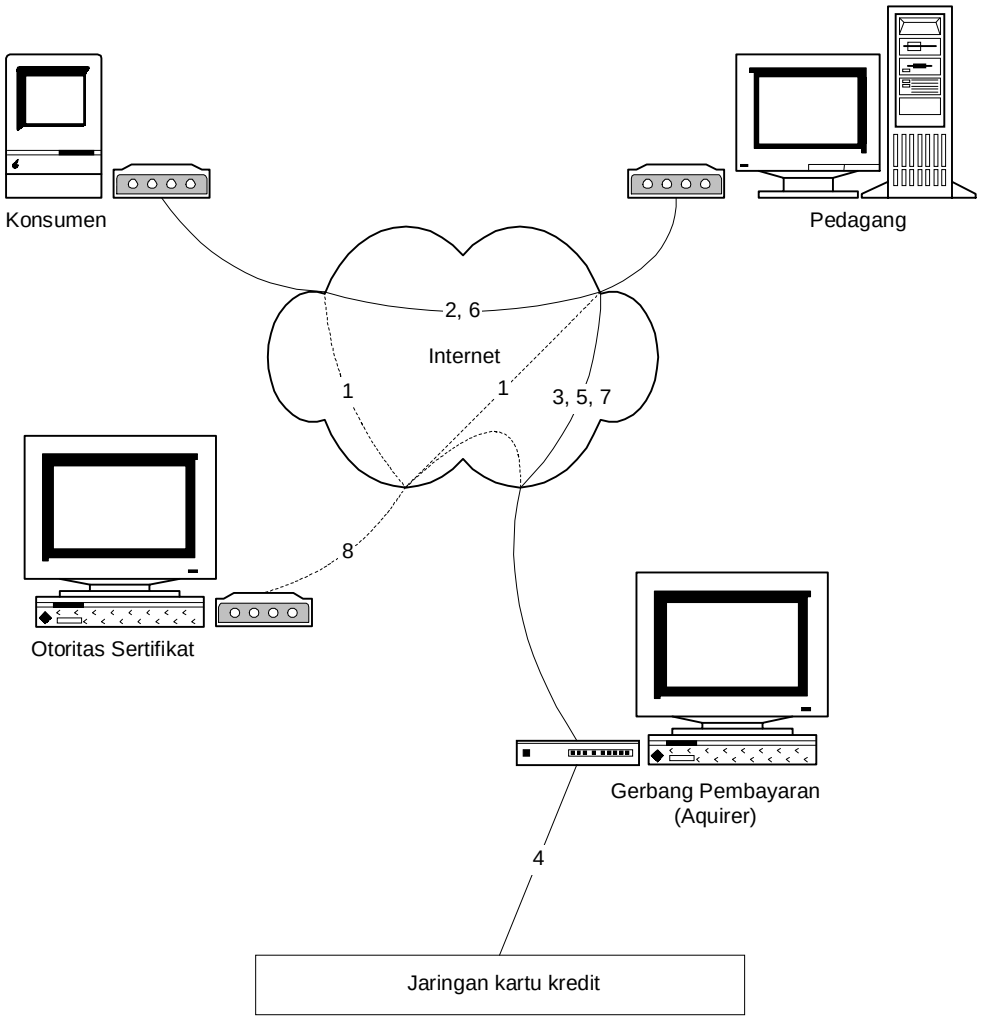
- **Cardholder**, yaitu konsumen yang menggunakan kartu pembayaran resmi yang dijamin oleh suatu Issuer untuk transaksi jual-beli di Internet,
- **Issuer,** yaitu institusi keuangan/bank yang mengeluarkan kartu pembayaran bagi cardholder dan melakukan otorisasi terhadap kartu tersebut ketika digunakan untuk berbelanja di Internet,

- **Merchant,** yaitu pedagang yang menjual dagangannya melalui internet dan dijamin oleh suatu Acquirer dalam melakukan transaksi pembayarannya lewat Internet,
- **Acquirer,** yaitu institusi keuangan/bank yang menjamin merchant untuk berdagang dan melakukan otorisasi terhadap pembayaran dalam setiap transaksi dagang di Internet,
- **Payment Gateway,** yaitu suatu alat yang biasanya dioperasikan oleh Acquirer (bisa juga oleh pihak ketiga lain) yang berfungsi untuk memroses instruksi pembayaran, menghubungkan antara Acquirer dan Issuer,
- **Brand,** yaitu institusi di atas Issuer dan Acquirer, merupakan pemilik merk dari produk sistem pembayaran yang digunakan oleh Issuer dan Acquirer

3.2.3 Perangkat Lunak

SET tidak hanya dirancang untuk transaksi pada Web saja, namun juga bisa dipergunakan pada media lainnya. Pedagang dapat saja menyebarkan katalog dalam CD-ROM. Setelah konsumen memilih barang yang akan dibelinya dari katalog CD-ROM itu, konsumen kemudian dapat melakukan pembayaran dengan protokol SET, baik itu dengan *browser* atau surat elektronik.

3.2.4 Alur Transaksi



Gambar III.3. Diagram topologi protokol SET

Secara singkat, alur transaksi pada skenario SET dapat dijelaskan sebagai berikut [ARRI97]:

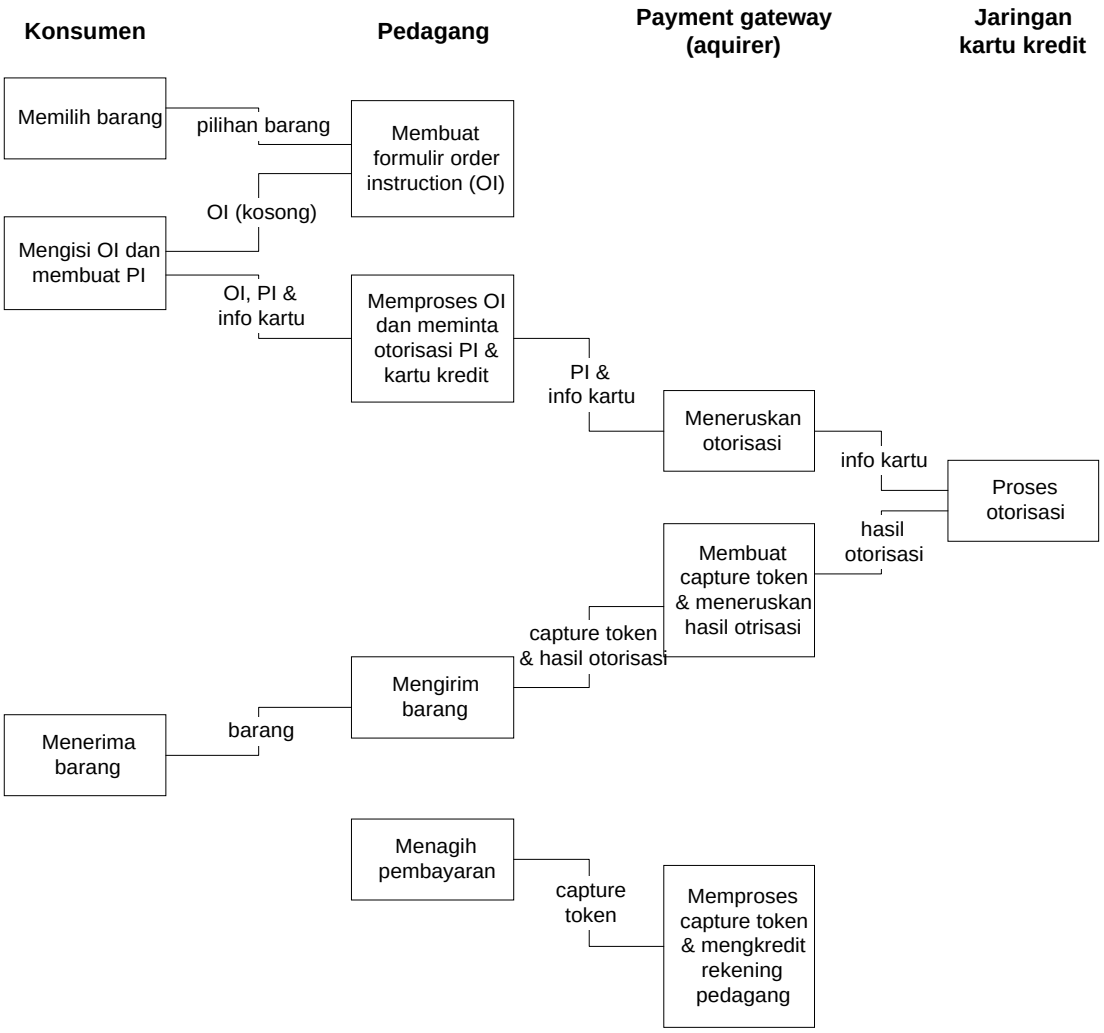
1. Untuk melakukan transaksi SET, konsumen dan pedagang harus mendapatkan sertifikat terlebih dahulu dari otoritas sertifikat (OS). Konsumen dalam langkah ini harus mengetikkan *personal account number* (PAN) dan informasi jati dirinya. Pedagang dalam langkah ini juga harus memberikan informasi jati dirinya kepada OS.
2. Konsumen kemudian dapat mulai berbelanja. Jika sudah memilih barang apa yang hendak dibeli, konsumen membuat *order instruction* (OI) dan *payment instruction* (PI). Konsumen menyerahkan OI dan PI kepada pedagang. PI tidak bisa dibaca oleh pedagang karena dienkripsi dengan kunci publik gerbang pembayaran (*payment gateway*).
3. Setelah pedagang memproses OI, maka pedagang melakukan otorisasi PI melalui

gerbang pembayaran. Seringkali *aquirer* bertindak sebagai gerbang pembayaran.

4. Gerbang pembayaran melakukan otorisasi kartu kredit dengan *issuer* melalui jaringan privat kartu kredit.
5. Jika otorisasi disetujui, maka gerbang pembayaran menginstruksikan pedagang untuk menyerahkan barang dagangannya kepada konsumen.
6. Konsumen menerima barang dagangannya.
7. Pedagang kemudian dapat memperoleh pembayarannya dengan melakukan proses *capture* melalui gerbang pembayaran pula. Langkah ini sering di-*batch*, sehingga akan ada tenggang waktu antara permintaan pembayaran (*payment capture*) dengan proses otorisasi.
8. Setiap melakukan komunikasi, setiap pihak yang terlibat dalam transaksi dapat melakukan otentikasi sertifikat digital pihak yang lain dengan menghubungi OS.

3.2.5 Klasifikasi

Berdasarkan skenario di atas, terlihat bahwa sistem perdagangan di Internet dengan skenario SET dijalankan secara *on-line*. Protokol SET dapat mendukung sistem pembayaran dengan kartu kredit, *charge card* dan kartu debit. Namun kini memang belum ada bank pengelola kartu debit yang menyatakan dukungannya terhadap protokol SET. Tentunya, transaksi dengan protokol SET ini terlacak. Pada skenario SET, transaksi tidak dapat dilakukan antarkonsumen (*peer-to-peer*), namun harus antara konsumen dengan pedagang. Konsumen dapat dilihat jati dirinya oleh pedagang, karena pedagang dan konsumen saling memeriksa sertifikat digital yang dipertukarkan. Meskipun begitu, informasi kartu konsumen tidak dapat diketahui pedagang. Protokol SET tidak cocok untuk transaksi *micropayments.*



Gambar III.4. Diagram alur data protokol SET

3.2.6 Keamanan dan Serangan

Inti dari keamanan dalam protokol SET adalah penggunaan sertifikat digital. Secara teoritis, tanpa *brute-force attack*, sertifikat digital dapat bertahan terhadap serangan *man-in-the-middle* dan juga *replay attack.* Hal ini disebabkan karena siapapun yang ingin melakukan pemeriksaan dapat memastikan apakah kunci publik yang diterimanya sah atau tidak. Seperti sudah dijelaskan sebelumnya, bagian yang rentan adalah saat pemberian sertifikat digital OS utama kepada pihak-pihak lain yang memerlukan seperti kepada para pengembang perangkat lunak untuk SET.

Sertifikat konsumen tidak memiliki informasi kartu konsumen, yakni *personal account number* (PAN) dan tanggal kadaluarsanya, namun berisi *hash* dari PAN, tanggal kadaluarsa dan sebuah angka rahasia yang hanya diketahui konsumen (*personal identification number* / PIN).

Jadi jika pedagang memeriksa sertifikat milik seorang konsumen, maka pedagang meskipun dapat melihat jati diri konsumen namun pedagang tetap tidak dapat melihat informasi kartunya. Seorang penyerang yang memiliki sertifikat seorang konsumen juga tidak dapat menggunakannya tanpa mengetahui informasi kartu dan PIN. Informasi kartu dan PIN dalam skenario SET dikirimkan dalam bentuk terenkripsi kepada gerbang pembayaran untuk pemeriksaan. Gerbang pembayaran tidak hanya memeriksa keabsahan sertifikat digital milik konsumen, tapi juga memeriksa apakah *hash* dari informasi kartu dan angka rahasia tadi sesuai dengan nilai *hash* yang ada dalam sertifikat.

Ukuran kunci yang dipergunakan dalam SET amat panjang, yang dapat dikategorikan *hard encryption*. Semua kunci berukuran 1024 bit, kecuali OS utama menggunakan ukuran kunci 2048 bit. Kunci sepanjang ini sulit dipecahkan dengan *brute-force-attack*.

Penggunaan sertifikat digital sebagai sarana pengamanan komunikasi memang membuat SPI yang menggunakan protokol SET menjadi sistem yang aman. Namun muncul pula masalah bagaimana menyimpan kunci privat dari sertifikat digital yang bersangkutan. Dalam implementasi awalnya, mungkin kunci itu disimpan dalam *hard disk* dengan pengamanan dienkripsi ber-*password*, namun pengembangan yang lebih jauh adalah dengan menyimpan sertifikat digital dan kunci privat itu di dalam *tamper-proof device* seperti kartu chip.

Protokol SET juga menggunakan suatu perangkat kriptografi baru, yakni tanda tangan pesan ganda (*dual signature*). Dengan menggunakan tanda tangan pesan ganda, gerbang pembayaran saat melakukan otorisasi dapat memastikan bahwa PI yang diterimanya memang benar berhubungan dengan suatu OI tertentu, namun gerbang pembayaran tetap tidak tahu isi OI tersebut.

3.2.7 Kepercayaan dan Penipuan

Karena mengandalkan sertifikat digital, maka tentunya pihak-pihak yang bertransaksi mengandalkan kepercayaan mereka terhadap OS yang menerbitkan sertifikat digital. Sama halnya dengan transaksi menggunakan kartu kredit, tentunya konsumen harus mempercayai lembaga keuangan pengelola kartu kredit yang melakukan otorisasi.

Untuk pemeriksaan sertifikat digital, pihak-pihak yang bertransaksi juga membutuhkan informasi dari OS mengenai sertifikat siapa saja yang dibatalkan. Tentunya OS yang menyimpan daftar sertifikat terbatalkan ini harus dipercayai oleh pihak-pihak yang melakukan transaksi.

3.2.8 Pencatatan

Skenario protokol SET tidak menspesifikasi pencatatan yang dilengkapi dengan tanda tangan digital dari pihak-pihak yang melakukan transaksi. Namun dalam implementasinya, pencatatan dapat dilakukan di perangkat lunak klien yang dipergunakan oleh konsumen dan juga di *web server* pedagang.

3.2.9 Penerimaan Pembayaran dan Biaya Transaksi

Spesifikasi SET tidak menjelaskan mengenai biaya tambahan atas transaksi. Namun, jika pihak *aquirer* sendiri yang menjadi gerbang pembayaran, tentunya boleh dikatakan tidak ada biaya tambahan. Jadi hampir tidak ada bedanya karena dalam transaksi kartu kredit *on-line* yang biasa, pedagang akan melakukan otorisasi itu melalui *aquirer* juga. Seperti sudah dijelaskan di atas, karena perubahan sistem dimana pedagang tidak mendapatkan informasi kartu kredit, maka memang ada perubahan prosedur penagihan ke *aquirer*.

3.2.10 Keuntungan SET

Berikut ini merupakan keuntungan-keuntungan menggunakan protokol SET dalam sistem perdagangan:

1. Transaksi dapat dilakukan dengan **lebih aman** karena:
   - Bagi *cardholder*: Merchant tidak dapat mengetahui nomor kartu kredit *cardholder*.
   - Bagi *merchant*: *cardholder* tidak dapat menyangkal jika ia sudah pernah melakukan suatu transaksi.
2. Transaksi dapat dilakukan dengan **lebih cepat dan lebih mudah** karena otorisasi kartu kredit dapat dilakukan secara otomatis (tidak membutuhkan penanganan manual).
3. Transaksi dapat dilakukan dengan **lebih murah**, karena resiko yang ditanggung oleh pihak Jaringan kartu kredit lebih kecil. Sebagai contoh, Bank biasanya meminta *charge* 3% s/d 5% untuk transaksi dengan SSL, sedangkan pada SET *charge* yang diminta oleh Bank hanya 2%.
4. **Otentitas data dapat lebih terjamin**, karena setiap pesan yang dikirim selalu disertai dengan sertifikat.
5. **Cardholder merasa lebih aman**. *Merchant* yang ingin bisa menerima pembayaran on-line, akan di-audit oleh *payment gateway*, sedangkan *payment gateway* adalah perusahaan yang memiliki integritas yang tinggi. Hal ini menyebabkan *cardholder* merasa lebih aman, karena ia bertransaksi dengan perusahaan yang terjamin integritasnya (*payment*

gateway).

6. **Sudah banyak vendor yang mendukung sistem SET** ini, diantaranya: Microsoft, IBM, Netscape, SAIC, GTE, Open Market, CyberCash, Terisa Systems and VeriSign. Bahkan kini perusahaan penyelenggara *charge card* seperti American Express, akhirnya menyatakan dukungannya untuk SET.
7. **Menyediakan suatu sistem pembayaran yang tidak terikat kepada suatu protokol perangkat keras atau perangkat lunak tertentu**, dengan kata lain dapat bekerja dengan berbagai macam perangkat lunak dan berbagai penyedia jasa.

3.2.11 Prospek

Suatu *pilot project* SET telah berhasil dilakukan di Jepang pada awal tahun 1997. Implementasi SET saat ini menggunakan *hard disk* sebagai media untuk menyimpan kunci privat. Meskipun dilindungi oleh *password*, namun tetap kurang aman. Oleh karena itu, diusulkan untuk menggunakan *smartcard* untuk menyimpan kunci privat. IBM kini juga mengembangkan protokol SET menjadi superSET [AARO97].

SSL	SET
Sudah terintegrasi dengan *browser-browser* yang terkenal, seperti Internet Explorer & Netscape Navigator.	Tidak terintegrasi dengan *browser-browser* tersebut.
Tujuan utamanya adalah untuk memberikan keamanan pada komunikasi *point to point* di saluran yang tak aman.	Tujuan utamanya adalah untuk memberikan keamanan pada transaksi perdagangan di saluran yang tak aman.
Pedagang dapat mengetahui nomor kartu pembayaran milik konsumen.	Pedagang tidak dapat mengetahui nomor kartu pembayaran milik konsumen.
Bank meminta biaya *charge* sebesar 3% sampai 5%.	Transaksi lebih murah karena bank hanya meminta biaya *charge* sebesar 2%.
Otorisasi tidak dilakukan secara otomatis, sehingga membutuhkan waktu lebih lama.	Transaksi lebih cepat karena otorisasi dilakukan secara otomatis.
Batasan kunci enkripsi 40-bit untuk *browser* di luar AS dan Kanada dan 128-bit untuk *browser* di AS dan Kanada. Pengembang dapat menentukan ukuran kunci enkripsinya	Batasan kunci enkripsi ditentukan oleh pengembang protokol SET.

sendiri jika tidak menggunakan fasilitas SSL dari *browser*.	
Transaksi secara *point to point*, hanya melibatkan konsumen dan pedagang.	Transaksi dilakukan tiga pihak, yaitu konsumen, pedagang, dan gerbang pembayaran.

Tabel III-1 Perbandingan Protokol SSL dan SET

3.3 KRIPTOGRAFI

3.3.1 Deskripsi

Kriptografi adalah ilmu yang mempelajari bagaimana membuat suatu pesan yang dikirim oleh pengirim dapat disampaikan kepada penerima dengan aman [SCHN96]. Kriptografi dapat memenuhi kebutuhan umum suatu transaksi:

- Kerahasiaan (*confidentiality*) dijamin dengan melakukan enkripsi (penyandian).
- Keutuhan (*integrity*) atas data-data pembayaran dilakukan dengan fungsi *hash* satu arah.
- Jaminan atas identitas dan keabsahan (*authenticity*) pihak-pihak yang melakukan transaksi dilakukan dengan menggunakan *password* atau sertifikat digital. Sedangkan keotentikan data transaksi dapat dilakukan dengan tanda tangan digital.
- Transaksi dapat dijadikan barang bukti yang tidak bisa disangkal (*non-repudiation*) dengan memanfaatkan tanda tangan digital dan sertifikat digital.

Pembakuan penulisan pada kriptografi dapat ditulis dalam bahasa matematika. Fungsi-fungsi yang mendasar dalam kriptografi adalah enkripsi dan dekripsi. Enkripsi adalah proses mengubah suatu pesan asli (*plaintext*) menjadi suatu pesan dalam bahasa sandi (*ciphertext*).

C = E (M)

dimana

M = pesan asli

E = proses enkripsi

C = pesan dalam bahasa sandi (untuk ringkasnya disebut sandi)

Sedangkan dekripsi adalah proses mengubah pesan dalam suatu bahasa sandi menjadi pesan asli kembali.

M = *D* (*C*)

D = proses dekripsi

Umumnya, selain menggunakan fungsi tertentu dalam melakukan enkripsi dan dekripsi, seringkali fungsi itu diberi parameter tambahan yang disebut dengan istilah kunci.

Untuk memudahkan penggambaran suatu skenario komunikasi dalam pembahasan selanjutnya, maka dipergunakan nama-nama orang yang relevan dengan peran yang dilakukannya dalam komunikasi itu.

Kode & nama	Penjelasan
A: Anto	Pihak pertama
B: Badu	Pihak kedua
C: Chandra	Pihak ketiga
E: Edi	Pihak penyadap informasi yang tidak diperuntukkan kepadanya (*eavesdropper*)
M: Maman	Pihak yang tidak hanya menyadap informasi, namun juga mengubah informasi yang disadap (*malacious person*)
T: Tari, Tata, Tania	Pihak yang dipercaya oleh pihak pertama, kedua dan ketiga (*trusted person*)

Tabel III-2. Nama-nama ganti untuk mempermudah penjelasan

3.3.2 Jenis Serangan

Selain ada pihak yang ingin menjaga agar pesan tetap aman, ada juga ternyata pihak-pihak yang ingin mengetahui pesan rahasia tersebut secara tidak sah. Bahkan ada pihak-pihak yang ingin agar dapat mengubah isi pesan tersebut. Ilmu untuk mendapatkan pesan yang asli dari pesan yang telah disandikan tanpa memiliki kunci untuk membuka pesan rahasia tersebut disebut kriptoanalisis. Sedangkan usaha untuk membongkar suatu pesan sandi tanpa mendapatkan kunci dengan cara yang sah dikenal dengan istilah serangan (*attack*).

Di bawah ini dijelaskan beberapa macam penyerangan terhadap pesan yang sudah dienkripsi:

- *Ciphertext only attack*, penyerang hanya mendapatkan pesan yang sudah tersandikan saja.
- *Known plaintext attack*, dimana penyerang selain mendapatkan sandi, juga mendapatkan pesan asli. Terkadang disebut pula *clear-text attack*.
- *Choosen plaintext attack*, sama dengan *known plaintext attack*, namun penyerang bahkan dapat memilih penggalan mana dari pesan asli yang akan disandikan.

Berdasarkan bagaimana cara dan posisi seseorang mendapatkan pesan-pesan dalam saluran komunikasi, penyerangan dapat dikategorikan menjadi:

- *Sniffing*: secara harafiah berarti mengendus, tentunya dalam hal ini yang diendus adalah pesan (baik yang belum ataupun sudah dienkripsi) dalam suatu saluran komunikasi. Hal ini umum terjadi pada saluran publik yang tidak aman. Sang pengendus dapat merekap pembicaraan yang terjadi.
- *Replay attack* [DHMM 96]: Jika seseorang bisa merekam pesan-pesan *handshake* (persiapan komunikasi), ia mungkin dapat mengulang pesan-pesan yang telah direkamnya untuk menipu salah satu pihak.
- *Spoofing* [DHMM 96]: Penyerang – misalnya Maman – bisa menyamar menjadi Anto. Semua orang dibuat percaya bahwa Maman adalah Anto. Penyerang berusaha meyakinkan pihak-pihak lain bahwa tak ada salah dengan komunikasi yang dilakukan, padahal komunikasi itu dilakukan dengan sang penipu/penyerang. Contohnya jika orang memasukkan PIN ke dalam mesin ATM palsu – yang benar-benar dibuat seperti ATM asli – tentu sang penipu bisa mendapatkan PIN-nya dan copy pita magentik kartu ATM milik sang nasabah. Pihak bank tidak tahu bahwa telah terjadi kejahatan.
- *Man-in-the-middle* [SCHN96]: Jika *spoofing* terkadang hanya menipu satu pihak, maka dalam skenario ini, saat Anto hendak berkomunikasi dengan Badu, Maman di mata Anto seolah-olah adalah Badu, dan Maman dapat pula menipu Badu sehingga Maman seolah-olah adalah Anto. Maman dapat berkuasa penuh atas jalur komunikasi ini, dan bisa membuat berita fitnah.

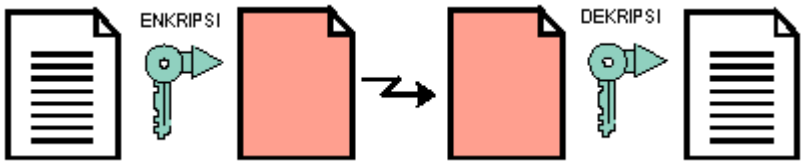
Kabel koaksial yang sering dipergunakan pada jaringan sangat rentan terhadap serangan *vampire tap* [Tane 89], yakni perangkat keras sederhana yang bisa menembus bagian dalam kabel koaksial sehingga dapat mengambil data yang mengalir tanpa perlu memutuskan komunikasi data yang sedang berjalan. Seseorang dengan *vampire tap* dan komputer jinjing dapat melakukan serangan pada bagian apa saja dari kabel koaksial.

Penyerang juga bisa mendapatkan kunci dengan cara yang lebih tradisional, yakni dengan melakukan penyiksaan, pemerasan, ancaman, atau bisa juga dengan menyogok seseorang yang memiliki kunci itu. Ini adalah cara yang paling ampuh untuk mendapat kunci.

3.3.3 Kunci Simetris

Ini adalah jenis kriptografi yang paling umum dipergunakan. Kunci untuk membuat pesan yang disandikan sama dengan kunci untuk membuka pesan yang disandikan itu. Jadi

pembuat pesan dan penerimanya harus memiliki kunci yang sama persis. Siapapun yang memiliki kunci tersebut – termasuk pihak-pihak yang tidak diinginkan – dapat membuat dan membongkar rahasia *ciphertext*. Problem yang paling jelas disini terkadang bukanlah masalah pengiriman *ciphertext*-nya, melainkan masalah bagaimana menyampaikan kunci simetris tersebut kepada pihak yang diinginkan. Contoh algoritma kunci simetris yang terkenal adalah DES (*Data Encryption Standard*) dan RC4.

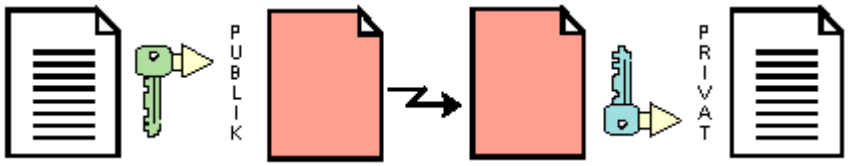


Gambar III.5. Kunci simetris

3.3.4 Kunci Asimetris

Pada pertengahan tahun 70-an Whitfield Diffie dan Martin Hellman menemukan teknik enkripsi asimetris yang merevolusi dunia kriptografi. Kunci asimetris adalah pasangan kunci-kunci kriptografi yang salah satunya dipergunakan untuk proses enkripsi dan yang satu lagi untuk dekripsi. Semua orang yang mendapatkan kunci publik dapat menggunakannya untuk mengenkripsikan suatu pesan, sedangkan hanya satu orang saja yang memiliki rahasia tertentu – dalam hal ini kunci privat – untuk melakukan pembongkaran terhadap sandi yang dikirim untuknya.

Dengan cara seperti ini, jika Anto mengirim pesan untuk Badu, Anto dapat merasa yakin bahwa pesan tersebut hanya dapat dibaca oleh Badu, karena hanya Badu yang bisa melakukan dekripsi dengan kunci privatnya. Tentunya Anto harus memiliki kunci publik Badu untuk melakukan enkripsi. Anto bisa mendapatkannya dari Badu, ataupun dari pihak ketiga seperti Tari.



Gambar III.6. Penggunaan kunci asimetris

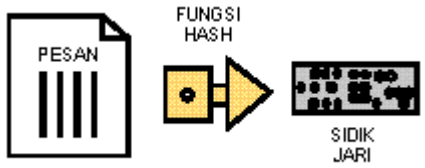
Teknik enkripsi asimetris ini jauh lebih lambat ketimbang enkripsi dengan kunci simetris.

Oleh karena itu, biasanya bukanlah pesan itu sendiri yang disandikan dengan kunci asimetris, namun hanya kunci simetrislah yang disandikan dengan kunci asimetris. Sedangkan pesannya dikirim setelah disandikan dengan kunci simetris tadi. Contoh algoritma terkenal yang menggunakan kunci asimetris adalah RSA (merupakan singkatan penemunya yakni Rivest, Shamir dan Adleman).

3.3.5 Fungsi *Hash* Satu Arah

Kini akan dibahas mengenai keutuhan pesan saat dikirimkan. Bagaimana jika Anto mengirimkan surat pembayaran kepada Badu sebesar 1 juta rupiah, namun di tengah jalan Maman (yang ternyata berhasil membobol sandi entah dengan cara apa) membubuhkan angka 0 lagi dibelakangnya sehingga menjadi 10 juta rupiah? Di mata Tari, pesan tersebut harus utuh, tidak diubah-ubah oleh siapapun, bahkan bukan hanya oleh Maman, namun juga termasuk oleh Anto, Badu dan gangguan pada transmisi pesan (*noise*). Hal ini dapat dilakukan dengan fungsi *hash* satu arah (*one-way hash function*), yang terkadang disebut sidik jari (*fingerprint*), *hash*, *message integrity check,* atau *manipulation detection code*.

Saat Anto hendak mengirimkan pesannya, dia harus membuat sidik jari dari pesan yang akan dikirim untuk Badu. Pesan (yang besarnya dapat bervariasi) yang akan di-*hash* disebut *pre-image,* sedangkan outputnya yang memiliki ukurannya tetap, disebut *hash-value* (nilai *hash*). Kemudian, melalui saluran komunikasi yang aman, dia mengirimkan sidik jarinya kepada Badu. Setelah Badu menerima pesan si Anto – tidak peduli lewat saluran komunikasi yang mana – Badu kemudian juga membuat sidik jari dari pesan yang telah diterimanya dari Anto. Kemudian Badu membandingkan sidik jari yang dibuatnya dengan sidik jari yang diterimanya dari Anto. Jika kedua sidik jari itu identik, maka Badu dapat yakin bahwa pesan itu utuh tidak diubah-ubah sejak dibuatkan sidik jari yang diterima Badu. Jika pesan pembayaran 1 juta rupiah itu diubah menjadi 10 juta rupiah, tentunya akan menghasilkan nilai *hash* yang berbeda.



Gambar III.7. Membuat sidik jari pesan

Fungsi *hash* untuk membuat sidik jari tersebut dapat diketahui oleh siapapun, tak terkecuali, sehingga siapapun dapat memeriksa keutuhan dokumen atau pesan tertentu. Tak ada algoritma rahasia dan umumnya tak ada pula kunci rahasia.

Jaminan dari keamanan sidik jari berangkat dari kenyataan bahwa hampir tidak ada dua pre-image yang memiliki *hash*-value yang sama. Inilah yang disebut dengan sifat *collision free* dari suatu fungsi *hash* yang baik. Selain itu, sangat sulit untuk membuat suatu pre-image jika hanya diketahui *hash*-valuenya saja.

Contoh algoritma fungsi *hash* satu arah adalah MD-5 dan SHA. *Message authentication code* (MAC) adalah salah satu variasi dari fungsi *hash* satu arah, hanya saja selain *pre-image,* sebuah kunci rahasia juga menjadi input bagi fungsi MAC.

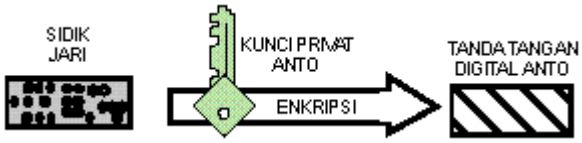
3.3.6 Tanda Tangan Digital

Badu memang dapat merasa yakin bahwa sidik jari yang datang bersama pesan yang diterimanya memang berkorelasi. Namun bagaimana Badu dapat merasa yakin bahwa pesan itu berasal dari Anto? Bisa saja saat dikirimkan oleh Anto melalui saluran komunikasi yang tidak aman, pesan tersebut diambil oleh Maman. Maman kemudian mengganti isi pesan tadi, dan membuat lagi sidik jari dari pesan yang baru diubahnya itu. Lalu, Maman mengirimkan lagi pesan beserta sidik jarinya itu kepada Badu, seolah-oleh dari Anto.

Untuk mencegah pemalsuan, Anto membubuhkan tanda tangannya pada pesan tersebut. Dalam dunia elektronik, Anto membubuhkan tanda tangan digitalnya pada pesan yang akan dikirimkan untuk Badu sehingga Badu dapat merasa yakin bahwa pesan itu memang dikirim oleh Anto. Sifat yang diinginkan dari tanda tangan digital diantaranya adalah:

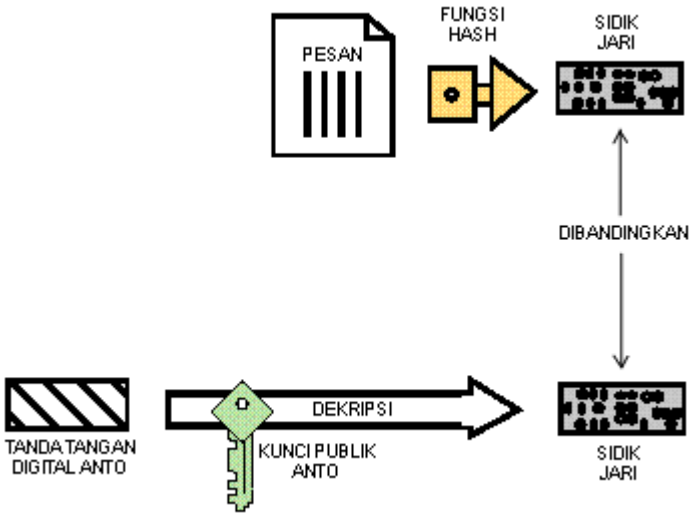
1. Tanda tangan itu asli (otentik), tidak mudah ditulis/ditiru oleh orang lain. Pesan dan tanda tangan pesan tersebut juga dapat menjadi barang bukti, sehingga penandatangan tak bisa menyangkal bahwa dulu ia tidak pernah menandatanganinya.
2. Tanda tangan itu hanya sah untuk dokumen (pesan) itu saja. Tanda tangan itu tidak bisa dipindahkan dari suatu dokumen ke dokumen lainnya. Ini juga berarti bahwa jika dokumen itu diubah, maka tanda tangan digital dari pesan tersebut tidak lagi sah.
3. Tanda tangan itu dapat diperiksa dengan mudah.
4. Tanda tangan itu dapat diperiksa oleh pihak-pihak yang belum pernah bertemu dengan penandatangan.
5. Tanda tangan itu juga sah untuk kopi dari dokumen yang sama persis.

Meskipun ada banyak skenario, ada baiknya kita perhatikan salah satu skenario yang cukup umum dalam penggunaan tanda tangan digital. Tanda tangan digital memanfaatkan fungsi *hash* satu arah untuk menjamin bahwa tanda tangan itu hanya berlaku untuk dokumen yang bersangkutan saja. Bukan dokumen tersebut secara keseluruhan yang ditandatangani, namun biasanya yang ditandatangani adalah sidik jari dari dokumen itu beserta *timestamp*-nya dengan menggunakan kunci privat. *Timestamp* berguna untuk menentukan waktu pengesahan dokumen.



Gambar III.8. Pembuatan tanda tangan digital

Keabsahan tanda tangan digital itu dapat diperiksa oleh Badu. Pertama-tama Badu membuat lagi sidik jari dari pesan yang diterimanya. Lalu Badu mendekripsi tanda tangan digital Anto untuk mendapatkan sidik jari yang asli. Badu lantas membandingkan kedua sidik jari tersebut. Jka kedua sidik jari tersebut sama, maka dapat diyakini bahwa pesan tersebut ditandatangani oleh Anto.



Gambar III.9. Pemeriksaan keabsahan tanda tangan digital

3.3.7 Masalah Pertukaran Kunci Publik

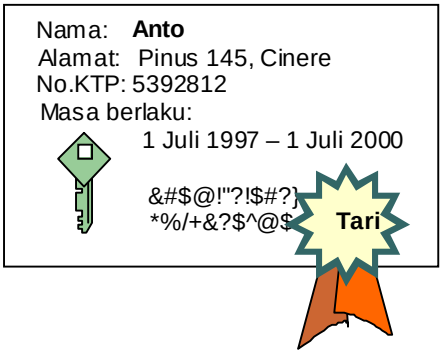
Anto hendak mengirimkan Badu suatu dokumen rahasia. Jika mereka belum pernah bertemu sebelumnya, tentu Badu harus mengirimkan kunci publiknya kepada Anto agar Anto dapat melakukan enkripsi yang pesannya hanya dapat dibuka oleh Badu. Demikian juga pula sebaliknya, Anto harus mengirimkan kepada Badu kunci publiknya agar Badu dapat memeriksa keaslian tanda tangan Anto pada pesan yang dikirim. Dengan cara ini Anto dapat memastikan pesan itu sampai ke tujuannya, sedangkan Badu dapat merasa yakin bahwa pengirim pesan itu adalah Anto.

Masalah yang muncul adalah bagaimana mereka dapat saling bertukar kunci dengan aman? Bisa saja di tengah pertukaran kunci-kunci publik milik Anto dan Budi itu diganti dengan kunci publik milik Maman. Dengan begitu Maman dengan bebas dapat menyadap dan mengubah seluruh informasi. Inilah suatu contoh dari *man-in-the-middle attack.*

Anto dan Badu harus sama-sama yakin bahwa kunci-kunci publik yang mereka dapatkan benar-benar otentik. Mereka bisa mendapatkannya dari sesorang yang dipercaya, Tari misalnya. Setiap anggota jaringan diasumsikan telah memiliki saluran komunikasi pribadi yang aman dengan Tari. Saluran inilah yang dimanfaatkan untuk mengirim kunci publik Badu ke Anto (dan sebaliknya). Tari menjadi penjamin keabsahan kunci jika Anto dan Badu sebelumnya tidak pernah bertukar kunci publik. Skenario ini tetap membutuhkan kunci-kunci kriptografi lagi (baik itu kunci simetris ataupun kunci asimetris) untuk pengamanan saluran komunikasi antara Tari dengan Anto atau Badu.

3.3.8 Sertifikat Digital

Masalah di atas dapat dipecahkan dengan penggunaan sertifikat digital. Tari tidak lagi setiap saat menjadi penukar kunci, namun Tari cukup menandatangani kunci publik milik setiap orang di jaringan tersebut. Sebenarnya dalam sertifikat tersebut tak hanya berisi kunci publik, namun dapat berisi pula informasi penting lainnya mengenai jati diri pemilik kunci publik, seperti misalnya nama, alamat, pekerjaan, jabatan, perusahaan dan bahkan *hash* dari suatu informasi rahasia. Semua orang mempercayai otoritas Tari dalam memberikan tanda tangan, sehingga orang-orang dalam jaringan itu merasa aman menggunakan kunci publik yang telah ditandatangani Tari.

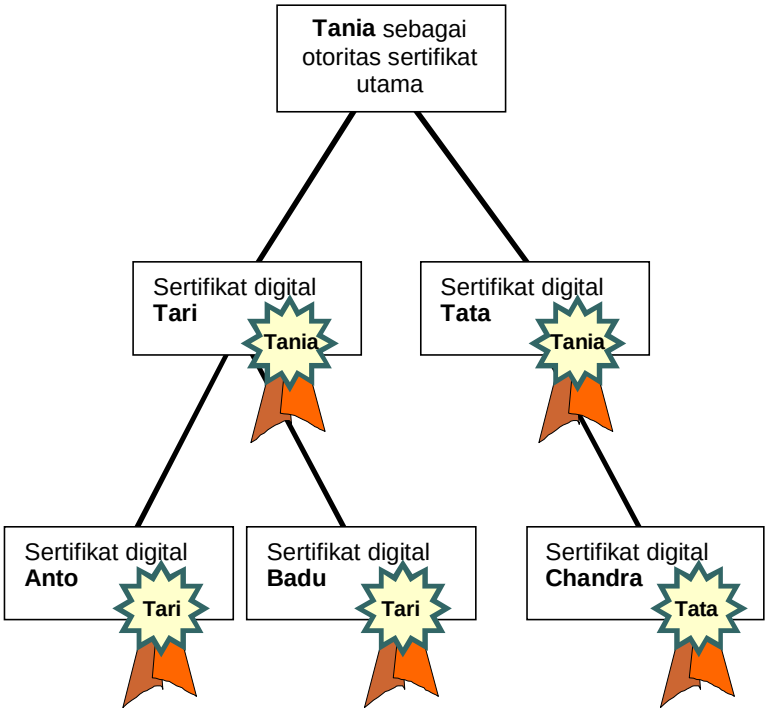


Gambar III.10. Contoh sertifikat digital

Jika Maman berhasil mencuri sertifikat digital yang dipertukarkan antara Anto dan Badu, serta menggantinya dengan sertifikat digital milik dirinya sendiri, maka Anto dan Badu dapat segera melihat bahwa sertifikat digital yang diterimanya bukan ‘lawan bicara’ yang semestinya.

Bagaimana jika Chandra – yang berada di luar jaringan Tari – hendak berkomunikasi dengan Anto? Chandra memiliki juga sertifikat, tetapi tidak ditandatangani oleh Tari, melainkan oleh Tata, seseorang yang dipercaya dalam jaringan tempat Chandra berada. Tari dan Tata adalah otoritas sertifikat (*certificate authority*), yaitu pihak-pihak yang berwenang memberikan sertifikat. Namun Anto tidak mengenal dan tidak mempercayai Tata. Masalah ini dapat diselesaikan jika ada otoritas sertifikat (OS) yang kedudukannya lebih tinggi dari Tata dan Tari – katakanlah Tania. Tania memberikan pengesahan kepada Tata dan Tari. Jadi ada hirarki dari sertifikat digital. Jika Tania berada pada kedudukan hirarki yang paling tinggi, maka Tania disebut otoritas sertifikat utama (*root certificate authority*).

Anto mempercayai tanda tangan Tari. Namun karena Tari sendiri keberadaannya disahkan oleh Tania, tentunya Anto harus mengakui otoritas Tania. Jika Tania memberikan pengesahan kepada OS lain dibawahnya, seperti Tata, maka dengan merunut struktur hirarki percabangan OS, Anto dapat memeriksa kebenaran sertifikat digital milik Chandra yang disahkan oleh Tata.



Gambar III.11. Contoh hirarki otoritas sertifikat digital

Serangan terhadap sistem yang memiliki pengamanan dengan sertifikat digital sulit dilakukan. Jelas Edi tidak mendapatkan apa-apa kalaupun ia memainkan ulang percakapan antara Anto dan Chandra. Edi membutuhkan kunci privat untuk bisa membuka pesan-pesan yang dipertukarkan, padahal kunci privat itu tidak ada di dalam sertifikat digital.

Penukaran sertifikat digital Chandra dengan sertifikat digital Maman akan segera diketahui, karena sertifikat digital itu pasti berbeda. Sedangkan jika sertifikat yang dipertukarkan antara Chandra dan Anto tidak diganti, tetapi yang diganti oleh Maman adalah pesan yang dipertukarkan, maka tentu ada ketidakcocokan dalam pemeriksaan tanda tangan digital.

Secara teoritis keunggulan dari tanda tangan digital adalah kemampuan untuk melakukan proses otentikasi secara *off-line*. Pemeriksa cukup memiliki kunci publik dari OS utama untuk mengetahui sah-tidaknya kunci publik dari lawan bicaranya. Selain itu untuk meningkatkan keamanan, kunci publik OS utama bisa saja diintegrasikan dalam program aplikasi. Namun kenyataannya, karena ada kemungkinan sertifikat digital tersebut hilang, tercuri atau identitas pemilik sertifikat berubah (perubahan alamat surat elektronik atau nomor KTP misalnya), maka sertifikat digital perlu diperiksa keabsahannya dengan melihat daftar sertifikat terbatalkan (*certificate revocation list*) yang disimpan oleh OS.

3.3.9 Tanda Tangan Pesan Ganda

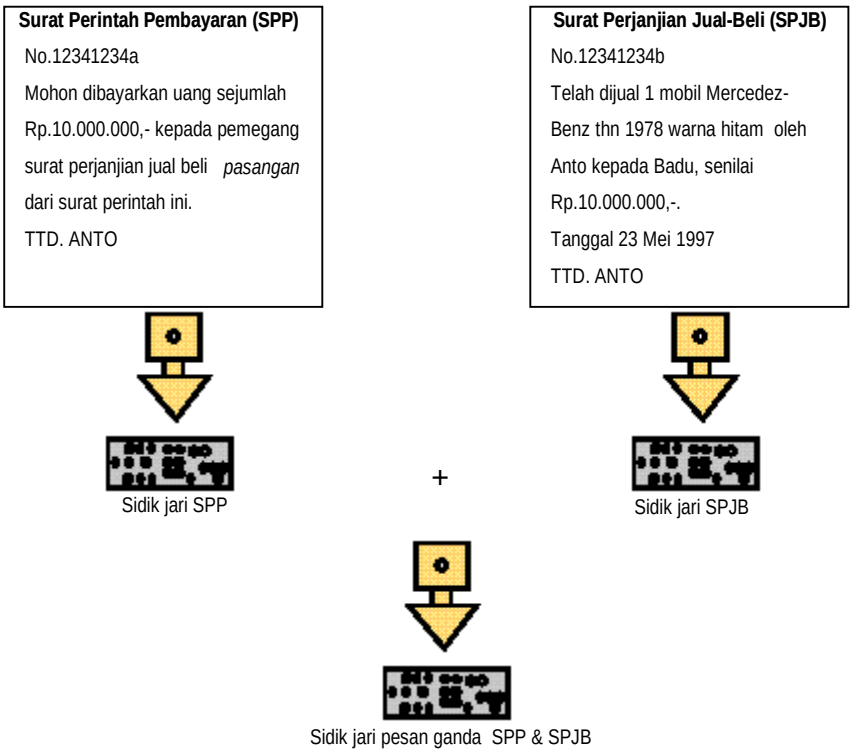
Andaikan Anto membuat perjanjian jual-beli dengan Badu. Untuk masalah pembayaran, Anto menginstruksikan bank untuk memberikan kepada Badu sejumlah uang sesuai dengan perjanjian jual-beli, namun Anto tidak ingin agar bank mengetahui isi perjanjian jual-beli itu.

Anto membuat sidik jari dari SPP (yaitu *Hash*(SPP)) dan sidik jari SPJB (yakni *Hash*(SPJB)). Kemudian, Anto membuat sebuah sidik jari baru dari gabungan kedua sidik jari sebelumnya (*Hash* ((*Hash*(SPP) + *Hash*(SPJB))). Hasil *hash* tersebut dinamakan sidik jari pesan ganda SPP & SPJB.

Anto menyerahkan surat perjanjian jual belinya kepada Badu. Selain itu Anto juga menyerahkan surat perintah pembayaran beserta sidik jari pesan ganda SPP & SPJB kepada bank. Saat Badu ingin mengambil uang di bank, Badu membuat sidik jari dari surat perjanjian jual beli (SPJB). Badu menyerahkan sidik jari SPJB kepada bank.

Bank membuat sidik jari dari surat perintah pembayaran (SPP).

Bank menggabungkan sidik jari SPP dengan sidik jari SPJB yang diterimanya dari Badu, kemudian meng-*hash*-nya sehingga dihasilkan sidik jari pesan ganda SPP & SPJB. Jika sidik jari pesan ganda SPP & SPJB yang baru dibuat itu sama dengan yang telah diberikan oleh Anto, maka bank menjalankan kewajibannya kepada Badu.



Gambar III.12. Pembuatan sidik jari pesan ganda

Jika sidik jari pesan ganda SPP & SPJB dienkripsi dengan kunci privat Anto, maka akan menjadi tanda tangan pesan ganda (*dual-signature*) Anto untuk kedua perjanjian tersebut [ViMa 97].

3.3.10 Protokol Pembagian Rahasia

Jika Anto memiliki rahasia, ia dapat memberikan 'separuh' rahasia itu kepada Badu dan 'separuh' rahasia itu kepada Chandra. Badu, yang menerima paruh pertama rahasia Anto, tidak bisa mengetahui apa isi rahasia itu. Demikian pula dengan Chandra. Namun, jika Badu dan Chandra menggabungkan potongan-potongan rahasia itu, maka akan tergambar rahasia Anto. Pembagian rahasia (*secret splitting)* dapat dilakukan dengan cara:

1. Anto membuat seuntai string acak R yang panjangnya sama dengan pesan rahasia M.
2. Anto melakukan operasi XOR antara M dengan R, sehingga menghasilkan S.
3. Anto memberikan R kepada Badu dan S kepada Chandra
4. Jika Badu dengan Chandra bertemu, maka mereka sanggup mendapatkan pesan rahasia M dengan cara melakukan operasi XOR antara S dengan R.

3.3.11 Protokol Komitmen-Bit

Protokol ini bermanfaat kalau misalnya Anto hendak membuat suatu pernyataan atau komitmen (katakanlah suatu string binari 1000), namun Anto tak ingin agar Badu mengetahui isi pernyatan tersebut sebelum saatnya. Badu harus merasa yakin bahwa Anto pada saatnya nanti, benar-benar mengeluarkan isi pernyataan yang sebenarnya saat melakukan komitmen, dan tidak mengeluarkan pernyataan yang sudah diubah (misalnya mengubah string tadi menjadi 1001). Ada beberapa jenis protokol komitmen-bit, namun di bawah ini hanya dijelaskan salah satu diantaranya, yakni dengan fungsi *hash* satu arah:

1. Anto membuat dua buah string secara acak, yakni R_1 dan R_2
2. Anto menggabungkan kedua string acak itu ke dalam pernyataannya (b) yang akan dikomitmenkan menjadi (R_1, R_2, b)
3. Anto menghitung *hash* dari gabungan string itu, *Hash*(R_1, R_2, b).
4. Anto kemudian mengirimkan *hash* tersebut beserta R_1 kepada Badu. Badu akan menyimpannya untuk pemeriksaan nanti.
5. Jika sudah tiba saatnya untuk menunjukkan pernyataannya, Anto memberikan seluruh string (R_1, R_2, b) kepada Badu.

6. Badu memeriksa fungsi *hash* dari (R_1, R_2, b). Jika cocok dengan *hash* yang diperiksanya dulu, maka pernyataan Anto tidak diubah.

3.3.12 Tanda Tangan Buta (*Blind Signature*)

Badu disodori 100 amplop tertutup oleh Anto. Amplop itu berisi secarik pesan dan kertas karbon. Badu membuka 99 amplop secara acak. Jika seluruh amplop yang dibuka ternyata berisi pesan yang mirip, maka Badu dapat merasa bahwa amplop ke-100 juga berisi pesan yang mirip pula. Namun, jika satu saja dari 99 amplop tadi ada yang isi berbeda dari yang lain, maka Badu dapat mencurigai bahwa isi amplop ke-100 bisa saja juga tidak mirip dengan isi ke-98 amplop lainnya.

Dalam kasus dimana ternyata ke-99 amplop yang dibuka secara acak tadi berisi pesan yang mirip, maka dengan keyakinan yang cukup tinggi Badu berani menandatangani amplop terakhir yang belum dibuka. Tanda tangan Badu akan menembus amplop dan kertas karbon, sehingga pesan dalam amplop akan tertandatangani oleh Badu. Badu kurang lebih tahu apa isi pesan di amplop ke-100 itu. Protokol tanda tangan buta (*blind signature*) bekerja sebagai berikut:

1. Anto ‘mengalikan’ dokumen (yang akan ditandatangani) dengan sebuah faktor pembuta.
2. Anto mengirimkan dokumen itu kepada Badu
3. Badu menandatangani dokumen itu
4. Badu mengembalikan dokumen yang sudah ditandatangani tadi kepada Anto
5. Anto membaginya dengan faktor pembuta, sehingga mendapatkan dokumen yang asli sudah tertandatangani oleh Badu.

3.3.13 Protokol Uang Digital

3.3.13.1 Deskripsi

Berdasarkan beberapa teori penunjang di atas, maka dapatlah dibangun suatu protokol untuk uang digital. David Chaum, memiliki beberapa paten atas protokol uang digital yang diciptakannya. Berikut ini dijelaskan salah satu protokol uang digital:

- Anto menyiapkan n lembar uang dengan nilai tertentu. Setiap uang diberi nomor seri acak X yang cukup panjang, sehingga kemungkinan 2 bilangan acak sama kecil sekali. Dalam setiap uang juga ada n (I_1, I_2, …, I_n) string identifikasi yang berguna untuk memberikan informasi mengenai pemilik uang, yakni Anto. Anto kemudian memecah tiap-tiap string identitas diri itu tadi menjadi dua bagian dengan menggunakan protokol pemecahan rahasia. Lantas Anto melakukan bit-komitmen pada setiap pecahan. Contoh uang yang disiapkan adalah:

Nilai: Rp.1.000,-

Nomor seri acak: X

String identitas: $I_1 = (I_{1L}, I_{1R})$

$I_2 = (I_{2L}, I_{2R})$

. . . .

$I_n = (I_{nL}, I_{nR})$

- Anto memasukkan uang itu kedalam yang juga disisipi kertas karbon amplop (mengalikan uang dengan faktor pembuta), lalu memberikannya kepada bank.
- Bank akan meminta Anto untuk membuka n - 1 amplop itu secara acak. Bank memeriksa apakah semua uang tersebut memiliki nilai yang sama. Bank juga meminta kepada Anto untuk membuktikan kejujuran dirinya saat menuliskan string identifikasi pada uang itu, dengan cara menggabungkan pasangan-pasangan string identifikasi.
- Jika bank merasa bahwa Anto tidak melakukan kecurangan, maka bank akan menandatangani uang terakhir yang masih di dalam amplop itu dan menyerahkannya kepada Anto. Tanda tangan bank akan menembus amplop dan kertas karbon sehingga uang di dalamnya tertandatangani.
- Anto membuka amplop. Uang siap dipakai.
- Anto menyerahkan uang kepada Badu. Badu sebagai penerima uang, akan memeriksa apakah tanda tangan bank pada uang itu absah.
- Badu akan menyuruh Anto untuk membuka salah satu sisi dari setiap string identifikasi di setiap uang dengan cara memberikan string pemilih sepanjang n-bit. Artinya, jika string pemilih itu $b_1, b_2, \ldots, b_n$ maka Anto harus membuka sisi kiri atau kanan dari I_i, tergantung apakah b_i itu 0 atau 1.
- Setelah itu Badu membawa uang tersebut ke bank. Bank akan memeriksa apakah nomor seri uang tersebut sudah pernah diterima oleh bank. Kalau belum ada, maka uang tersebut dinyatakan sah.
- Jika nomor seri uang itu sudah pernah diterima oleh bank, maka bank akan memeriksa string identitas yang sudah terbuka pada uang itu dan membandingkannya dengan string identitas pada uang dengan nomor seri sama yang pernah diterima bank sebelumnya. Jika ternyata string identitas itu sama, maka berarti Badu yang menggandakan uang tersebut. Namun jika berbeda, maka berarti Anto yang menggandakan uang digital tersebut.

3.3.13.2 Pembelanjaan Ganda

Jika Anto menggandakan uang digitalnya lalu menggunakan uang digital yang sama itu

dua kali, bank dapat mendeteksinya meskipun Badu tidak bisa. Badu memang ‘membuka’ identitas uang, namun hanya separuh-separuh. Kalau uang digital itu pernah diberikan Anto kepada Chandra, maka tentu Chandra juga pernah ‘membuka’ separuh identitas uang digital tadi secara acak. Nah, kemungkinan bahwa proses pembukaan identitas oleh Badu dan Chandra itu sama (maksudnya sama urutan pembukaannya, misalnya kiri-kiri-kanan-kiri-kanan, dan seterusnya) adalah 1 per 2n. Andaikan n cukup besar, katakanlah 16 saja, maka kemungkinan Badu dan Chandra secara acak membuka paruhan identitas dengan urutan sama adalah 1 : 65536. Artinya, jika Anto memberikan uangnya kepada dua orang yang berbeda, kemungkinan besar paruhan identitas yang dibuka juga berbeda. Jika saat otentikasi uang digital oleh bank ditemukan bahwa ada uang digital dengan nomor seri sama yang telah diuangkan, dan paruhan identitasnya berbeda, maka kemungkinan besar Anto menyerahkan uang digital yang sama kepada dua orang yang berbeda.

Sedangkan apabila Badu menguangkan uang digital yang sama dua kali, karena paruhan identitas dari uang digital yang diotentikasi itu sama persis dengan yang sudah tercatat, maka kemungkinan besar uang itu diberikan Anto kepada orang yang sama. Badulah yang ketahuan menguangkan uang digital yang sama dua kali. Penggunaan uang digital yang sama dua kali dikenal dengan istilah pembelanjaan ganda (*double spending*).

3.3.14 Panjang Kunci

3.3.14.1 Panjang Kunci Simetris

Meskipun ada beberapa cara bagi seorang kriptoanalis untuk memecahkan pesan rahasia, namun cara yang cukup umum dilakukan adalah dengan melakukan *brute-force attack*. Dengan cara ini, seorang penyerang mencoba seluruh kemungkinan kunci yang ada, sampai menemukan sebuah kunci yang jika dipergunakan untuk mendekripsi pesan yang disandikan akan memunculkan suatu pesan yang bermakna. Tentunya cara ini bermanfaat hanya jika sudah diketahui algoritmanya, namun tidak diketahui kuncinya apa.

PIN 5 digit berarti biasanya ada 100.000 kombinasi. Kelihatannya cukup, namun sebenarnya kurang. Dengan sebuah komputer pribadi saja bisa dengan mudah diselesaikan. Salah satu pencegahannya adalah dengan pembatasan seberapa banyak pemakai dapat mencoba memasukkan PIN. Biasanya dibatasi tiga kali.

Berikut ini diberikan contoh dari *brute-force attack* pada suatu algoritma ‘geser pada papan ketik QWERTY’:

Sandi *zsdyrtvstf* dicoba dengan kunci 3 menjadi *bjkwpqmjql*

Sandi *zsdyrtvstf* dicoba dengan kunci 2 menjadi *nkleqwzkwa*
Sandi *zsdyrtvstf* dicoba dengan kunci 1 menjadi *mastercard*

Ternyata kunci 1 cocok, karena dalam pesan yang disandikan itu mungkin ada transaksi yang menggunakan kartu kredit 'mastercard'. Dengan menggunakan kunci yang sama, kemudian penyerang berusaha mendekripsikan bagian-bagian lain dari pesan, mungkin berusaha mengambil nomor kartu kreditnya. Kunci itu juga dapat dipakai untuk keperluan lain, misalnya untuk melakukan penipuan (*spoofing*).

DES, sebuah algoritma simetris, memiliki panjang kunci 56-bit, artinya ada 2^{56} kemungkinan kunci. Sedangkan peraturan di Amerika Serikat yang akan diberlakukan pada tahun 1998 nanti akan melarang ekspor teknologi enkripsi lebih dari 40-bit. Sedangkan untuk keperluan dalam negeri Amerika Serikat, kunci 128-bit masih diizinkan penggunaannya [STAR97].

Tahun 1995, Michael Wierner merancang sebuah chip yang mengkhususkan diri untuk melakukan *brute-force attack* pada metoda enkripsi DES [SCHN96]. Chip tersebut dapat menemukan kunci rahasia dalam waktu rata-rata 3,5 jam dan kunci itu dijamin dapat ditemukan dalam waktu 7 jam. Harga pembuatannya adalah 1 juta dollar AS. Sesuai hukum Moore, setiap 18 bulan kemampuan komputer meningkat 2 kali lipat untuk harga yang sama. Maka, pada tahun 2000, harga chip itu hanya berkisar 100.000 dolar AS. Harga ini masih dalam jangkauan daya beli beberapa mafia kejahatan terorganisir. Karena itu, kini disarankan untuk menggunakan DES dengan kunci 112-bit.

Panjang kunci DES	**Jaminan waktu untuk menemukan kunci**
40-bit	0,4 detik
56-bit	7 jam
64-bit	74 jam 40 menit
128-bit	157.129.203.952.300.000 tahun

Tabel III-3. Serangan brute-force pada DES

Protokol keamanan SSL (*Secure Socket Layer*) pada Netscape Navigator menggunakan algoritma RC4 40-bit untuk enkripsi simetrisnya. Tahun 1995, Damien Doligez menjebolnya menggunakan 120 komputer Unix yang terhubung pada jaringan dalam waktu 8 hari [STAR97]. Dengan cara seperti ini, dijamin bahwa dalam 15 hari kunci itu pasti ditemukan.

Panjang kunci RC4	Jaminan waktu untuk menemukan kunci
40-bit	15 hari
56-bit	2.691,49 tahun
64-bit	689.021,57 tahun
128-bit	12.710.204.652.610.000.000.000.000 tahun

Tabel III-4. Serangan *brute-force* pada RC4

3.3.14.2 Panjang Kunci Asimetris

Sedangkan pada sistem enkripsi kunci publik-privat, yang memegang peranan dalam menjebol kunci privat adalah kesulitan mencari faktor prima bilangan yang sangat besar. Beberapa kunci yang dipergunakan 10 tahun lalu saja kini sama sekali tidak laik pakai seiring dengan perkembangan ilmu pengetahuan dan teknologi.

Kunci publik yang dimanfaatkan SSL adalah teknologi kunci publik 40-bit dari RSA, yang ternyata dapat dijebol dalam waktu 1,3 hari dengan 100 komputer menggunakan *brute-force attack* [DHMM 96].

Ronald Rivest, salah seorang penemu RSA, juga pernah menghitung bahwa untuk menemukan kunci RSA 512-bit dengan cara *brute-force attack* membutuhkan biaya 8,2 juta dollar AS [DALE96]. Untuk kasus tertentu, ini pun tidak aman. Kini perusahaan-perusahaan disarankan menggunakan kunci 2048 bit agar data aman sampai tahun 2015.

3.4 SMARTCARD

3.4.1 Sejarah dan Faktor yang Mempengaruhi Munculnya Smartcard

Sebenarnya *smartcard* bukan barang baru. Industri *smartcard* dimulai ketika Bull (perusahaan komputer dari Perancis) dan Motorola(perusahaan chip dari Amerika Serikat) merancang dan mengembangkan *smartcard* untuk pertama kalinya pada tahun 1977 untuk perusahaan perbankan Perancis, Cartes Bancaires [COAR97]. Perusahaan perbankan ini selalu mengalami kerugian yang disebabkan adanya penipuan dan pemalsuan kartu kredit. Kartu palsu itu dapat digunakan untuk membayar atau untuk mendapatkan layanan. Mereka mencari jalan keluar untuk mengatasi hal tersebut, yaitu dengan menggunakan *smartcard*. Selain itu mereka ingin mengurangi biaya pemroses cek. Ada juga faktor eksternal yang mendukung munculnya

smartcard. Faktor tersebut adalah mahalnya biaya telekomunikasi di Eropa karena biaya instalasi jaringan yang mahal. Sedangkan di Amerika, biaya telepon murah, transaksi secara on-line melalui jalur telepon tidak menjadi masalah. Transaksi secara *on-line* ini dilakukan dengan kartu magnetis dengan menghubungkan *terminal* (*reader*) dengan *database* pusat untuk mengetahui apakah kartu tersebut valid sebelum transaksi dilakukan. Dengan *smartcard*, transaksi dapat dilakukan secara *off-line*. Untuk mengetahui ke-valid-an suatu kartu cukup hanya dengan memeriksa apakah Kode PIN yang dimasukkan oleh pemegang kartu sesuai dengan Kode PIN yang tertulis di kartu, tidak perlu menghubungi database pusat.

Perkembangan *smartcard* tidak terlepas faktor perkembangan teknologi mikroprosesor, peningkatan kapasitas memori serta perkembangan Internet. Mikroprosesor adalah *chip* yang menentukan kekuatan komputasi dan fleksibilitas. Dengan ukuran yang hanya sebesar 1 ruas jari, *chip* yang sekarang ada mempunyai kekuatan komputasi yang sama dengan komputer Macintosh 2 pada tahun 1988 ! [COAR97] Kekuatan komputasi ini menentukan seberapa cepat dan jumlah komputasi yang bisa dilakukan. Kekuatan komputasi yang besar memungkinkan penghitungan yang kompleks dapat dilakukan. Penghitungan yang kompleks biasanya digunakan untuk memperkuat keamanan misalnya untuk melakukan enkripsi data.

Faktor kedua yaitu meningkatnya kapasitas memori dalam arti makin murahnya harga memori (untuk kapasitas yang sama) sehingga lebih banyak data dan aplikasi yang dapat disimpan di dalam kartu.

Faktor yang terakhir adalah perkembangan aplikasi bisnis di Internet. Teknologi Internet merupakan teknologi yang paling pesat perkembangannya. Pertumbuhannya diperkirakan 10-15% per bulan dan mempunyai potensi pasar sebesar 600 milyar dolar. Para pelaku bisnis pun mulai memandang Internet sebagai sarana bisnis. Namun masalah timbul ketika kerahasian data lebih di utamakan. Internet adalah milik publik, semua orang bisa menggunakan jaringan ini untuk saling menukar informasi. Tetapi di lain pihak ada orang-orang yang memanfaatkan situasi ini dengan tujuan untuk menguntungkan diri sendiri, atau bahkan untuk mengganggu orang lain. Misalnya dengan menyadap informasi, kemudian mengubah informasi tersebut dan diteruskan ke pihak penerima. *Smartcard* menawarkan solusi dengan fasilitas enkripsi data, sebelum data tersebut dikirim melalui jaringan publik.

3.4.2 Pengertian Smartcard

Smartcard adalah kartu plastik yang berukuran sama dengan kartu kredit yang di dalamnya terdapat *chip* silikon. *Chip* merupakan *integrated circuit* yang terdiri dari prosesor dan memori. Chip, seperti layaknya CPU (*Central Processing Unit*) di komputer, bertugas

melaksanakan perintah dan menyediakan *power* ke kartu. *Smartcard* merupakan pengembangan dari kartu magnetis, namun berbeda dengan kartu magnetis yang hanya dipakai sebagai tempat penyimpanan data, *smartcard* mempunyai kemampuan untuk memroses dan menginterpretasikan data, serta menyimpan data tersebut secara aman. Apalagi dengan perkembangan algoritma kriptografi, data yang disimpan akan dienkripsi terlebih dahulu, sehingga tidak mudah dibaca oleh pihak yang tidak berwenang/berhak. Hal ini akan mempersulit pemalsuan kartu. Selain perbedaan dengan adanya *chip*, *smartcard* memiliki kapasitas memori yang lebih besar dari kartu magnetis.

3.4.3 Jenis Memori pada Smartcard

Secara umum ada 3 jenis memori yang digunakan :

1. ROM (*Read Only Memory*), berfungsi untuk menyimpan program utama dan sifatnya permanen.
2. RAM (*Random Access Memory*), berfungsi untuk menyimpan data sementara ketika proses sedang berjalan atau hasil penghitungan selama mengeksekusi perintah. Data yang disimpan di dalamnya akan hilang begitu kartu dicabut (*power* hilang).
3. EEPROM (*Electrically Erasable Programmable Read Only Memory*), berfungsi untuk menyimpan program dan data yang sewaktu-waktu bisa diubah. Seperti halnya hard disk pada komputer, jenis memori ini akan tetap menyimpan data meskipun tidak ada *power*(permanen).

3.4.4 Tipe-tipe Smartcard

Ada 2 tipe *smartcard*, yaitu *intelligent smartcard* yang mempunyai mikroprosesor dan menawarkan kemampuan membaca, menulis dan melakukan penghitungan, seperti mikrokomputer kecil. Yang kedua adalah kartu memori yang tidak mempunyai mikroprosesor dan digunakan hanya untuk tempat menyimpan. Kartu memori menggunakan *security logic* untuk mengatur akses ke memori.

3.4.5 Standar-standar Smartcard

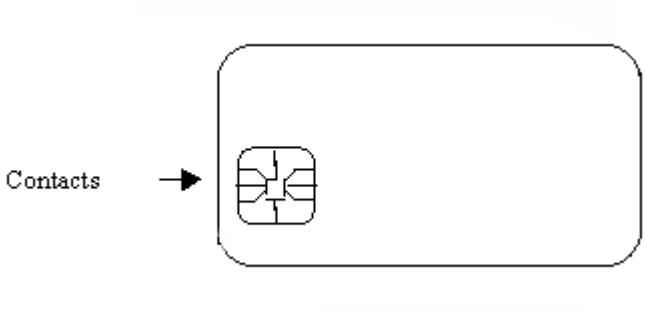
Seperti layaknya setiap teknologi, ada beberapa standar untuk *smartcard*. Dengan mengetahui standar tersebut, pemrogram dapat membangun aplikasi yang nantinya bisa beroperasi di lingkungan yang yang memenuhi standar yang sama. Standar tersebut dibagi menjadi 2 bagian besar, yaitu standar horisontal dan standar vertikal. standar horisontal dapat

digunakan oleh semua aplikasi, sedangkan standar vertikal lebih spesifik ke sistem. Berikut ini rincian standar-standar tersebut:

1. Standar Horisontal , meliputi:
   - ISO 7816 - menggambarkan antarmuka pada level bawah dari *smartcard*. Pada level ini, data byte dikirim antara *reader* dan kartu.
   - PC/SC - standar untuk berkomunikasi dengan *smartcard* yang terhubung dengan mesin 32-bit, yaitu Win3.1/Win95/NT.
   - OCF - antarmuka Java untuk berkomunikasi dengan *smartcard* dalam lingkungan Java.
   - JavaCard - mendefinisikan operating sistem berbasis Java untuk *smartcard*.
   - ISO 7816 didefinisikan oleh Internasional Standard Organization, berisi sekumpulan standar yang harus dipenuhi oleh *smartcard*. ISO 7816 berisi :
     - *Physical characteristics (part 1)*
     - *Dimensions and location of the contacts (part 2)*
     - *Electronic signals and Transmission protocols (part 3)*
     - *Inter-industry commands for interchange (part 4)*
     - *Application identifiers (Part 5)*
     - *Inter-industry data elements (Part 6)*
     - *Inter-industry commands for SCQL (Part 7)*
2. Standar Vertikal, meliputi:
   - Mondex - digital cash yang hanya menggunakan *smartcard*.
   - VisaCash - kartu debit
   - Proton - bentuk lain dari E-cash
   - MPCOS-EMV - kartu multi fungsi yang memungkinkan pemrogram mengimplementasikan tipe *currency* atau *token* yang didefinisikan sendiri.

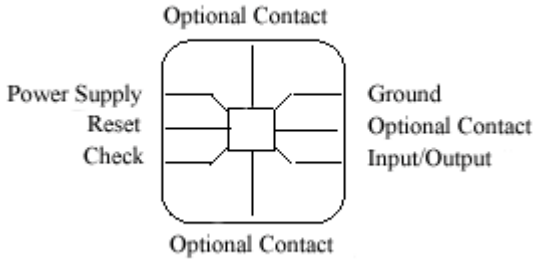
3.4.6 Karakter Fisik

Diagram berikut menggambarkan karakter fisik dari *smartcard*, yang didefinisikan dalam ISO 7816 bagian pertama.



Gambar III.13. Karakter fisik dari *smartcard*

Pada umumnya, *smartcard* tidak berisi *power supply*, *display* atau *keyboard*. *Smartcard* berinteraksi dengan dunia luar dengan menggunakan antarmuka komunikasi serial melalui 8 titik kontak. Ukuran dan letak dari kontak tersebut didefinisikan didalam ISO 7816, bagian kedua. Gambar berikut menunjukkan kontak di dalam *smartcard*.



Gambar III.14. Delapan titik kontak

Smartcard dimasukan ke dalam perangkat penerima kartu (*Card Acceptance Device/CAD*), yang dapat dihubungkan dengan komputer. Istilah lain yang digunakan untuk CAD adalah *terminal, reader* dan IFD (*interface device*/perangkat antarmuka). Semuanya mempunyai fungsi dasar, yaitu menyediakan *power* ke kartu dan membangun hubungan pertukaran data.

3.4.7 Komunikasi antara Smartcard dan Aplikasi

Aplikasi berkomunikasi dengan *reader* (yang kemudian akan berkomunikasi dengan *smartcard*) menggunakan protokol yang standar, yaitu protokol *International Standard Organization* (ISO) 7816. *Smartcard* merupakan *personal hardware* yang harus berkomunikasi dengan perangkat lainnya untuk mengakses perangkat *display* atau jaringan. Kartu dapat dimasukkan ke dalam *reader*, yang biasanya disebut *terminal*, termasuk *terminal* yang beroperasi menggunakan RF (*radio frequencies*).

Smartcard dapat berkomunikasi dengan *reader* dengan 2 cara :

1. *contact smartcard* - koneksi dibuat ketika *reader* bersentuhan dengan chip yang ada di *smartcard*.
2. *contactless smartcard* – dapat berkomunikasi melalui antena, mengurangi keperluan untuk memasukkan dan mengambil kartu. Dengan *contactless*, yang harus dilakukan hanya mendekatkan *smartcard* ke *reader*, dan selanjutnya *smartcard* akan berkomunikasi. *Contactless smartcard* dapat digunakan di dalam aplikasi dimana pemasukan/penarikan kartu tidak praktis dan pertimbangan kecepatan.

Dalam rangka mengembangkan aplikasi berbasis *smartcard*, perlu beberapa perangkat: *smartcard reader*, perangkat lunak untuk berkomunikasi dengan *reader* maupun perangkat lunak yang berkomunikasi dengan kartu dan *smartcard*.

Reader menyediakan *path* untuk aplikasi, untuk mengirim dan menerima *command* dari kartu. Ada beberapa tipe *reader* di pasaran. Yang paling banyak dijumpai adalah serial, PCCard dan keyboard model. Serial *reader* berhubungan dengan *serial port* pada komputer. Kode yang disediakan juga mendukung PCCard *reader*, beberapa laptop dilengkapi dengan slot PCCard yang *built-in*.

3.4.8 Format APDU

Smartcard tidak berarti tanpa adanya *smartcard reader*, yang berfungsi sebagai perantara komunikasi antara *smartcard* dengan peralatan lain seperti komputer. Komputer membaca atau menulis data melalui *smartcard reader*, kemudian *smartcard reader* mengubah perintah membaca/menulis tersebut ke dalam bahasa yang dimengerti *smartcard*.

Masing-masing perusahaan menyediakan protokol yang berbeda untuk berkomunikasi dengan *reader*. Komunikasi dengan *smartcard* berdasarkan format APDU(*Application Protocol Data Unit*).

APDU merupakan unit dasar untuk pertukaran paket di dalam *smartcard*. Komunikasi antara kartu dengan *reader* dilakukan dengan APDU. APDU dinyatakan sebagai data paket yang berisi perintah lengkap atau *response* yang lengkap dari kartu. Untuk menyediakan fungsionalitas seperti ini, APDU mendefinisikan struktur yang didefinisikan dalam beberapa dokumen ISO 7816.

ISO mendefinisikan standar bagaimana aplikasi berkomunikasi dengan *smartcard*. Sayangnya, ISO tidak mendefinisikan standar untuk berkomunikasi dengan *reader*. Sehingga untuk mengirim perintah ke kartu, pertama pemrogram perlu menemukan *command* yang

dimengerti oleh kartu, kemudian membungkus *command* tersebut dengan ISO *command* packet, kemudian dibungkus lagi dengan pembungkus yang diperlukan oleh *reader*.

Model *master-slave* digunakan di mana *smartcard* selalu memainkan posisi yang pasif. Dengan kata lain, *smartcard* selalu menunggu perintah APDU dari *terminal*. Kemudian *smartcard* mengeksekusi aksi yang ditentukan di dalam APDU dan mengembalikannya ke *terminal* dengan *response* APDU. *Command* APDU dan *response* APDU dipertukarkan antara kartu dan *terminal*.

Gambar adalah format *command* dan *response* APDU. Struktur APDU didefinisikan dalam ISO 7816-4.

Command APDU						
Mandatory Header				*Conditional Body*		
CLA	INS	P1	P2	Lc	Data field	Le

Gambar III.15. *Command* APDU

Header terdiri dari 4 field: *class* (CLA), perintah (INS) serta parameter 1 dan 2 (P1 dan P2). Masing-masing *field* berukuran 1 byte :

- CLA: *class* byte. Di beberapa *smartcard* digunakan untuk mengidentifikasikan aplikasi.
- INS: *Instruction* byte. Byte ini menyatakan kode instruksi/perintah.
- P1 dan P2: Parameter byte. Menyediakan qualifikasi lebih lanjut untuk perintah APDU.
- *Conditional body* terdiri dari 3 *field*, yaitu Lc, datafield dan Le.
- Lc menyatakan jumlah byte di dalam *data field* dari *command* APDU,
- *Data field* menyatakan data yang diperlukan oleh *command* APDU.
- Le menyatakan jumlah maksimal dari byte yang diharapkan di dalam *data field* dari *response* APDU.

Response APDU		
Conditional Body	*Mandatory Trailer*	
Data field	SW1	SW2

Gambar III.16. *Response* APDU

- Response APDU terdiri dari *conditional body* dan *mandatory trailer*.
- *Conditional body* berisi *data field* yang menyatakan data yang diperlukan oleh *response* APDU.
- *Mandatory trailer* terdiri dari *status byte* SW1 dan SW2 menyatakan status proses dari *command* APDU di dalam kartu.

3.4.9 Penggunaan Smartcard

Keuntungan menggunakan smartcard :

1. Lebih handal daripada *kartu magnetik* (kartu magnetik)

Kehandalan dari smartcard disebabkan oleh proteksi terhadap keamanan data yang disimpan. Keamanannya tidak hanya tergantung pada chip, namun juga keseluruhan sistem termasuk aplikasi serta proses pembuatan dari *smartcard* itu sendiri. Chip menjamin keamanan data yang disimpan di dalam *smartcard* disebabkan adanya mekanisme enkripsi sehingga tidak mudah dibaca oleh pihak yang tidak berwenang. Untuk membuat aplikasi *smartcard* juga perlu rancangan *security* terhadap aplikasi itu sendiri, misalnya aplikasi dibuat agar hanya pihak yang berwenang yang dapat menggunakan *smartcard* dan aplikasi yang ada di dalamnya. Selain keamanan chip dan aplikasi, keamanan terhadap proses pembuatan *smartcard*, terutama pembuatan mikroprosesor juga perlu dipertimbangkan. Kebanyakan dari perusahaan pembuat chip menyembunyikan detail dari rangkaian mikroprosesor, tidak terkecuali pada *customer*-nya. Dalam hal ini ada 3 fase, yaitu *designed-in security*, kontrol terhadap informasi dan proses pembuatan dan pemasaran [PAMI98]. *Designed-in security* meliputi perancangan dari chip mikroprosesor. Kontrol terhadap informasi meliputi bagaimana informasi yang rahasia disimpan. Sedangkan proses pembuatan dan pemasaran lebih banyak memperhatikan aspek keamanan dari chip tersebut, misalnya tempat penyimpanan yang aman.

2. Lebih banyak menyimpan informasi daripada *kartu magnetik.*

Kapasitas memori dari *smartcard* lebih besar dibanding kartu magnetik. Kartu magnetik hanya memiliki memori sebesar 140 byte [ARCL97] yang hanya cukup untuk menyimpan kode PIN dan data untuk *login* ke dalam *server-based system*. Oleh karena itu, transaksi lebih banyak dilakukan secara *on-line*. Sedangkan *smartcard* mempunyai ukuran memory bermacam-macam, misalnya dari 1 Kbyte (CP1 dari ASE(Alladin Smartcard Environment)), 2 Kbyte (CC1 dari ASE(Alladin Smartcard Environment)), 22 Kbyte (JavaCard) dan 31 Kbyte(MSC0402 dari Motorola). Selain berisi informasi, *smartcard* juga berisi sistem operasi yang mengendalikan seluruh proses yang terjadi di *smartcard*.

3. Lebih sulit untuk ditiru daripada *kartu magnetik*

Kartu magnetik mempunyai pita magnetik pada permukaaannya. Peng-*copy*-an terhadap kartu magnetik dilakukan dengan meng-*copy* pita magnetik tersebut ke kartu lain. Pada *smartcard* peng-*copy*-an terhadap kartu sulit dilakukan, ini disebabkan karena setiap kartu memiliki nomor seri yang unik, tidak ada 2 buah kartu yang memiliki nomor seri yang sama. Jika pengaman dari kartu dilakukan dengan menghitung hash dari nomor seri kartu, maka peng-*copy*-an kartu tidak mungkin dilakukan. Selain itu juga disebabkan karena proteksi terhadap data dengan menggunakan *secret code*, sehingga data tidak dapat dibaca tanpa mengetahui *secret code*-nya.

4. Dapat digunakan kembali

Setelah nilai yang tertulis di dalam *smartcard*, misalnya jumlah pulsa/uang habis, *smartcard* dapat di'isi' ulang dengan menuliskan nilai tertentu ke dalamnya. Ini bisa dilakukan selama kondisi *smartcard* masih baik, misalnya tidak terdapat kerusakan pada chip. Berbeda dengan kartu magnetik, setelah nilai yang ada di dalamnya habis, maka kartu tersebut tidak dapat digunakan kembali.

5. Dapat melakukan banyak fungsi di berbagai area industri

Walapun kartu magnetik telah banyak dimanfaatkan di berbagai sektor, misalnya sektor perbankan dan sektor telekomunikasi, tetapi fungsi yang dapat dilakukan terbatas atau disebut *single function*. Misalnya sebagai kartu kredit untuk melakukan fungsi kredit. Karena keistimewaan yang dimiliki oleh *smartcard*, yaitu dalam hal kapasitas simpan dan kemampuan untuk melakukan proses, *smartcard* menawarkan skema *multi-function*, yaitu satu kartu untuk berbagai layanan. *Smartcard* banyak dimanfaatkan misalnya di sektor telekomunikasi, misalnya SIM card pada layanan GSM. SIM selain sebagai kartu telepon dengan sistem *Pre-paid* juga akan dikembangkan layanan untuk kredit, jadi semacam ATM pribadi. Di samping itu *smartcard* telah dimanfaatkan di sektor lain, seperti sektor keuangan, transportasi, dan kesehatan.

6. Selalu mengalami evolusi (sesuai dengan perkembangan chip komputer dan memori).

Smartcard mempunyai standar mikroprosesor 8-bit, namun saat ini mulai dikembangkan mikroprosesor 32-bit yang mempunyai keuntungan, yaitu memungkinkan melakukan pemrograman dengan menggunakan bahasa tingkat tinggi dan meningkatkan kekuatan komputasi untuk fungsi matematika yang kompleks yang tidak mungkin dilakukan pada mikroprosesor 8-bit [MRRK98]. Peningkatan kekuatan komputasi ini akan mempercepat jalannya program dan waktu transaksi. Dan yang paling penting, peningkatan MIPS (*million instruction per second*) memungkinkan industri *smartcard* memanfaatkan kemajuan teknologi biometri dan kriptografi.
Selain perkembangan mikroprosesor, perkembangan memori merupakan faktor penting dalam perkembangan *smartcard*. Misalnya proses pembuatan memori menggunakan 0.8 micron menghasilkan memori dengan ukuran 23K ROM, 8K EEPROM dan 384 byte RAM [MRRK98].

Dengan makin kecilnya satuan yang digunakan, misal 0.28 microm, makin kecil pula ukuran *die* (unit terkecil di dalam memori). Ini menyebabkan kapasitas memori di dalam chip tersebut menjadi semakin besar.

3.5 JAVA NATIVE INTERFACE (JNI)

JNI adalah *programming Interface* (antar muka pemrograman) antara *native method* dengan dengan *Java Virtual Machine*. *Native method* adalah *Java method* yang implementasinya ditulis di bahasa pemrograman lainnya, seperti C atau C++.

JNI tidak membatasi implementasi dari *Java Virtual Machine*. Pemrogram dapat menulis satu versi dari *native application* atau *library* yang dapat dijalankan di semua Java VM, walaupun implementasi internalnya berbeda.

3.5.1 Beberapa Alasan Penggunaan JNI

Pemrogram dapat menulis aplikasi seluruhnya dalam Java, namun ada beberapa situasi dimana Java tidak memenuhi kebutuhan aplikasi tersebut. Pemrogram menggunakan JNI untuk mengatasinya. Situasi tersebut misalnya :

- Java *class library* yang standar tidak mendukung *feature* yang *platform-dependent* yang diperlukan oleh aplikasi.
- Pemrogram mempunyai *library* yang ditulis dalam bahasa lainnya, dan ingin agar *library* tersebut dapat diakses oleh program Java melalui JNI.
- Pemrogram ingin mengimplentasikan bagian kecil kode yang *time-critical* dalam bahasa yang *low-level*, misalnya assembly.

3.5.2 Keuntungan dengan Adanya JNI

Interface tersebut menawarkan keuntungan sebagai berikut :

- masing-masing *vendor* VM dapat mendukung *native method*.
- *Tool builder* tidak perlu mempunyai *native method* yang berbeda-beda.
- pemrogram dapat menulis *native code* dan dapat menjalankannya di semua VM.

3.5.3 Kerugian Penggunaan JNI

Ada beberapa kerugian dari penggunaan JNI [SNMC97]:

- jika menggunakan *native method*, pemrogram harus mempunyai beberapa versi aplikasi

untuk berbagai target sistem yang akan digunakan oleh pemakai. Kode tidak portabel.

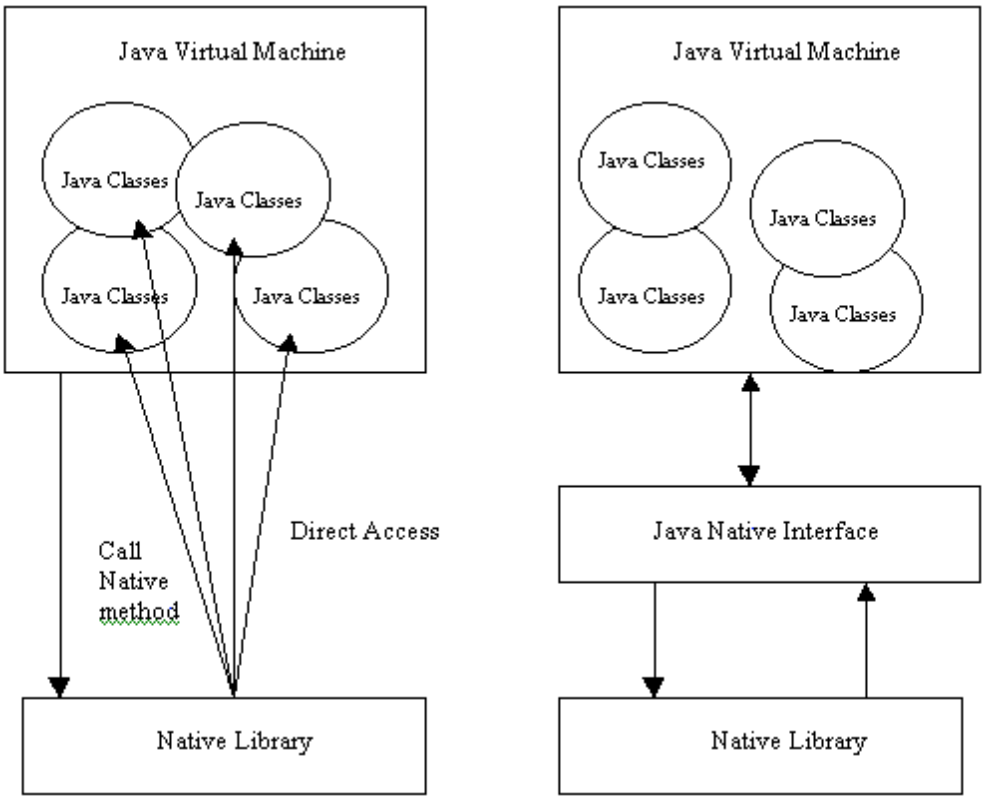
- *native method* memerlukan kemampuan pemrograman dan memerlukan banyak waktu untuk membuatnya sempurna.
- serangan terhadap program dapat menyebabkan resiko keamanan dengan mengganti kode *native*.
- mudah untuk menghancurkan sistem yang menggunakan *native method* yang mengandung *bug*.

3.5.4 Arsitektur

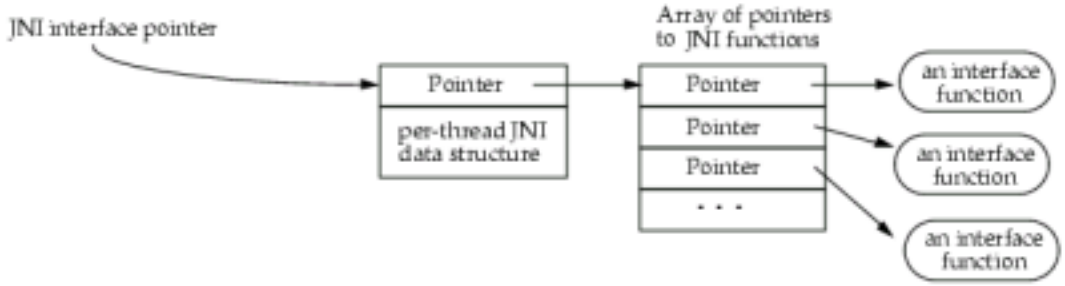
Arsitektur *runtime* pada Java 1.1 berbeda dengan Java 1.0. Perhatikan bagaimana Java 1.1 menggunakan lapisan *interface* untuk memisahkan VM dengan native *library* (lihat gambar 5.1). *Java Native Interface* diberikan ke *native library* sebagai C *pointer*. Merupakan *pointer to pointer* (lihat gambar 5.2), maksudnya *pointer* ini menunjuk ke *array of pointer*, yang masing-masing menunjuk ke *interface function*. *Interface* distrukturkan seperti C++ *virtual function table*[1].

Ini merupakan perubahan yang besar dari Java 1.0. Sebelumnya Java VM memanggil *native method* dan hanya memberikan *objek pointer*. Fungsi VM diakses secara langsung melalui pemanggilan *shared library*. Dengan JNI, Java VM memanggil *native method* dan memberikan JNI *pointer*. Parameter ini membatasi hubungan dengan *shared library*. Setiap fungsi yang diperlukan di-*embed* ke dalam JNI *pointer*.

Interface berisi serangkaian pemanggilan fungsi yang digunakan oleh *native method* untuk mengakses dan mempertukarkan data di dalam Java *runtime system*.



Gambar III.17. *Runtime system* pada JDK 1.0 dan JDK 1.1



Gambar III.18. *Interface pointer*

3.5.5 Perancangan dengan JNI

3.5.5.1 Loading and linking Native method

Native method di-*load* dengan *method* System.load*Library*. Dalam contoh berikut, bagaimana me-*load native library* :

```
package pkg;
class Cls {
        native double f(int I, String s);
        static {
                System.loadLibrary (“pkg_Cls”);
        }
}
```

Argumen dari load*Library* adalah nama dari *library*. Sistem akan secara otomatis menyesuaikan dengan *platform* dan mengubah nama *library* tersebut, misalnya dalam window akan ditambah *extension* .dll, sedangkan untuk unix akan ditambah *extension* .so.

3.5.5.2 Nama Native method

Nama dari *native method* merupakan gabungan dari komponen sebagai berikut:

- awalan Java_
- *fully qualified class name*
- pemisah garis bawah (“_”)
- nama *method*.
- untuk *native method* yang overload, 2 garis bawah(“__”) diikuti oleh *signature*

VM memeriksa nama *method* apakah cocok dengan nama *native library*. Pertama, VM akan melihat nama pendek (nama tanpa *signature*). Pemrogram menggunakan nama panjang hanya jika *native method overload* dengan *native method* lainnya. Namun tidak menjadi masalah jika *native method* mempunyai nama yang sama dengan non-*native method*, Non *native method* tidak berada di dalam *native library*.

3.5.5.3 Argumen Native method

JNI *interface pointer* adalah argumen pertama dari *native method*. JNI *interface pointer* mempunyai tipe JNIEnv. Argumen yang kedua berbeda tergantung apakah *native method* tersebut static atau bukan. Untuk non-static *method*, argumen kedua ini menunjuk ke objek, sedangkan untuk static *method* argumen kedua menunjuk ke Java *class*. Argumen selanjutnya sesuai dengan argumen dari *method* yang bersangkutan.

3.5.5.4 Referensi Objek Java

Tipe skalar seperti integer, character dan sebagainya disalin antara Java dan *native code*.

Sedangkan Objek Java di *pass by reference*. VM harus menjaga semua objek yang diberikan ke *native method*, sehingga objek tersebut tidak dibersihkan oleh *garbage collector*. Di samping itu *native code* harus mempunyai cara untuk memberi tahu VM bahwa *native code* sudah tidak memerlukan objek tersebut. Referensi ke objek tidak mencegah objek tersebut dibersihkan oleh *garbage collector*.

3.5.5.5 Referensi Global dan Lokal

JNI membagi referensi objek ke dalam 2 kategori : referensi lokal dan global. Referensi lokal hanya valid selama pemanggilan *native method*, dan secara otomatis dibebaskan setelah pemanggilan *method* tersebut kembali. Referensi global akan valid selama belum secara eksplisit dibebaskan.

Objek diberikan ke *native method* sebagai referensi lokal. Semua Java objek yang dikembalikan oleh fungsi JNI adalah referensi lokal. JNI memperbolehkan pemrogram membuat referensi global dari referensi lokal.

Dalam banyak kasus, pemrogram harus mempercayakan VM untuk membebaskan semua referensi lokal setelah pemanggilan terhadap *native method* kembali. Namun ada waktu-waktu di mana pemrogram harus secara eksplisit membebaskan referensi lokal, misalnya *native method* mengakses Java objek yang besar, dan membuat referensi lokal untuk objek Java tersebut. *Native method* kemudian melakukan penghitungan sebelum mengembalikannya ke pemanggil. Referensi lokal ke objek Java tersebut akan mencegah objek dibersihkan oleh *garbage collector* bahkan jika objek tersebut sudah tidak digunakan.

Untuk mengatasi hal tersebut, pemrogram secara manual menghapus referensi lokal di dalam *native method*.

3.5.5.6 Standar Native method

Standar *native method interface* harus memenuhi :

- *Binary compatibility*. Tujuan utamanya adalah *binary compatibility* dari *native method library* dengan semua implementasi dari Java VM pada berbagai *platform*. Pemrogram hanya perlu membuat satu versi dari *native method library*, dan selanjutnya *native code* tersebut harus dapat terus digunakan meskipun terdapat perubahan pada Java class.
- Efisiensi. Untuk mendukung *time-critical code*, *native method interface* harus mempunyai *overhead* yang kecil. Teknik yang banyak digunakan untuk memastikan VM-*independence* menggunakan *overhead* tertentu. Kadang pemrogram harus memilih antara efisiensi dengan VM-*independence.*

- fungsionalitas. *Interface* harus memberikan informasi representasi internal dari Java VM sehingga *native code* dapat melakukan fungsi yang diinginkan.

3.5.5.7 Penanganan Kesalahan

Berikut ini merupakan kemungkinan-kemungkinan terjadinya kesalahan dan penanganannya pada JNI :

- JNI memungkinkan *native method* untuk membangkitkan dan menangani Java *exception*. Fungsi JNI tertentu menggunakan mekanisme Java *exception* untuk menyatakan adanya kondisi *error*. Pada umumnya, mekanisme Java *exception* ini kurang cocok untuk melaporkan adanya *error* yang disebabkan oleh penggunaan yang salah dari fungsi JNI (misalnya memberikan argumen yang tidak valid).
- Tidak seperti di Java, dimana *error* secara otomatis diteruskan (*propagate*), *native code* dapat tidak mempedulikan *exception* yang timbul oleh pemanggilan JNI yang terakhir kali dan membuat pemanggilan JNI lainnya. Ini akan menghasilkan status sistem yang tidak valid.
- Memaksa fungsi JNI memeriksa semua kemungkinan *error* akan menurunkan unjuk kerja dari *native method*.
- Fungsi JNI tertentu, seperti *throw* (untuk *exception*) tidak dapat mempercayakan mekanisme *exception* itu sendiri untuk menyatakan kondisi *error*.
- Kebanyakan fungsi *library* C tidak menjaga terhadap *error* yang ditimbulkan oleh program. Misalnya fungsi printf, banyak menyebabkan *runtime error*, dari pada mengembalikan kode yang menyatakan *error* ketika merima kesalahan. Memaksa fungsi *library* C untuk memeriksa semua kemungkinan kondisi *error* tampaknya akan menghasilkan pemeriksaan ganda terhadap kode pemakai dan *library*.
- Fungsi JNI akan menggunakan kombinasi dari *error* kode dan Java *exception* untuk melaporkan kondisi *error*. Implementasi dari JNI akan memilih memeriksa *programming error* dari pada meminta dan mengembalikan dan membangkitkan Java *exception*.

3.6 STANDAR PENGKODEAN ASN.1 , BER , DAN DER

Secara umum abstraksi merupakan sebuah kunci dalam mengatur suatu pengembangan perangkat lunak. Dengan abstraksi, seorang desainer dapat melihat bagian dari sebuah sistem tanpa perlu tahu bagaimana bagian itu secara nyata akan diimplementasikan. Salah satu sistem kompleks yang menggunakan abstraksi adalah *Open Systems Interconnection* (OSI). OSI merupakan arsitektur standar yang mengatur hubungan komponen komputer dari lapisan fisik

sampai ke lapisan aplikasi. Sistem ini merupakan sebuah sistem terbuka (*open system*) yang mendukung penggunaan berbagai macam implementasi pada tiap-tiap lapisan.

3.6.1 Abstract Syntax Notation One (ASN.1)

Abstract Syntax Notation One (ASN.1) adalah sebuah notasi untuk menjelaskan abstraksi dari tipe data dan nilainya. ASN.1 ini merupakan suatu abstraksi yang lebih spesifik dalam OSI. Dalam ASN.1, sebuah tipe adalah kumpulan dari nilai-nilai. Untuk beberapa tipe tertentu ada yang nilainya terhingga (*finite*) dan ada juga tipe yang nilainya tak terhingga (*infinite*).

ASN.1 mempunyai empat jenis tipe yaitu :

- Tipe sederhana (*simple types*), merupakan sebuah tipe yang atomik dan tidak terdiri dari komponen-komponen lain. Tipe-tipe yang termasuk dalam jenis ini antara lain :
  - BIT STRING, yaitu string yang terdiri dari bit-bit (0 dan 1),
  - IA5String, yaitu string yang terdiri dari karakter IA5 (ASCII),
  - INTEGER, yaitu representasi tipe integer,
  - NULL, yaitu suatu nilai null,
  - OBJECT IDENTIFIER, yaitu sebuah barisan integer yang mengidentifikasikan suatu objek seperti algoritma atau tipe atribut,
  - OCTET STRING, yaitu suatu string oktet (8 bit string),
  - PrintableString, yaitu string yang terdiri dari karakter-karakter yang dapat dicetak,
  - UTCTime, yaitu nilai dari koordinat waktu universal atau Greenwich Mean Time (GMT).
- Tipe terstruktur (*structured types*), merupakan tipe-tipe yang terdiri dari tipe-tipe yang lain. Tipe-tipe yang termasuk dalam jenis ini antara lain :
  - SEQUENCE, yaitu suatu kumpulan terurut dari tipe-tipe yang lain,
  - SEQUENCE OF, yaitu kumpulan terurut dari jumlah kemunculan tipe-tipe tertentu,
  - SET, yaitu kumpulan tidak terurut dari tipe-tipe yang lain,
  - SET OF, yaitu kumpulan tidak terurut dari jumlah kemunculan tipe-tipe tertentu.
- Tipe tag (*tagged types*), merupakan tipe-tipe yang digunakan untuk membedakan antar tipe terstruktur. Tipe-tipe yang termasuk jenis ini antara lain :
  - [class number] IMPLICIT, yaitu tipe yang diperoleh dari tipe-tipe lain dengan mengganti tag dari tipe-tipe tersebut,
  - [class number] EXPLICIT, yaitu tipe yang diperoleh dari tipe-tipe lain dengan menambahkan tag di luar tipe-tipe tersebut.
- Tipe-tipe lain (*others types*), merupakan tipe-tipe yang digunakan untuk menjelaskan nilai-nilai apa saja. Tipe-tipe yang termasuk jenis ini antara lain :

- CHOICE, yaitu tipe yang menjelaskan gabungan dari satu atau lebih alternatif atau pilihan,
- ANY, yaitu tipe yang menjelaskan nilai dari tipe-tipe apa saja.

Pada ASN.1, tipe-tipe selain CHOICE dan ANY mempunyai sebuah tag yang berisi kelas dan nomor yang positif. Tipe-tipe dalam ASN.1 pada dasarnya sama jika nomor tag dari tipe-tipe tersebut sama. Dengan kata lain, nama dari tipe tidak mempengaruhi arti abstraksinya, yang mempengaruhi hanya nilai tag dari tipe-tipe tersebut. Ada empat kelas dari tag :

- *Universal*, untuk tipe-tipe yang artinya sama dalam semua aplikasi,
- *Application*, untuk tipe-tipe yang artinya spesifik pada suatu aplikasi,
- *Private*, untuk tipe-tipe yang artinya spesifik dari perusahaan tertentu,
- *Context-spesific*, untuk tipe-tipe yang artinya spesifik pada suatu tipe terstruktur tertentu, digunakan untuk membedakan tipe-tipe yang mempunyai tag sama dalam sebuah tipe terstruktur.

Tipe-tipe dalam ASN.1 sangat fleksibel, suatu notasi yang mirip bahasa pemrograman, dengan aturan-aturan khusus sebagai berikut :

- Tata letak (*layout*) bukan sesuatu yang penting, banyak spasi dan garis baru akan diartikan sebagai sebuah spasi,
- Komentar menggunakan pasangan tanda penghubung (--), atau sepasang tanda penghubung dan garis baru,
- Pengenal (*identifier*) yaitu nama dari nilai dan field dimulai dengan huruf kecil, sedangkan untuk tipe referensi (*type reference*) yaitu nama dari tipe dimulai dengan huruf besar.

3.6.2 BASIC ENCODING RULES (BER)

Basic Encoding Rules (BER) merupakan salah satu cara untuk merepresentasikan nilai dalam *Abstract Syntax Notation One* (ASN.1) menjadi sebuah string oktet. Sebenarnya ada banyak cara untuk merepresentasikan nilai ASN.1, tetapi BER merupakan standararisasi dalam OSI. ASN.1 disandikan dengan membentuk sebuah metode dalam BER. Dalam setiap metode penyandian dalam BER terdiri dari tiga sampai empat bagian, yaitu :

- *Identifier octets*. Bagian ini mengidentifikasikan kelas dan nomor tag dari nilai ASN.1, dan mengindikasikan apakah metode tersebut primitif atau *constructed*.
- *Length octets*. Bagian ini digunakan sebagai panjang dari data (*content octets*), untuk *constructed, indefinite-length method* panjangnya tidak terbatas.

- *Contents octets*. Bagian ini merepresentasikan nilai untuk *primitive, definitife-length method*. Sedangkan untuk *constructed method,* bagian ini merepresentasikan gabungan dari niai-nilai tersebut dalam BER.
- *End-of-contets octets*. Bagian ini digunakan untuk menandai akhir data pada *constructed, indefinite-length method*.

Ada tiga metode untuk menyandikan sebuah nilai ASN.1 dalam BER, yaitu :

- *Primitive, definite-length method*. Metode ini digunakan untuk tipe-tipe sederhana dan tipe-tipe yang diperoleh dari tipe-tipe sederhana dengan *tag* yang implisit. Bagian-bagian penyandian BER yang digunakan antara lain :
  - *Identifier octets*. Ada dua bentuk, yaitu nomor rendah (*low tag number*) untuk nilai antara 0 dan 30, dan nomor tinggi (*high tag number*) untuk nilai yang lebih besar dari 31.
  - *Low-tag-number form*. Satu oktet. Bit 8 dan 7 mengidentifikasikan kelas, bit 0 bernilai "0" berarti primitive, dan bit 5-1 adalah nomor *tag*.
  - *High-tag-number form*. Dua atau lebih oktet. Oktet pertama adalah *low-tag-number form*, kecuali bit 5-1 semuanya bernilai "1". Oktet berikutnya berisi nomor tag berbasis 128, angka penting digunakan pertama, dengan jumlah bit yang minimal, dan dengan bit 8 dari masing-masing oktet diset ke "1" kecuali oktet terakhir.
  - *Length octets*. Ada dua bentuk, yaitu : *short* (untuk panjang antara 0 dan 127), dan *long definited* (untuk panjang antara 0 dan 2^{1008}-1).
  - *Short form.* Satu oktet. Bit 8 bernilai "0" dan bit 7-1 adalah panjangnya.
  - *Long form.* Dua sampai 127 oktet. Bit 8 dari oktet pertama mempunyai nilai "1" dan bit 7-1 memberikan nilai dari panjang oktet-oktet tersebut. Oktet berikutnya berisi panjang berbasis 256, dengan mendahulukan angka penting.
  - *Contents octets*. Metode ini memberikan representasi yang nyata untuk nilai suatu tipe, atau nilai dari tipe yang diperoleh dari *implisit tagging*.
- *Constructed, definite-length method*. Metode ini digunakan untuk tipe-tipe *string* sederhana, tipe-tipe terstruktur, tipe-tipe yang diperoleh dari tipe *string* sederhana dan tipe terstruktur dengan *tag* secara implisit, dan tipe-tipe yang diperoleh dari apapun dengan *tag* secara eksplisit.
  - *Identifier object*, seperti yang dijelaskan sebelumnya, kecuali bit 6 berisi nilai "1".
  - *Length octets*, sama seperti yang sebelumnya.
  - *Contents octets*, merupakan gabungan dari penyandian nilai-nilai dalam BER.
- *Constructed, indefinite-length method*. Metode ini digunakan untuk tipe-tipe *string*

sederhana, tipe-tipe terstruktur, tipe yang diperoleh dari *string* sederhana dan tipe terstruktur dengan *implicit tagging*, dan tipe-tipe yang diperoleh dari apa saja dengan *tag* secara eksplisit. Bagian-bagian dari BER pada metode ini antara lain :

- *Identifier octets*, sama seperti nomor 2.
- *Length octets*. Satu oktet, 80.
- *Contents octets*, sama seperti nomor 2.
- *End-of-contents octets*. Dua oktet, 00 00.

3.6.3 Distinguished Encoding Rules (DER)

Distinguished Encoding Rules (DER) merupakan subset dari *Basic Encoding Rules* (BER), dan memberikan cara yang tepat untuk merepresentasikan nilai dalam ASN.1 ke suatu string oktet (string 8-bit). DER ditujukan untuk aplikasi-aplikasi yang membutuhkan penyandian string oktet yang unik, sebagaimana yang digunakan pada tanda tangan digital pada ASN.1.

DER mempunyai aturan-aturan seperti yang dimiliki oleh BER ditambah lagi dengan aturan-aturan berikut :

- Jika panjang data diantara 0-127 maka digunakan bentuk pendek (*short form*).
- Jika panjangnya 128 atau lebih maka digunakan bentuk panjang (*long form*), selain itu panjang data harus disandikan dalam jumlah oktet yang minimal.
- Untuk tipe-tipe string dan tipe-tipe *tag* yang implisit yang diperoleh dari tipe string sederhana dan primitif maka digunakan *definite-length method*.
- Untuk tipe-tipe terstruktur, tipe-tipe yang secara implisit diperoleh dari tipe-tipe terstruktur, dan tipe-tipe *tag* yang secara eksplisit diperoleh dari apa saja (*anything*) dan *constructed* maka digunakan *definite-length method*.

BAB IV
TEKNOLOGI WALLET

Bab ini membahas tentang teknologi *wallet* serta penggunaannya dalam Sistem Perdagangan di Internet (SPI).

4.1 PENDAHULUAN

Transaksi perdagangan yang dilakukan melalui Internet biasanya menggunakan *browser* untuk antar mukanya [ARCL98]. Protokol SSL sudah menjadi standar untuk komunikasi yang aman pada setiap *browser* saat ini. Protokol ini sudah terintegrasi dengan *browser-browser* utama seperti Microsoft Internet Explorer dan Netscape Navigator. Namun transaksi di Internet membutuhkan data-data tertentu yang perlu di-*customize* oleh pemakai. Data-data tersebut perlu disesuaikan dengan kebutuhan transaksi dan spesifikasi protokol pembayaran sehingga harus ada mekanisme tertentu untuk melakukan *customize* data sebelum dikirim ke alamat tujuan. Solusinya adalah dengan menggunakan sebuah aplikasi yang dapat dijalankan dari *browser* untuk melakukan transaksi, selanjutnya aplikasi ini disebut sebagai *wallet*.

Protokol SET sebagai protokol yang dirancang khusus untuk transaksi di Internet, mau tidak mau juga harus menggunakan aplikasi *wallet*. *Browser* akan menjalankan aplikasi *wallet* kemudian mengirimkan data-data transaksi yang dibutuhkan ke *wallet* tersebut. Konsumen dapat melakukan *customize* data-data dalam *wallet* dan memberi persetujuan pada transaksi tersebut. Jika transaksi dilakukan, *wallet* akan mengirimkan data-data transaksi ke alamat pedagang sesuai dengan spesifikasi protokol SET.

Aplikasi *wallet* telah banyak dikembangkan oleh vendor-vendor terkemuka di dunia, misalnya Microsoft Wallet oleh Microsoft, vWallet oleh Verisign, MasterCard Wallet oleh MasterCard, dan Java Wallet oleh Sun Microsystem. Sebenarnya *wallet-wallet* tersebut mempunyai kesamaan dari sisi teknologi yang digunakan. Teknologi yang banyak digunakan pada *wallet-wallet* tersebut mengikuti standar dari *browser-browser* yang paling banyak digunakan saat ini, yaitu Internet Explorer dan Netscape Navigator. Internet Explorer menggunakan teknologi ActiveX [ETMG96] sedangkan Netscape Navigator menggunakan teknologi Java Applet pada arsitektur Netscape ONE (*Open Networking Environment*) [MIMO97]. Keduanya berusaha agar *browser* miliknya dapat menjalankan teknologi *browser*

yang lain (*interoperability*). Microsoft menggunakan ActiveX Control untuk menjalankan Java Applet pada Internet Explorer, sedangkan Netscape menggunakan Netscape Plug-ins untuk menjalankan ActiveX pada Netscape Navigator.

4.2 TEKNOLOGI ACTIVEX

4.2.1 Deskripsi

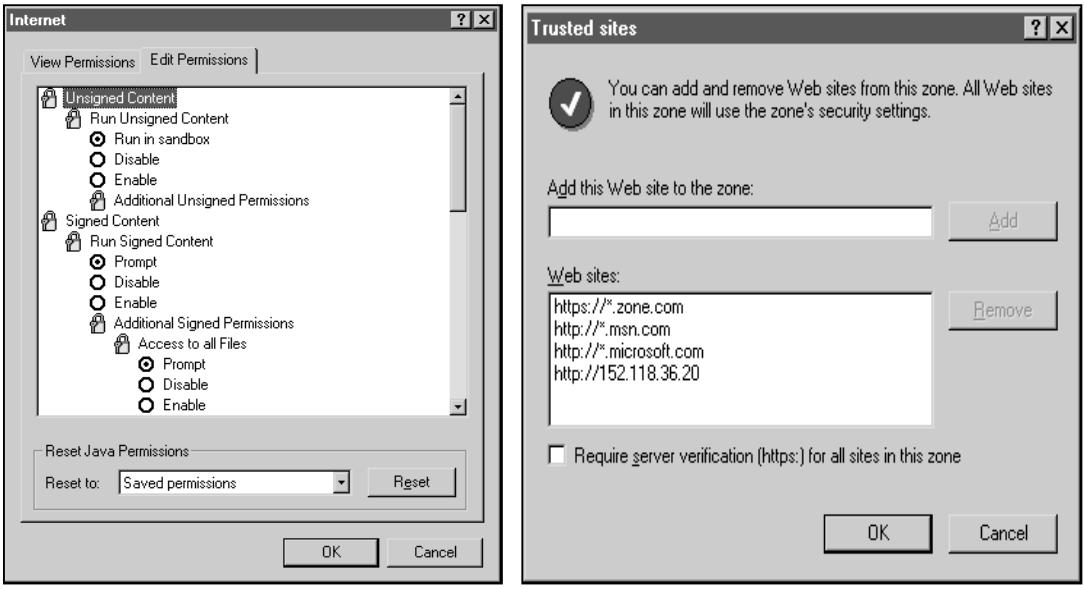
ActiveX adalah teknologi yang digunakan untuk membuat *web* interaktif yang merupakan pengembangan dari teknologi OLE (*Object Link & Embedding*) yang diterapkan pada dokumen HTML (*Hypertext Markup Language*) [ETMG96]. Teknologi ini diperkenalkan oleh Microsoft pada bulan Maret 1996, dalam rangka *Professional Developer's Conference* (PDC) di San Francisco, Amerika Serikat. Empat komponen utama pada ActiveX antara lain :

- ActiveX Control, yaitu program yang dapat digabungkan (*embed*) dengan halaman web sebagai *container*-nya.
- ActiveX Scripting / VBScript, yaitu bahasa pemrograman tingkat tinggi pada *browser*.
- ActiveX Document, yaitu sebuah dokumen yang merepresentasikan format-format data tertentu seperti dokumen pada Microsoft Word dan *spreadsheet* pada Microsoft Excel.
- ActiveX Server Scripting (ISAPI), yaitu bahasa *script* pada *server* sebagai tandingan untuk CGI script.

4.2.2 Keamanan

ActiveX menggunakan teknologi tanda tangan digital (*digital signature*) untuk menjamin keamanan komputer dari kode-kode yang berbahaya. Berdasarkan ada tidaknya tanda tangan digital maka ActiveX dapat dikategorikan menjadi Signed ActiveX Control dan Unsigned ActiveX Control. Signed ActiveX Control adalah ActiveX Control yang telah ditanda tangani oleh pembuat atau pemiliknya, sedangkan Unsigned ActiveX Control adalah ActiveX Control yang tidak ditanda tangani. Pemakai dapat menentukan tingkat keamanan *browser* dengan hanya menerima Signed ActiveX Control saja atau juga menerima Unsigned ActiveX Control. ActiveX Control yang digunakan oleh seorang pemakai dapat melakukan proses seperti layaknya program biasa. Kemampuan ActiveX untuk melakukan proses di komputer pemakai menyebabkan pemakai harus tetap waspada terhadap segala kemungkinan yang dilakukan ActiveX tersebut. Sebuah ActiveX Control yang dibuat oleh orang-orang yang tidak bertanggung jawab dapat berisi program yang merusak sistem komputer. Berdasarkan pertimbangan-pertimbangan ini, seorang

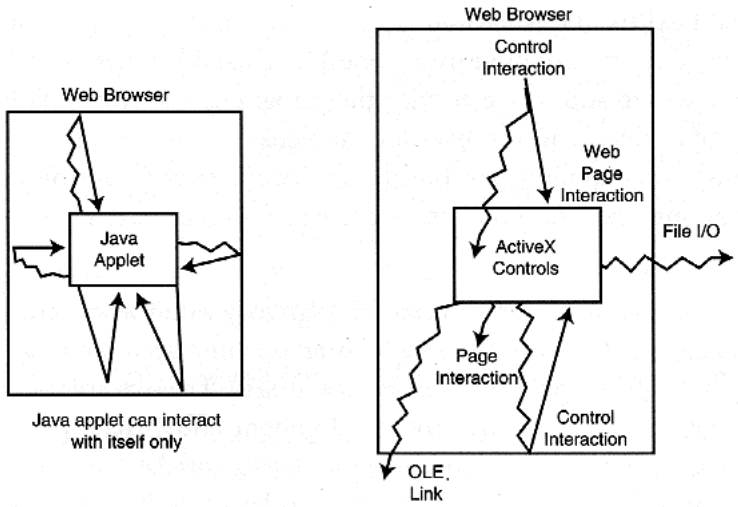
pemakai sebaiknya tidak menerima Unsigned ActiveX Control dan Signed ActiveX Control yang ditanda tangani oleh pihak-pihak yang tidak dikenal. Gambar IV.1 memperlihatkan sistem keamanan ActiveX pada Internet Explorer yang memperbolehkan pemakai untuk mengatur tindakan *browser* terhadap ActiveX tertentu.



Gambar IV.1. Sistem keamanan ActiveX pada Internet Explorer

4.2.3 Aplikasi

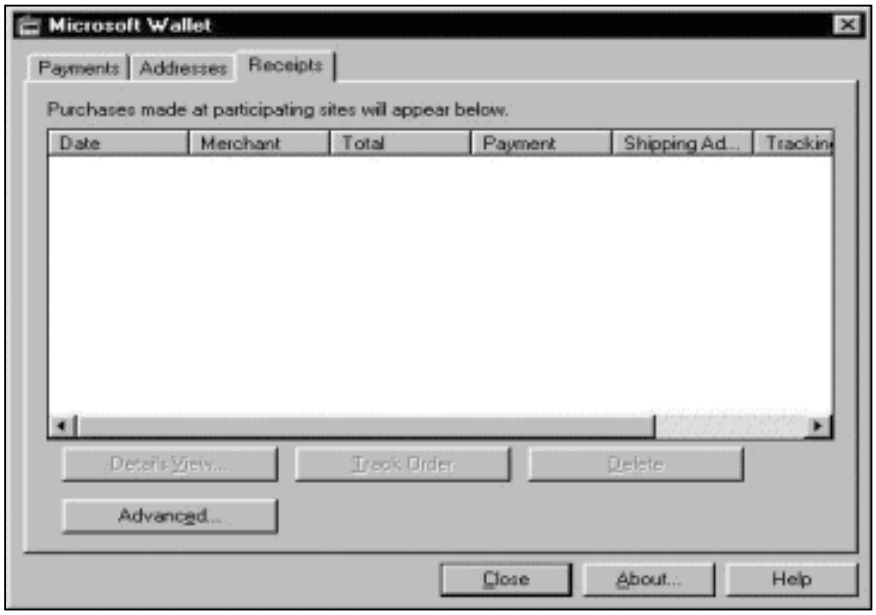
Sebagian besar aplikasi *wallet* yang telah dikembangkan menggunakan teknologi ActiveX. Alasannya adalah saat diperkenalkan pertama kalinya, teknologi ini merupakan satu-satunya teknologi yang mendukung halaman web interaktif yang dapat mengakses sumber daya lokal dengan sistem keamanan yang baik. Teknologi Java Applet yang baru dikeluarkan beberapa bulan sebelumnya tidak dapat digunakan untuk mengakses sumber daya (*resources*) komputer lokal. Pada saat itu Java versi 1.0 yang baru dikeluarkan oleh Sun Microsystem masih menggunakan teknologi *applet* yang *restricted*, yaitu *applet* tidak dapat melakukan proses-proses yang berhubungan dengan sumber daya komputer lokal, seperti mengakses hardisk, membuka hubungan port, dan mengakses sistem. Gambar IV.2 memperlihatkan perbedaan antara Java Applet 1.0 dan ActiveX.



Gambar IV.2. Java Applet 1.0 vs ActiveX

Wallet-wallet berbasis ActiveX yang telah dikeluarkan saat ini sudah cukup banyak dan yang terkenal adalah Microsoft Wallet oleh Microsoft, vWallet oleh Verisign, DigiCash Wallet oleh Digicash, eWallet oleh Launchpad Tech, BlueMoney Wallet oleh BlueMoney Soft, GlobeSet Wallet oleh GlobeSet Inc, dan Qwallet oleh Qwallet Inc.

Microsoft Wallet adalah produk *wallet* dari Microsoft yang bertujuan untuk memberi keamanan dan kenyamanan pada konsumen saat melakukan transaksi di Internet [WERN96]. Teknologi yang digunakan saat ini adalah SSL, tetapi Microsoft juga sudah mengeluarkan *wallet* yang mendukung spesifikasi SET yaitu Microsoft Wallet 3.0 (gambar IV.3).

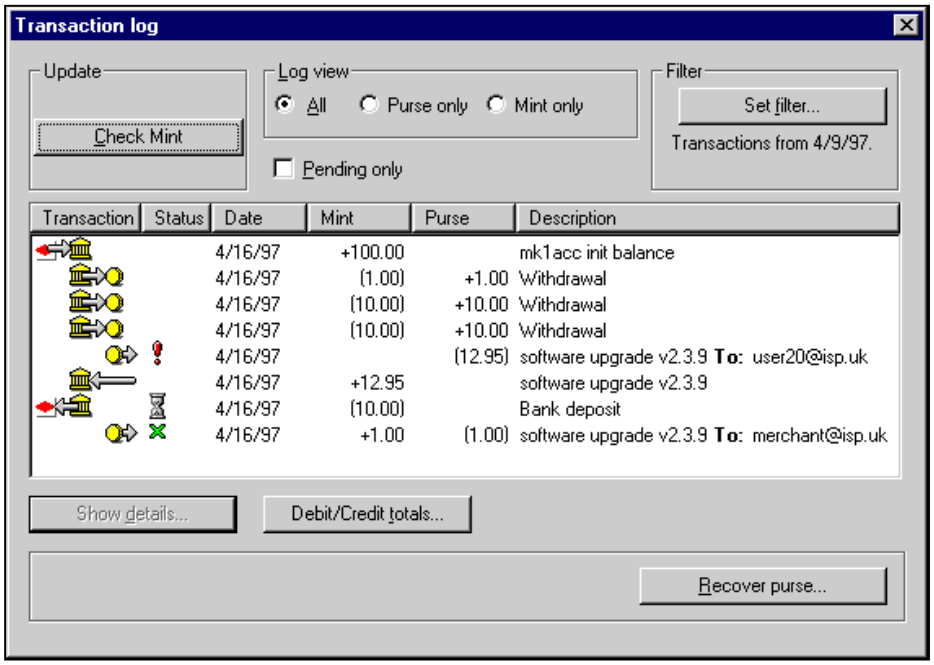


Gambar IV.3. Microsoft Wallet

Transaksi menggunakan Microsoft Wallet sama dengan *wallet-wallet* yang lain, yaitu konsumen menyimpan informasi kartu kredit miliknya kemudian memproteksinya dengan *password* di dalam *wallet* kemudian informasi tersebut dapat digunakan pada transaksi-transaksi berikutnya. Microsoft melakukan distribusi *wallet* melalui beberapa perangkat lunak yang dikeluarkannya, antara lain Microsoft Internet Explorer, Microsoft Windows, Microsoft Site Server, Microsoft Commerce, dan juga melalui Internet. Berikut ini adalah teknologi ActiveX Control yang diterapkan pada Microsoft Wallet :

- *Address Control*, digunakan untuk menyimpan alamat dari pihak-pihak lain dalam transaksi.
- *Payment Control*, digunakan untuk menyimpan informasi perdagangan, seperti kartu kredit, uang digital, dan cek elektronis.

Digicash Wallet adalah *wallet* dengan teknologi ActiveX yang pertama kali menggunakan uang digital (*electronic money*) untuk melakukan transaksi di Internet. Konsumen dapat membeli uang digital dari otoritas yang mengeluarkannya, uang ini disimpan di dalam *hardisk* konsumen. Setiap terjadi transaksi, konsumen dapat mengatur jumlah uang yang akan dikeluarkan tergantung biaya barang yang dibelinya. Gambar di bawah ini memperlihatkan *log* transaksi menggunakan uang digital pada Digicash Wallet.



Gambar IV.4. Digicash Wallet

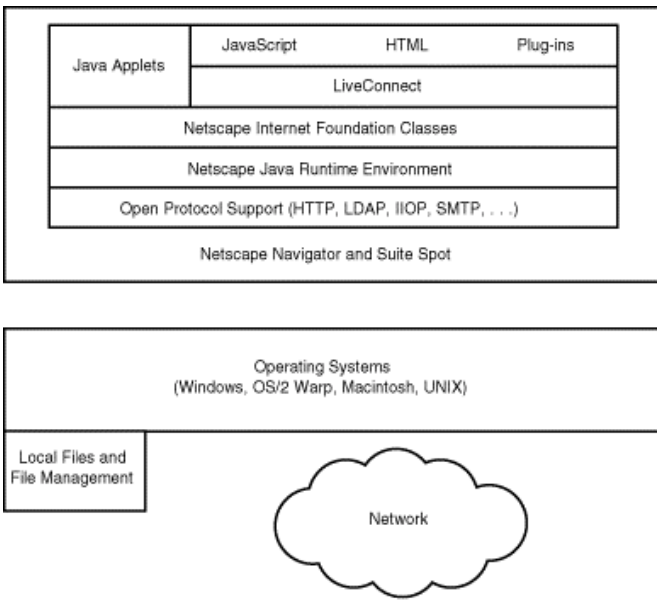
4.2.4 Prospek

ActiveX merupakan sebuah teknologi yang tidak tergantung pada bahasa pemrograman (*language idependent*) sehingga dapat diimplementasi dengan Visual C++, Visual Basic, Borland C++, Borland Delphi, dan bahkan dapat menggunakan bahasa Java. Java Applet dapat diintegrasi dengan ActiveX Control menggunakan ActiveX Control yang bertindak sebagai *Java Virtual Machine* (JVM). Java Virtual Machine pada dasarnya adalah sebuah *interpreter* yang berfungsi untuk menjalankan program Java. Integrasi antar Java Applet dengan ActiveX control merupakan prospek di masa depan mengingat dominansi Java sebagai bahasa pemrograman di Internet dan ActiveX sebagai standar untuk aplikasi terdistribusi yang interaktif [WERN96].

4.3 NETSCAPE PLUG-INS

4.3.1 Deskripsi

Netscape Plug-Ins merupakan bagian dari Netscape *Open Networking Environment* (ONE) yang berfungsi untuk merepresentasikan berbagai tipe data yang akan ditampilkan dalam halaman *web* [MIMO97]. Netscape ONE adalah kumpulan peralatan (*tools*) dan teknologi yang dapat digunakan untuk mengembangkan aplikasi-aplikasi di Internet dan Intranet. Berikut ini diberikan gambaran mengenai arsitektur Netscape ONE.



Gambar IV.5. Arsitektur Netscape ONE

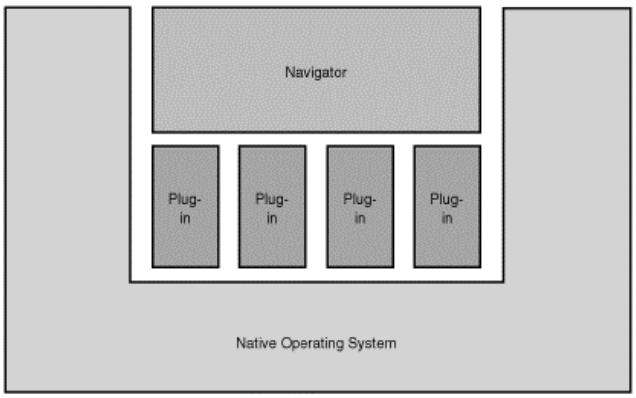
Netscape telah melengkapi kemampuan Netscape Navigator dengan *plug-ins* sejak versi 3.0, bersamaan dengan itu juga mengeluarkan teknologi Netscape yang disebut LiveConnect yang berfungsi untuk mengintegrasikan *plug-ins*, JavaScript dan Java Applet. JavaScript adalah bahasa *object oriented* tingkat tinggi yang digunakan pada pemrograman *browser*. Selain itu, JavaScript juga dapat digunakan untuk melakukan pemrograman pada CGI (*Common Gateway Interface*), teknologi ini disebut LiveWire (JavaScript *Server*). Java Applet adalah bahasa *object oriented* yang dapat digabungkan dengan halaman *web* untuk membuat sebuah *web* yang interaktif. Plug-ins digunakan untuk membaca dan merepresentasikan tipe data tertentu agar dapat dibaca oleh *browser* Netscape Navigator. Tidak seperti Java Applet yang dapat digunakan pada setiap *platform* yang ada, *plug-ins* dibuat secara spesifik untuk jenis *platform* tertentu saja. GUI (*Graphical User Interface*) untuk *plug-ins* di tiap *platform* juga berbeda, tergantung jenis *platform* itu sendiri. Aplikasi windows akan menggunakan Windows API, aplikasi Macintosh menggunakan Macintosh Toolbox, sedangkan aplikasi UNIX dapat menggunakan salah satu "*widgets*" yang ada dilingkungan sistem X Windows.

4.3.2 Keamanan

Netscape Plug-Ins menggunakan tanda tangan digital (*digital signature*) seperti yang digunakan pada ActiveX untuk melakukan autentikasi. Perusahaan Netscape telah mengeluarkan aplikasi yang digunakan untuk membuat tanda tangan digital yang disebut sebagai Netscape Signing Tools. Aplikasi ini dapat digunakan untuk menandatangani berbagai jenis file yang akan diautentikasi melalui *browser*, seperti file HTML, applet, audio, video, plug-ins, dan bahkan ActiveX Control.

4.3.3 Aplikasi

Aplikasi *wallet* yang menggunakan teknologi ActiveX hanya dapat digunakan pada *browser* Microsoft Internet Explorer khususnya yang mendukung Windows 32-bit. Hal ini disebabkan teknologi ini menggunakan MFC (*Microsoft Foundation Classes*) *library* yang bertujuan untuk memperkecil ukuran ActiveX Control. Akibatnya, Microsoft Internet Explorer hanya dapat digunakan di lingkungan win32 pada generasi Microsoft Windows, padahal ada banyak sistem operasi yang ada saat ini seperti OS2, Unix, Mac, dan IBM AS400.



Gambar IV.6. Arsitektur Netscape Plug-Ins

Perusahaan Ncompass Labs telah membuat *plug-ins* untuk teknologi ActiveX pada *browser* Netscape. Dengan adanya *plug-ins* ini maka ActiveX Control yang dibuat khusus untuk *browser* Microsoft Internet Explorer dapat juga digunakan pada Netscape Navigator. *Plug-Ins* ini mengakibatkan sistem operasi-sistem operasi yang sebelumnya tidak dapat menjalankan ActiveX kini dapat menjalankannya, tentunya dengan menggunakan *browser* Netscape Navigator. Demikian juga halnya dengan *wallet* yang menggunakan teknologi ActiveX dapat dijalankan dengan baik menggunakan *browser* Netscape Navigator.

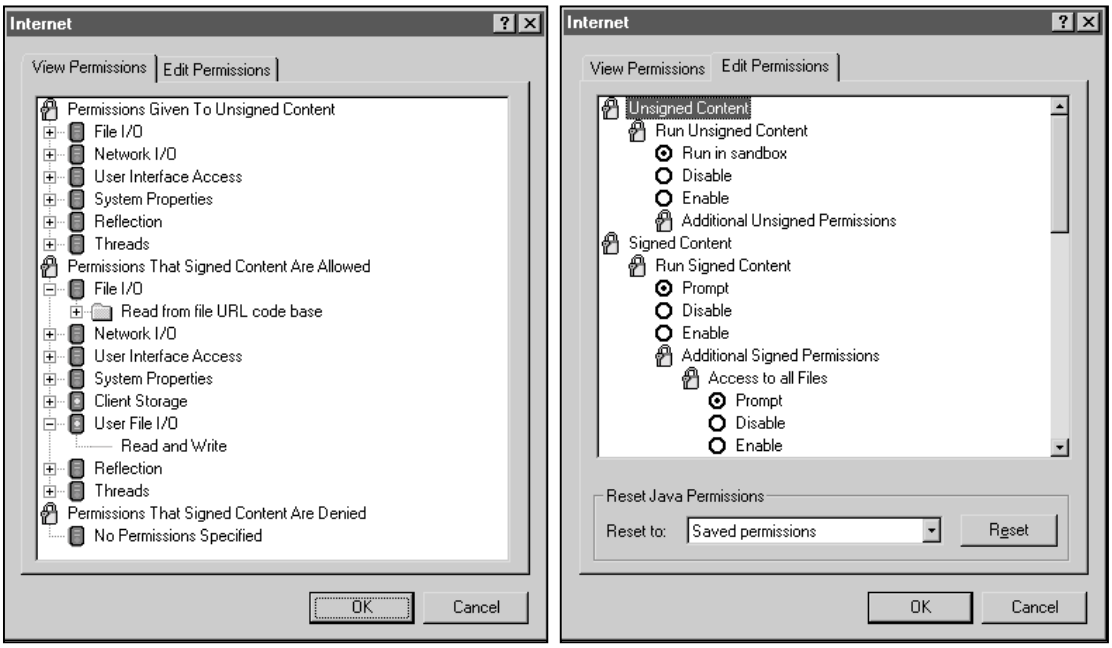
4.3.4 Prospek

Netscape Plug-Ins merupakan teknologi yang digunakan untuk merepresentasikan berbagai tipe data yang akan ditampilkan di halaman *web*. Saat ini telah banyak *plug-ins* yang dibuat untuk Netscape Navigator, salah satunya adalah ActiveX *plug-ins* yang digunakan untuk merepresentasikan teknologi ActiveX dengan *browser* Netscape Navigator. Disamping itu, Netscape dengan teknologi Netscape ONE telah berhasil mengintegrasikan halaman web. Teknologi-teknologi Netscape ONE yang mendukungnya adalah Live Wire (integrasi basis data dengan *web*), Live Connect (integrasi *plug-ins*, Java, dan Java Script), dan Live Payment (integrasi pembayaran antara *client* dengan *server*).

4.4 TEKNOLOGI JAVA APPLET

4.4.1 Deskripsi

Java adalah bahasa pemrograman *object oriented* yang dikeluarkan oleh Sun Microsystem. Salah satu teknologinya adalah Java Applet yang digunakan untuk membuat sebuah *web* yang interaktif. Java Applet dapat digunakan pada *browser* yang *Java-enabled*, teknologi ini sudah dimiliki sejak Netscape Navigator 2.0 dan Internet Explorer 3.0. *JDK (Java Development Kit*) juga menyediakan program *applet viewer* yang digunakan untuk menguji *applet* sebelum diintegrasikan dengan halaman *web*.

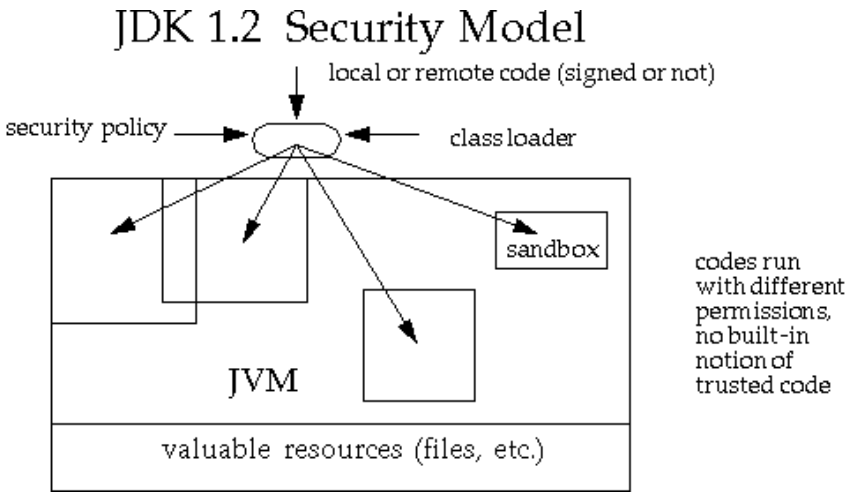


Gambar IV.7. Sistem keamanan Java Applet pada Internet Explorer

4.4.2 Keamanan

Java Applet telah mengalami perubahan yang cukup banyak sejak versi pertamanya. Java Applet versi 1.0 menggunakan sistem keamanan yang *applet restricted*, yaitu *applet* tidak dapat mengakses sumber daya lokal untuk menjamin keamanan di komputer pengguna. Java Applet versi 1.1 dan 1.2 sudah diperbolehkan untuk mengakses sumber daya lokal seperti *hardisk*, hubungan *port*, dan sistem komputer. Pada versi ini, Java Applet menggunakan *priviledge service* untuk mengatur prioritas tiap-tiap *applet* yang dijalankan. Jadi, sebuah *applet* dapat mempunyai hak-hak tertentu yang digunakan untuk mengakses sumber daya lokal. Hak-hak tesebut misalnya

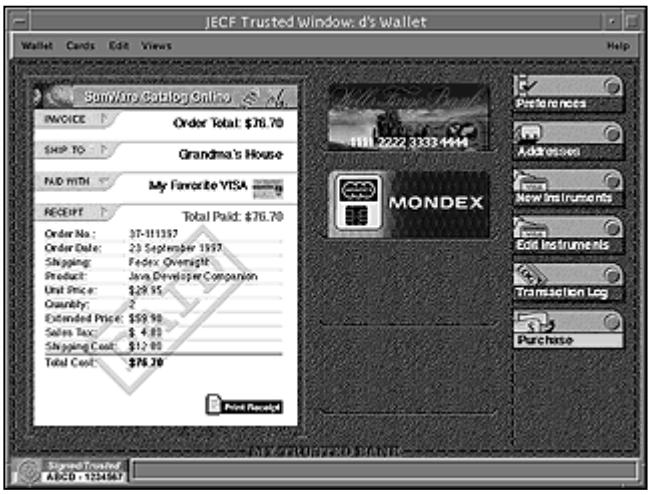
membaca *hardisk*, menulis *hardisk*, membuka hubungan *port*, dan mengakses sistem. Pemakai dapat mengijinkan sebuah *applet* untuk mempunyai akses pada sistem komputer dan dapat melarang *applet* tertentu untuk mengaksesnya (*sandbox model*). Gambar berikut ini memperlihatkan model keamanan pada Java Applet 1.2.



Gambar IV.8. Model Keamanan Java Applet 1.2

4.4.3 Aplikasi

Aplikasi *wallet* yang telah menggunakan teknologi Java Applet saat ini adalah Java Wallet yang dibuat Sun Microsystem. Java Wallet menyediakan antar muka yang digunakan untuk melakukan transaksi elektronis. Java Wallet itu sendiri merupakan sebuah aplikasi yang harus diinstalasi di komputer konsumen sebelum dapat digunakan, sedangkan pedagang menjual barang-barangnya di *web* melalui applet-applet yang berfungsi sebagai *basket*. Applet-applet ini terhubung dengan *cassettes* yang menyediakan *service* tambahan seperti protokol pembayaran, *security*, informasi konsumen, dan basis data. Pedagang mengimplementasikan *cassettes* dan applet-applet ini sesuai dengan kebutuhannya. Java Wallet didesain untuk diletakkan di komputer lokal dan hanya memperbolehkan sebuah aplikasi Java Wallet saja di tiap komputer. Sistem keamanan pada Java Wallet mengikuti sistem keamanan pada Java Applet yang digunakannya. Gambar IV.9 memperlihatkan transaksi yang menggunakan Java Wallet v1.1.3.



Gambar IV.9. Java Wallet dari Sun MicroSystem

4.4.3.1 Setting Java Wallet

Setelah Java Wallet di *download* dan di *install* maka sebelum digunakan harus dilakukan *setting* lebih dulu, antara lain:

- memasukkan identitas pribadi (*user profile*), seperti nama, alamat, dan nomor telepon,
- memasukkan pengenal untuk Java Wallet, contohnya "Iman's Java Wallet",
- menambahkan metode pembayaran yang dibutuhkan, seperti protokol SET,
- mengatur sistem keamanan pada Java Wallet,
- mengatur *preferences* pada Java Wallet, seperti warna dan tekstur,
- mengatur sertifikat yang akan digunakan.

4.4.3.2 Pembayaran dengan Java Wallet

Java Wallet memperbolehkan konsumen untuk memilih jenis pembayaran yang diperbolehkan pedagang, seperti kartu kredit, kartu debit, uang digital, atau kupon digital. Pembayaran tersebut dapat dilakukan secara :

- Pembayaran langsung ke pedagang, yaitu setelah konsumen memilih barang-barang yang akan dibelinya, maka ia dapat menekan tombol "PAY".
- Pre-*authorized payment*, yaitu pembayaran konsumen yang bertambah sesuai dengan lamanya menggunakan pelayanan yang disediakan pedagang, contohnya permainan di Internet.
- Transfer nilai uang secara *online*, hal ini dilakukan jika seorang pemakai akan melakukan transfer uang ke pemakai yang lain.

4.4.3.3 Administrasi Java Wallet

Seorang pemakai dapat melakukan fungsi-fungsi administrasi pada aplikasi Java Wallet, baik saat melakukan transaksi maupun saat tidak melakukan transaksi. Fungsi-fungsi yang terdapat pada administrasi Java Wallet, antara lain:

- *Upgrade* Java Wallet, yaitu melakukan instalasi versi baru dari Java Wallet, grafik, *cassettes*, dan basis data,
- Tambah/Ubah identitas di Java Wallet, yaitu pemakai dapat menambah dan mengubah identitas di Java Wallet,
- Administrasi instrumen, yaitu pemakai dapat menambah atau menghapus intrumen-instrumen yang digunakan di Java Wallet, misalnya protokol pembayaran,
- Alamat, yaitu pemakai dapat mengubah alamat yang digunakan untuk menerima barang-barang yang telah dibeli,
- Log transaksi, yaitu pemakai dapat melihat, mengurutkan dan menghapus keterangan transaksi-transaksi yang pernah dilakukan.
- *Preferences*, yaitu pemakai dapat mengatur pilihan pada Java Wallet, misalnya tingkat keamanan, batasan nilai uang yang dikirim, dan identitas diri.

4.4.3.4 Interaksi Java Wallet dengan Pedagang

Interaksi antara Java Wallet dengan pedagang tidak *linear*, melainkan bersifat komunikasi dua arah. Sebagian besar komunikasi ini terjadi secara implisit karena pemakai tidak perlu mengetahuinya. Komunikasi-komunikasi ini antara lain terjadi pada proses protokol pembayaran, koneksi "*backend*", sistem keamanan dan protokol autentikasi. Ketika konsumen mengunjungi alamat pedagang, pedagang dapat memperoleh identitas konsumen di Java Wallet, seperti nama, alamat, dan sebagainya. Konsumen dapat mengatur informasi mana yang boleh dilihat oleh pedagang, tergantung *trusted* yang diberikan kepada pedagang tersebut.

4.4.4 Prospek

Java Wallet yang relatif baru dibandingkan wallet-wallet lain yang menggunakan teknologi ActiveX mempunyai prospek yang bagus untuk investasi karena murahnya biaya yang dikeluarkan untuk pengembangan, instalasi, dan perawatan [ARCL98]. Prospek-prospek yang dimiliki Java Wallet antara lain untuk digunakan pada *online banking*, investasi (*investment*), penagihan (*billing*), pertukaran (*bartering*), dan negosiasi kontrak (*contract negotiation*).

ACTIVEX	**JAVA APPLET**
Menggunakan *library* MFC (*Microsoft Foundation Class*).	Menggunakan *library* JRE (*Java Runtime Environment*).
ActiveX dibuat khusus pada lingkungan win32.	Java Applet dapat dijalankan pada berbagai *platform* dan *browser* yang Java *enabled*.
Autentikasi menggunakan tanda tangan digital.	Autentikasi menggunakan tanda tangan digital dan *policy file*.
Signed ActiveX yang *trusted* mempunyai kekuasaan yang penuh terhadap sistem komputer.	Setiap *applet* mempunyai *permission* yang menentukan hak-hak yang dimilikinya.
Browser tidak dapat mencatat proses-proses yang dilakukan ActiveX.	Java *enabled browser* dapat mencatat proses-proses yang dilakukan oleh sebuah applet.
Kerusakan data dapat terjadi jika menjalankan sebuah ActiveX yang mempunyai kesalahan (*bug*).	Java Applet mempunyai penanganan kesalahan yang lebih baik sehingga mengurangi resiko terjadinya kerusakan data jika menjalankan *applet* yang mempunyai kesalahan (*bug*).

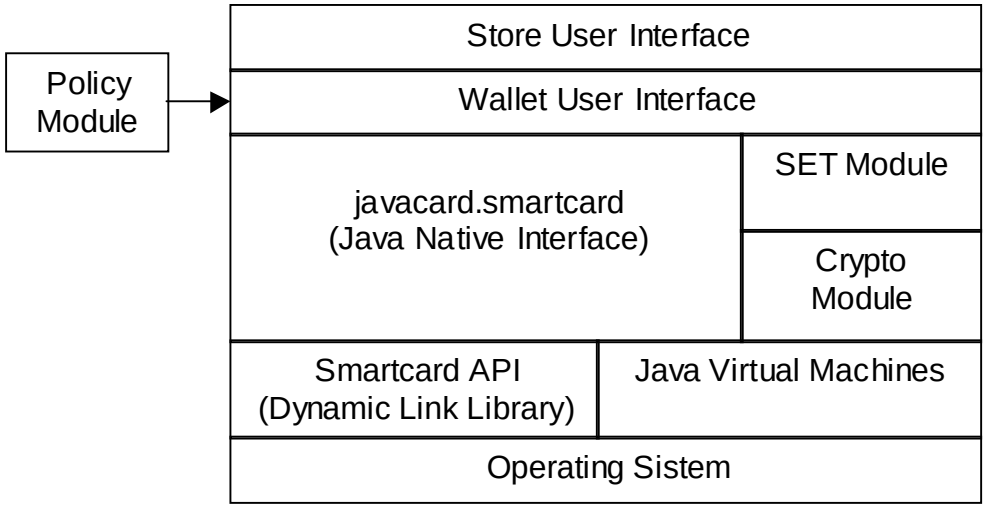
Tabel IV-1 Perbandingan ActiveX dan Java Applet

BAB V
PERANCANGAN SISTEM

Bab ini membahas mengenai perancangan SmartWallet yang terdiri dari arsitektur SmartWallet, analisa dan perancangan proses, perancangan *file* / basis data, dan perancangan antar muka (*user interface)* dari aplikasi tersebut. Penelitian tugas akhir ini menggunakan bahasa pemrograman Java yang berorientasi *object* karena modul-modulnya yang lebih terstruktur dan *reusable* (modul-modulnya dapat digunakan bersama-sama dengan aplikasi Java yang lain). Bahasa pemrograman Java juga telah berkembang menjadi bahasa pemrograman pada *browser* dengan adanya teknologi Java Applet yang memungkinkan pembuatan aplikasi yang interaktif di halaman *web*.

5.1 ARSITEKTUR SMARTWALLET

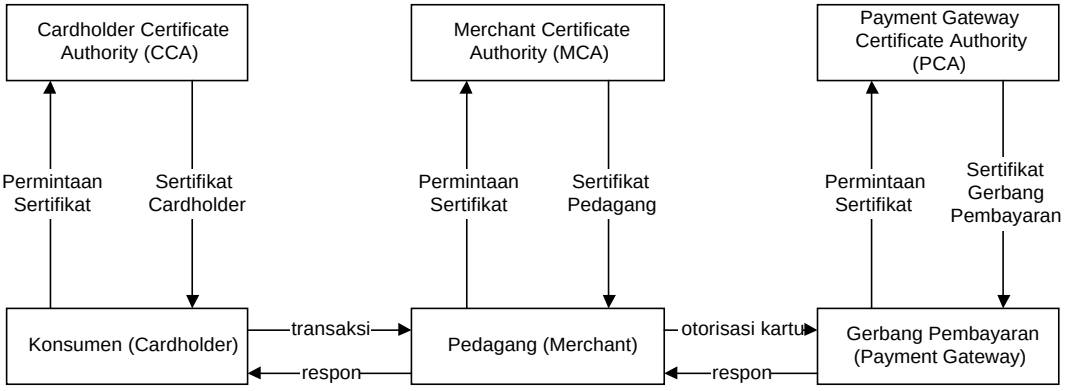
SmartWallet adalah Java Applet yang berbasis *Smartcard* dan protokol SET. Aplikasi SmartWallet terdiri atas beberapa modul yang saling melengkapi satu sama lain. Modul-modul ini dapat dilihat pada arsitektur SmartWallet berikut ini.



Gambar V.1. Arsitektur SmartWallet

Penelitian ini tidak mengimplementasikan semua bagian / modul di atas. Berikut ini adalah modul-modul yang menggunakan implementasi yang sudah ada:

- Modul SET menggunakan modul yang telah diimplementasi pada penelitian yang lain. Dwinanda Prayudi [DAPI99] mengimplementasikan protokol SET antara konsumen (*cardholder*) dan pedagang (*merchant*), Arif Prihasanta [AFPA99] mengimplementasikan protokol SET antara pedagang (*merchant*) dan gerbang pembayaran (*payment gateway*), serta Haris Fauzi [HSFI99] mengimplementasikan protokol SET antara konsumen (*cardholder*) dan *Certificate Authority* (CA). Gambar V.2 memperlihatkan hubungan antar modul pada protokol SET.
- Modul Kriptografi menggunakan modul yang dibuat oleh *Australian Business Association* (ABA) dan disimpan dalam *package Java Cryptography Extension* (JCE) 1.2 dari Sun Microsystem.
- *Java Virtual Machines* (JVM) menggunakan Java v1.2.2, merupakan *interpreter* Java yang dibuat oleh Sun Microsystem.
- Sistem Operasi yang digunakan adalah Microsoft Windows 98/NT4.



Gambar V.2. Diagram Protokol SET

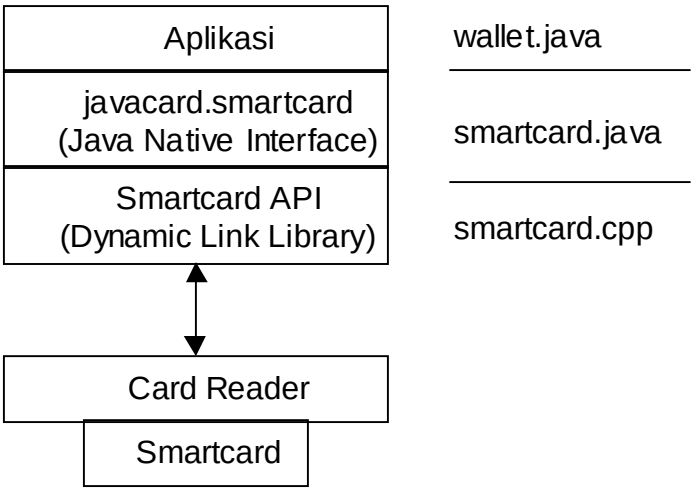
5.1.1 Smartcard

Smartcard digunakan sebagai media penyimpanan alternatif selain *hardisk*. Penggunaan *smartcard* sebagai media alternatif disebabkan kelebihan-kelebihan yang dimilikinya, yaitu:

- Keamanan yang lebih terjamin, yaitu *smartcard* merupakan sebuah *tamper device* yang menyulitkan pencurian data-data di dalamnya,
- Fleksibel, yaitu mudah untuk dibawa kemana-mana,

- Pemakai dapat melakukan transaksi di komputer manapun yang mempunyai *smartcard device*,
- *Smartcard* dapat digunakan juga sebagai kartu identitas atau tanda pengenal,
- Penggunaan *smartcard* oleh beberapa aplikasi (*multi application*), yang dapat digunakan secara bersama-sama oleh aplikasi-aplikasi yang berbeda, misalnya Access Card, Health Card, dan Transport Card.

Data-data dikirimkan ke *smartcard* dengan *format* APDU (*Application Protocol Data Unit*). Ada dua jenis APDU yaitu Command APDU (*reader*-kartu) dan Response APDU (kartu-*reader*). Pemrograman APDU ini selanjutnya disimpan dalam sebuah *library* yang berupa *native code* dalam bentuk *Dynamic Link Library* (DLL). *Java Native Interface* (JNI) digunakan sebagai antar muka pemrograman Java yang memanggil file DLL. Paket file JNI ini (*javacard.smartcard*) kemudian digunakan sebagai modul untuk pemrograman *smartcard* menggunakan bahasa Java.



Gambar V.3. Arsitektur *Smartcard*

Smartcard yang digunakan dapat bervariasi tergantung dari kebutuhan pemakai, meskipun saat implementasi hanya menggunakan *microprocessor card* dengan ukuran memori 2048 bytes (2 KB). Alternatif *smartcard* yang dapat digunakan adalah:

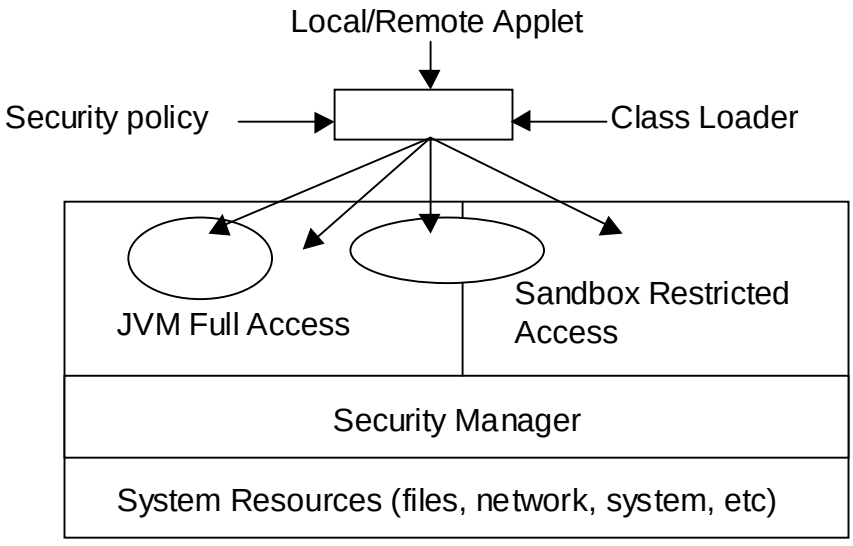
1. *Memory cards*. Kartu jenis ini hanya berfungsi untuk menyimpan data, tidak mempunyai prosesor atau sistem keamanan yang canggih melainkan hanya perlindungan fisik (karena *smartcard* bersifat *tamper proof*).
2. *Memory protected cards*. Kartu jenis ini mempunyai sistem keamanan yang lebih canggih

daripada *memory cards*, misalnya mekanisme *password* untuk mengakses kartu.

3. *Microprocessor cards*. Kartu Jenis ini mempunyai prosesor sehingga dapat melakukan komputasi walaupun terbatas. Kemampuannya antara lain mengorganisasikan berkas (*file*) yang dilindungi dengan *password*.
4. *Java cards*. Kartu ini dilengkapi dengan *Java Virtual Machine* sedemikian hingga dapat dimasukkan berbagai program ke dalamnya.
5. *Public key cards*. Kartu ini mendukung *public key cryptography* (kriptografi asimetri) sehingga proses enkripsi / dekripsi dapat dilakukan secara internal dan dapat menyimpan kunci asimetri.

5.1.2 Modul Policy

SmartWallet menggunakan teknologi Java Applet, akibatnya perlu suatu teknik tertentu agar applet ini diterima oleh sistem keamanan *browser*. Modul *policy* ini digunakan untuk memberi kepercayaan kepada SmartWallet agar dapat mengakses sumber daya lokal yang ada di komputer *client*. Kepercayaan yang diberikan kepada SmartWallet adalah hak untuk membuat / mengakses *file* di direktori tertentu, hak untuk mengakses *library*, hak untuk membaca *user profile*, dan hak untuk melakukan komunikasi jaringan (*network*).



Gambar V.4. Modul Policy pada JDK 1.2

Setiap applet yang dijalankan oleh *class loader* mempunyai keterbatasan untuk mengakses sumber daya komputer (*system resources*). Secara default, *applet-applet* tersebut tidak mempunyai hak sama sekali (*sandbox restricted access*). Namun dengan adanya modul *policy*, kita dapat mengatur hak-hak yang dapat dimiliki oleh masing-masing *applet* tersebut. Berdasarkan hak-hak tersebut, *applet-applet* ini diberi kebebasan oleh *security manager* untuk mengakses sumber daya di komputer lokal.

HARDISK	SMARTCARD
Tidak praktis jika selalu dibawa ke mana-mana.	Praktis untuk dibawa ke mana-mana.
Relatif lebih mudah untuk mencuri data di dalamnya.	Merupakan *tamper proof device* sehingga sulit untuk mencuri data di dalamnya.
Konsumen hanya dapat melakukan transaksi di komputer yang menyimpan informasi miliknya.	Konsumen dapat melakukan transaksi di komputer manapun yang mempunyai *smartcard device*.
Hardisk tidak dapat digunakan sebagai tanda pengenal.	*Smartcard* dapat digunakan untuk fungsi lain seperti kartu identitas.
Dapat menyimpan data dalam ukuran yang besar.	Menyimpan data dengan ukuran tertentu, tergantung ukuran *smartcard* (2KB, 4KB, 8KB, 16KB, 32KB, atau 64KB).

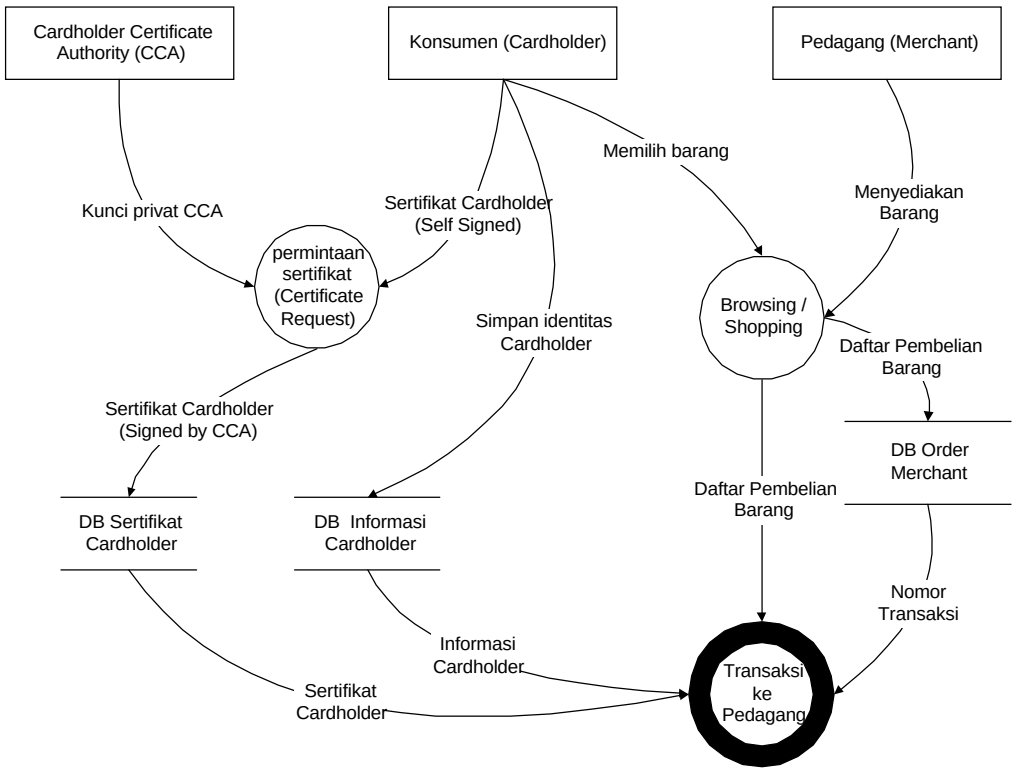
Tabel V.1 Perbandingan Hardisk dan Smartcard

5.2 ANALISA DAN PERANCANGAN PROSES

Bagian ini menjelaskan alur proses pada aplikasi SmartWallet. Proses yang dilakukan pada aplikasi ini dapat dikategorikan menjadi empat bagian, yaitu :

1. **Pembuatan identitas baru**. Konsumen baru yang menggunakan aplikasi ini harus memasukkan identitas diri dan informasi kartu kredit miliknya untuk disimpan dalam media penyimpanan sehingga konsumen tersebut tidak perlu memasukkan lagi informasi tersebut pada transaksi berikutnya, tetapi cukup dengan membuka informasi yang telah disimpannya.

2. **Membuka Identitas**. Konsumen dapat membuka identitas yang telah disimpan dalam media penyimpanan untuk melakukan proses permintaan sertifikat atau melakukan transaksi.
3. **Permintaan Sertifikat**. Konsumen yang belum disahkan oleh badan otoritas sertifikat (Certificate Authority / CA) perlu meminta (*request*) tanda tangan dari CA tersebut agar dipercaya oleh pedagang (*merchant*) dan gerbang pembayaran (*payment gateway*).
4. **Transaksi**. Konsumen yang sudah melakukan kedua proses di atas dapat melakukan transaksi dengan pedagang.



Gambar V.5. Diagram Aliran Data pada SmartWallet

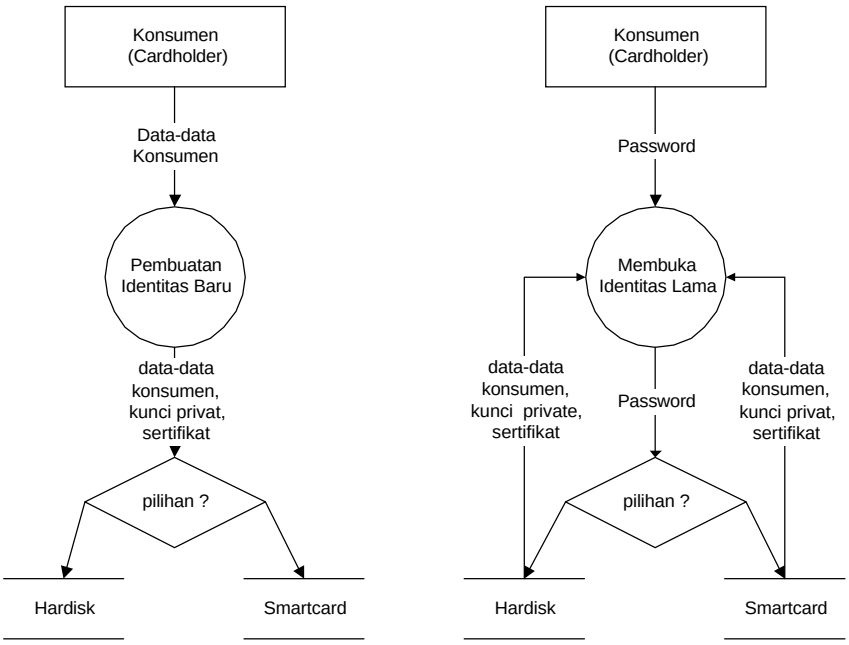
5.2.1 Pembuatan Identitas Baru

Konsumen yang akan melakukan transaksi harus memasukkan identitasnya untuk disimpan dalam media penyimpanan. Identitas tersebut akan digunakan pada transaksi selanjutnya sehingga konsumen tidak perlu memasukkan identitasnya untuk yang kedua kalinya. Data-data yang perlu dimasukkan oleh konsumen saat inisialisasi pertama kali adalah :

1. Nama (*name*), digunakan untuk identitas pada kartu dan sertifikat.

2. Jenis pembayaran (*payment type*), yaitu jenis pembayaran yang digunakan pada kartu, terdiri atas kartu kredit (*credit card*) dan kartu debit (*debit card*).
3. Nama kartu (*card name*), yaitu nama kartu yang digunakan, misalnya American Express, Visa Card, Master Card, dan Dinners Club Card.
4. Nomor kartu (*card number*), yaitu nomor kartu yang bersangkutan, misalnya 7813-2312-1123-2894.
5. Masa habis berlakunya kartu (*card expiry*), yaitu bulan dan tahun berakhirnya masa berlakunya kartu, misalnya January 2005.
6. Kode pengenal (*password*), digunakan untuk membuka identitas dari media penyimpanan saat digunakan untuk transaksi.
7. Unit organisasi (*organization unit*), yaitu nama unit organisasi, misalnya bidang keuangan.
8. Nama organisasi (*organization name*), yaitu nama organisasi, misalnya Universitas Indonesia.
9. Nama lokasi (*locality name*), yaitu nama lokasi dari organisasi, misalnya Depok.
10. Nama daerah (*state name*), yaitu nama daerah atau propinsi, misalnya Jawa Barat.
11. Identitas negara (*country*), yaitu identitas negara yang digunakan secara internasional, misalnya ID untuk Indonesia.

Setelah konsumen selesai memasukkan data-data tersebut, ia harus melakukan pembuatan kunci privat dan sertifikat dengan menekan tombol “Generate Key” yang secara otomatis membuat kunci privat RSA 1024-bit dan sertifikat digital MD5RSA 1024-bit. Selanjutnya konsumen dapat memilih media penyimpanan yang digunakan untuk menyimpan data-data tersebut yaitu *hardisk*, *smartcard*, atau kedua-duanya.



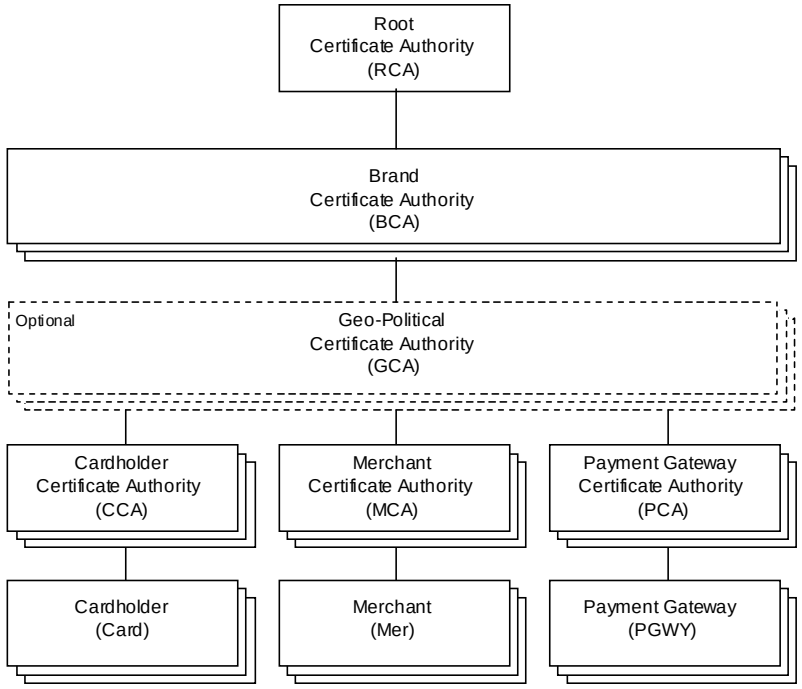
Gambar V.6. Proses menyimpan dan membuka identitas konsumen

5.2.2 Membuka Identitas

Konsumen dapat membuka identitas yang telah disimpan sebelumnya dalam media penyimpanan. Kode pengenal (*password*) harus dimasukkan oleh konsumen untuk membuka identitas tersebut. Media penyimpanan yang digunakan juga dapat dipilih oleh konsumen untuk membuka identitas yang dimilikinya. Setelah membuka identitasnya, konsumen dapat menggunakan identitas tersebut untuk transaksi. Jika sertifikat digital milik konsumen belum ditanda tangani oleh CA maka konsumen dapat meminta (*request*) sertifikat ke CA sebelum melakukan transaksi.

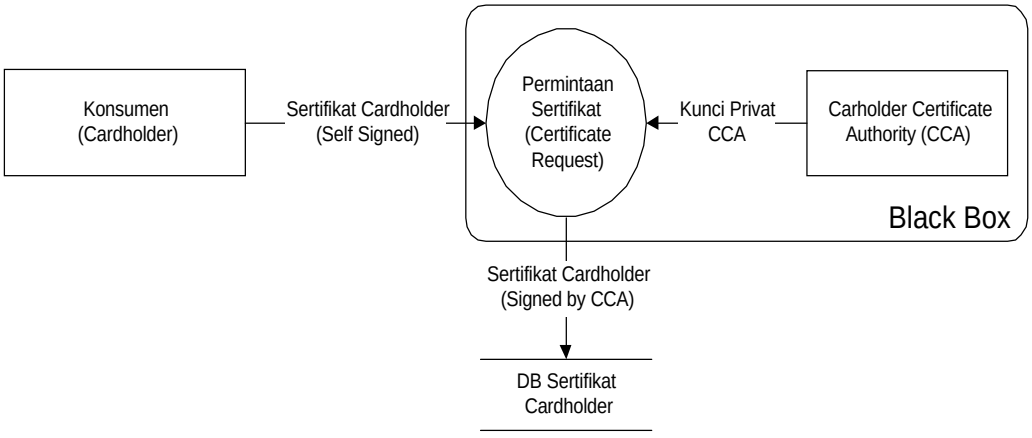
5.2.3 Permintaan Sertifikat (Certificate Request)

Konsumen perlu meminta tanda tangan digital dari badan otoritas sertifikat digital (*Certificate Authority* / CA). Sertifikat konsumen yang sudah ditanda tangani oleh CA dapat digunakan sebagai tanda pengenal milik konsumen untuk melakukan transaksi dengan pedagang. Gambar berikut memperlihatkan arsitektur manajemen sertifikat yang merupakan struktur hirarki sertifikat internasional. Permintaan sertifikat yang diimplementasi disini adalah permintaan sertifikat yang dilakukan oleh konsumen ke *Cardholder Certificate Authority* (CCA).



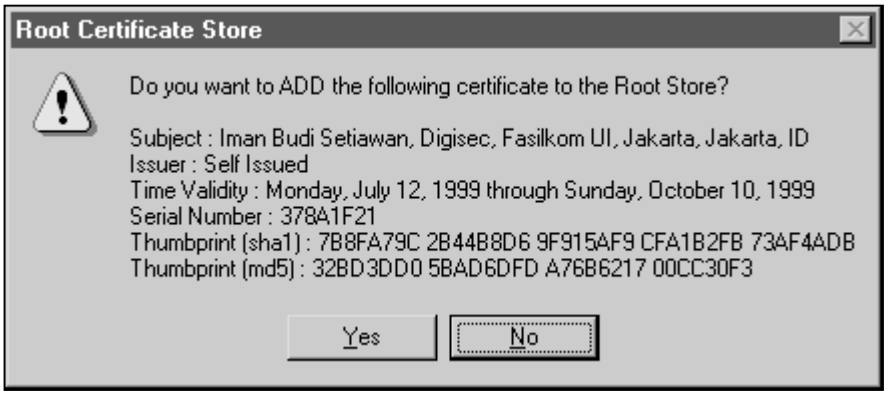
Gambar V.7. Arsitektur Manajemen Sertifikat

Permintaan sertifikat itu sendiri terdiri dari beberapa pesan (*message*) yang saling dipertukarkan oleh konsumen (*cardholder*) dan CCA. Pesan-pesan yang saling dipertukarkan antara konsumen dengan CCA dapat dikategorikan menjadi dua jenis yaitu permintaan (*request*) dari konsumen dan jawaban (*response*) dari CCA.



Gambar V.8. Permintaan Sertifikat oleh Cardholder

Penelitian tugas akhir ini membatasi proses permintaan sertifikat sampai sertifikat siap dikirim ke CCA (gambar V.8). Permintaan sertifikat dilakukan dalam suatu *format* standar sertifikat ITU (*International Telecommunication Union*) X.509 yang telah ditandatangani oleh konsumen sendiri (*self signed*). Proses permintaan sertifikat (*Certificate Request*) dari konsumen ke CCA dilakukan dengan menggunakan protokol SET dan digambarkan sebagai *black box* pada gambar di atas. Proses ini merupakan penelitian tugas akhir yang dilakukan oleh Haris Fauzi [HSFI99]. CCA akan menandatangani sertifikat konsumen kemudian sertifikat yang sudah ditandatangani CCA tersebut dikembalikan ke konsumen. Kemudian konsumen menyimpan sertifikat yang sudah ditandatangani oleh CCA dalam media penyimpanan yang dimilikinya atau langsung menggunakan sertifikat tersebut untuk melakukan transaksi. Gambar berikut ini memperlihatkan penambahan sertifikat yang dapat dipercaya pada sistem komputer berbasis Microsoft Windows.



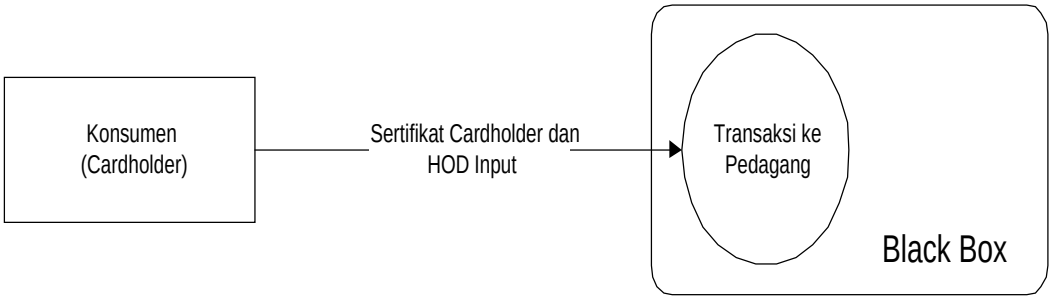
Gambar V.9. Penambahan Sertifikat pada Microsoft Windows

5.2.4 Transaksi

Aplikasi SmartWallet digunakan untuk melakukan transaksi perdagangan di Internet. Transaksi tersebut dapat diuraikan sebagai berikut :

1. Konsumen memilih (*browsing*) barang-barang yang akan dibelinya dan menekan tombol "ORDER" untuk melakukan transaksi.
2. *Browser* akan mengirim data-data transaksi ke pedagang dan menunggu jawaban dari pedagang yang berisi nomor transaksi.
3. Pedagang menerima informasi pembelian dari pedagang dan membuat nomor transaksi.
4. Setelah menerima nomor transaksi, *browser* membuka aplikasi SmartWallet.
5. Informasi mengenai pembelian dapat dilihat oleh konsumen pada aplikasi ini.

6. Konsumen membuka informasi kartu kredit miliknya yang telah disimpan sebelum dengan memasukkan *password*.
7. Konsumen menekan tombol “TRANSACTION” untuk melakukan transaksi.
8. Aplikasi SmartWallet akan membuat sebuah objek HODInput yang berisi data-data transaksi yang akan dikirimkan ke pedagang.
9. Aplikasi SmartWallet mengirimkan objek HODInput dan sertifikat konsumen yang sudah ditandatangani CCA ke proses SET antara konsumen (*cardholder*) dan pedagang (*merchant*).



Gambar V.10. Pengiriman Data Transaksi oleh Konsumen

Berikut ini tipe HOD input yang didefinisikan dalam ASN.1 :

```
HODInput ::= SEQUENCE {
        od                      OD,
        purchAmt                CurrencyAmount,
        odSalt                  Nonce,
        installRecurData  [0]   InstallRecurData  OPTIONAL,
        odExtensions      [1]   MsgExtensions {{ODExtensionsIOS}} OPTIONAL
}
 OD ::= OCTET STRING                    -- Order description
 Nonce ::= OCTET STRING (SIZE(20))
```

Penelitian tugas akhir ini membatasi pengiriman transaksi dengan mengirimkan objek HODInput ke proses SET antara konsumen dan pedagang (gambar V.9). Proses SET yang terjadi antara konsumen dan pedagang digambarkan sebagai *black box* pada gambar di atas. Proses ini menggunakan implementasi yang dibuat oleh Dwinanda Prayudi [DAPI99]. Proses ini akan mengirimkan pesan ke SmartWallet jika transaksi telah selesai atau mengalami kegagalan saat melakukan transaksi.

5.3 PERANCANGAN FILE / BASIS DATA

Media penyimpanan yang digunakan adalah *hardisk* dan *smartcard* menggunakan *format* penyimpanan *Distinguished Encoding Rules* (DER) yang merupakan standar penyimpanan pada protokol SET.

Data-data yang disimpan dalam media penyimpanan dapat dibedakan menjadi tiga jenis, yaitu:

1. *File* yang berisi informasi kartu kredit milik konsumen (.swt). *File* ini dienkripsi menggunakan algoritma Triple-DES (DESEDE) 168-bit kemudian disimpan dalam format penyimpanan DER Encode.

DB Informasi Kartu Konsumen					
Header	Nama	Jenis Pembayaran	Nama Kartu Pembayaran	Nomor Kartu Pembayaran	Akhir Berlaku Kartu

Gambar V.11. Struktur Data Informasi Kartu

Struktur penyimpanan *file* ini adalah sebagai berikut :

- Header (3 bytes) : tanda pengenal *file wallet*, yaitu “SWT”.
- Nama (20 bytes) : berisi nama konsumen yang mempunyai kartu pembayaran.
- Jenis Pembayaran (20 bytes) : jenis kartu pembayaran milik konsumen.
- Nama Kartu Pembayaran (20 bytes) : nama kartu pembayaran milik konsumen.
- Nomor Kartu Pembayaran (20 bytes) : nomor kartu pembayaran milik konsumen.
- Akhir Berlaku kartu (6 bytes) : waktu habisnya kartu pembayaran (*expiry date*).

2. *File* yang berisi kunci privat milik konsumen (.key). *File* ini menggunakan standar PKCS-7 dan dienkripsi menggunakan algoritma Triple-DES (DESEDE) 168-bit kemudian disimpan menggunakan format standar DER Encode.

DB Kunci Privat Konsumen (PKCS #8)			
Versi	Algoritma Identifier	Kunci Privat (Octet String)	Atribut

Gambar V.12. Struktur Data Kunci Privat Konsumen

Struktur file kunci privat berdasarkan PKCS-7 dapat direpresentasikan dengan ASN.1 sebagai

berikut :

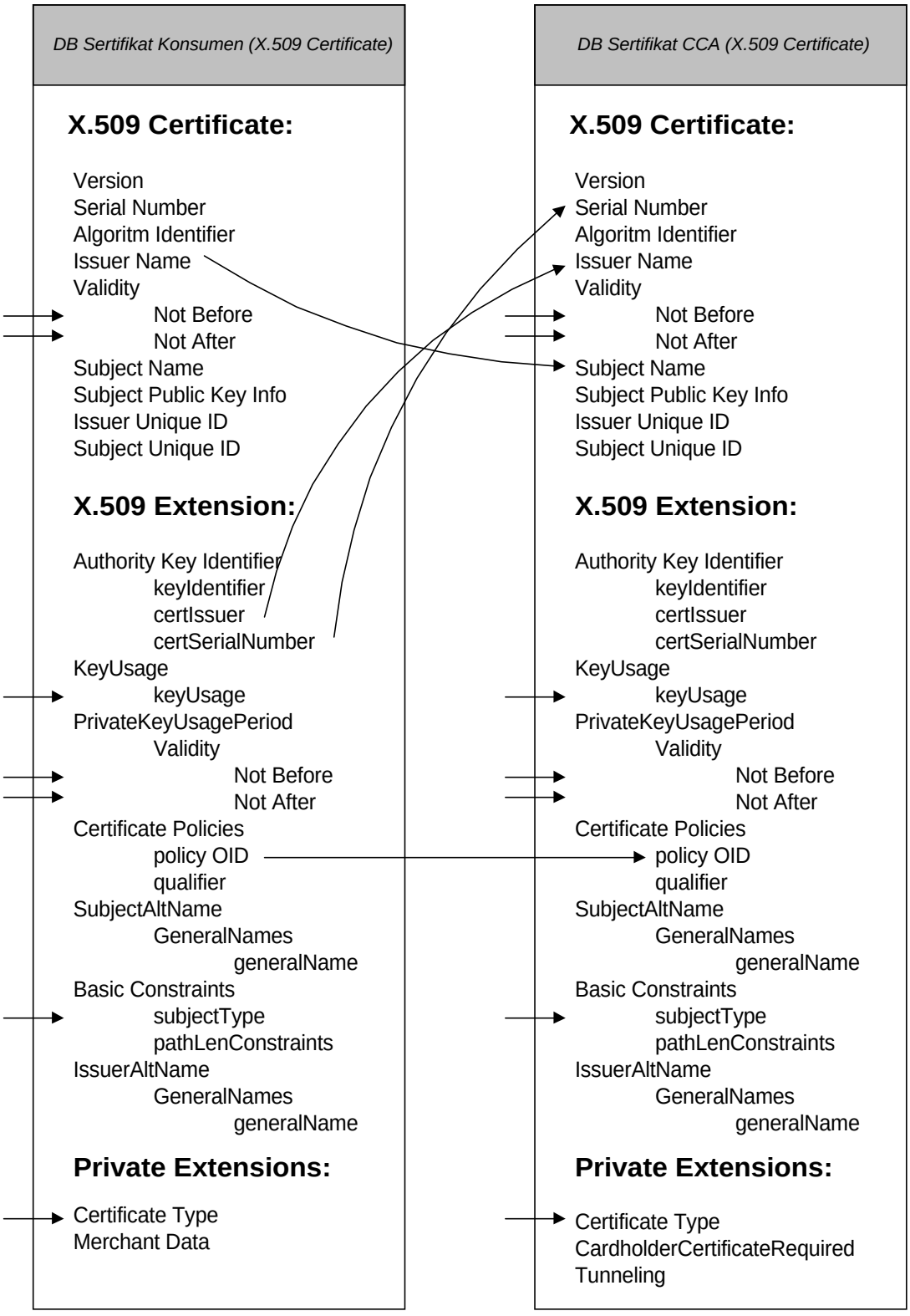
```
PrivateKeyInfo ::= SEQUENCE {
        version Version,
        privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,
        privateKey PrivateKey,
        attributes [0] IMPLICIT Attributes OPTIONAL
}
Version ::= INTEGER
PrivateKeyAlgorithmIdentifier ::= AlgorithmIdentifier
PrivateKey ::= OCTET STRING
Attributes ::= SET OF Attribute
```

Berikut ini adalah struktur kunci privat yang terenkripsi dalam format ASN.1 :

```
EncryptedPrivateKeyInfo ::= SEQUENCE {
        EncryptionAlgorithm EncryptionAlgorithmIdentifier,
        EncryptionData EncryptedData
}
EncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
EncryptedData ::= OCTET STRING
```

3. File yang berisi sertifikat milik konsumen (.cer). File ini menggunakan standar ITU (*International Telecommunication Union*) X.509 kemudian disimpan dalam format standar DER Encode.

Ketiga *file* ini disimpan dalam *hardisk* pada direktori ${user.home} agar tidak dapat dicuri oleh pemakai lain. Enkripsi yang dilakukan pada *file* pertama dan kedua bertujuan agar pencuri tidak dapat menggunakan *file* jika berhasil mencuri *file-file* tersebut. File sertifikat tidak dienkripsi karena *file* ini untuk diberikan pada orang lain yang akan berkomunikasi dengan konsumen. Penyimpanan dengan *smartcard* lebih aman lagi, selain sulit untuk mendapatkan data yang telah dikunci dengan PIN, data-data tersebut juga masih dienkripsi dengan Triple-DES 168-bit. Selain itu, konsumen dapat melakukan transaksi di komputer mana saja yang mempunyai *smartcard reader*.



Gambar V.13. Struktur Data Sertifikat X.509

Struktur *file* sertifikat menurut standar X.509 dapat direpresentasikan dengan ASN.1 sebagai berikut :

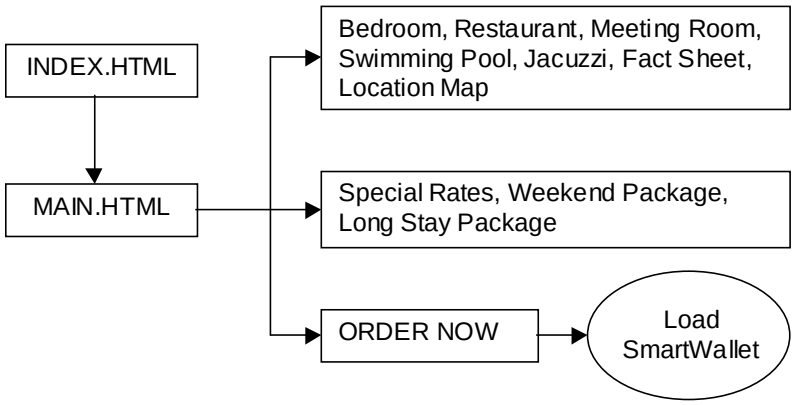
```
Certificate ::= SEQUENCE {
        tbsCertificate TBSCertificate,
        signatureAlgorithm AlgorithmIdentifier,
        signature BIT STRING
}

TBSCertificate ::= SEQUENCE {
    version [0] EXPLICIT Version DEFAULT v1,
    serialNumber CertificateSerialNumber,
    signature AlgorithmIdentifier,
    issuer Name,
    validity Validity,
    subject Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
      -- If present, version must be v2 or v3
    subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,
      -- If present, version must be v2 or v3
    extensions [3] EXPLICIT Extensions OPTIONAL
      -- If present, version must be v3
}
```

5.4 PERANCANGAN ANTAR MUKA (USER INTERFACE)

Perancangan antar muka (*user interface*) merupakan salah satu bagian yang penting dalam perancangan aplikasi, terutama aplikasi yang berhubungan langsung dengan pemakai.

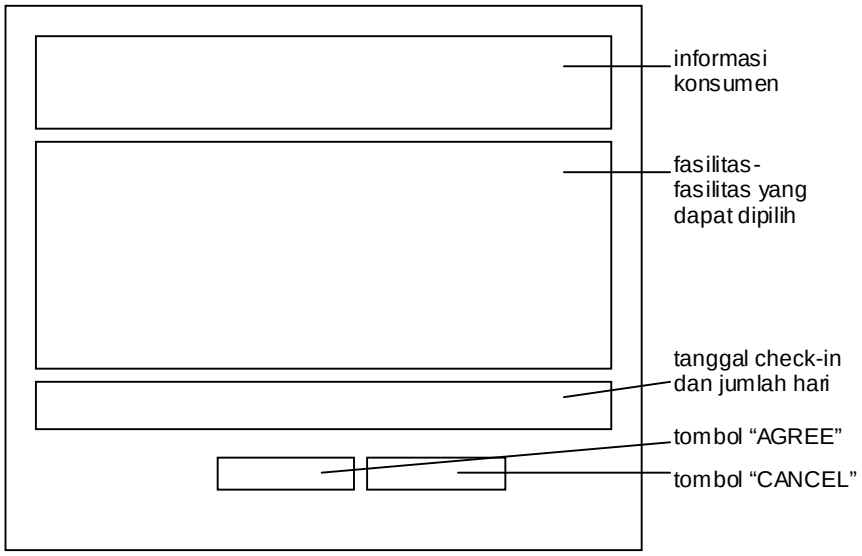
5.4.1 Store User Interface

Store user interface adalah halaman *web* yang digunakan oleh konsumen untuk melakukan pemilihan (*browsing*) barang-barang yang akan dibelinya. Antar muka ini dibuat sebagai contoh yang diberikan untuk menjalankan aplikasi SmartWallet, jadi bukan tujuan utama pada penelitian tugas akhir ini. Pedagang dapat membuat sendiri antar muka untuk dagangannya kemudian menggunakan SmartWallet untuk melakukan transaksi yang berbasis protokol SET.



Gambar V.14. Struktur hubungan antar halaman di website pedagang

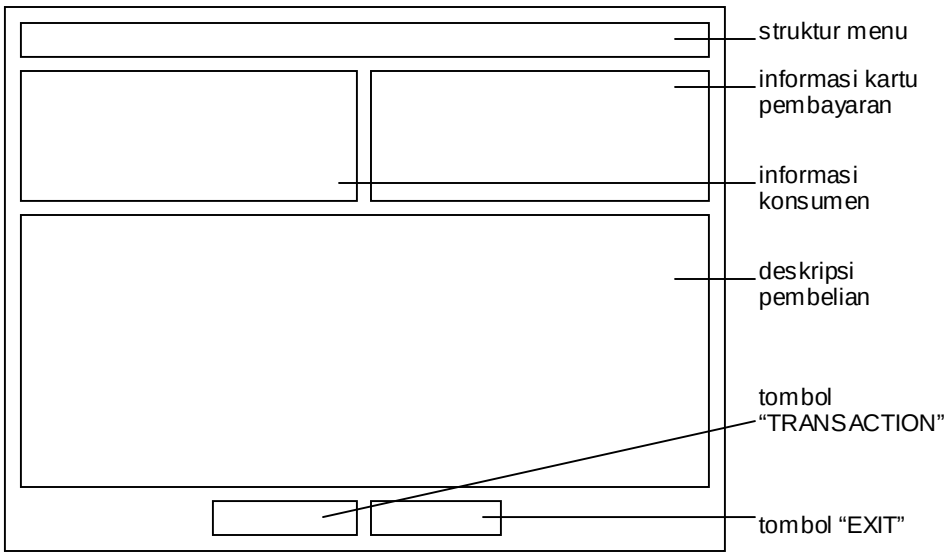
Gambar di atas memperlihatkan hubungan antar halaman saat konsumen melakukan *browsing* di *website* pedagang. Pertama kali konsumen memasuki halaman pertama yang berisi informasi mengenai *website* tersebut kemudian konsumen dapat masuk ke halaman utama yang mempunyai link ke fasilitas-fasilitas yang tersedia. Konsumen dapat melihat berbagai macam fasilitas yang disediakan pedagang selanjutnya konsumen dapat menuju halaman "ORDER NOW" untuk memilih fasilitas-fasilitas yang dibutuhkannya. Setelah konsumen memilih fasilitas-fasilitas tersebut, ia dapat menekan tombol "AGREE" untuk melakukan transaksi. *Browser* akan membuka aplikasi SmartWallet yang akan mengatur proses transaksi selanjutnya.



Gambar V.15. Halaman "ORDER NOW"

5.4.2 Wallet User Interface

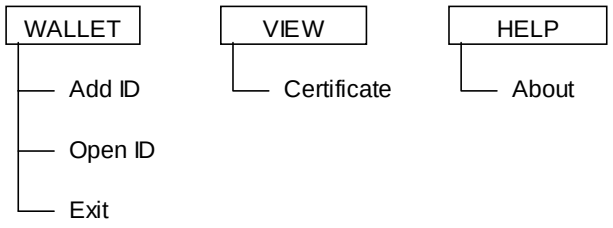
Wallet user interface adalah antar muka dari aplikasi SmartWallet yang berhubungan langsung dengan konsumen. Empat buah komponen utama SmartWallet yang merupakan antar muka dengan konsumen adalah antar muka utama, antar muka untuk penambahan identitas, antar muka untuk membuka identitas, dan antar muka untuk permintaan sertifikat.



Gambar V.16. Perancangan Antar Muka Utama SmartWallet

Gambar di atas merupakan antar muka utama dari aplikasi SmartWallet. Antar muka ini akan ditampilkan setelah konsumen menekan tombol "ORDER" saat melakukan *browsing* di *website* perdagang. Antar muka ini terdiri dari beberapa bagian yaitu:

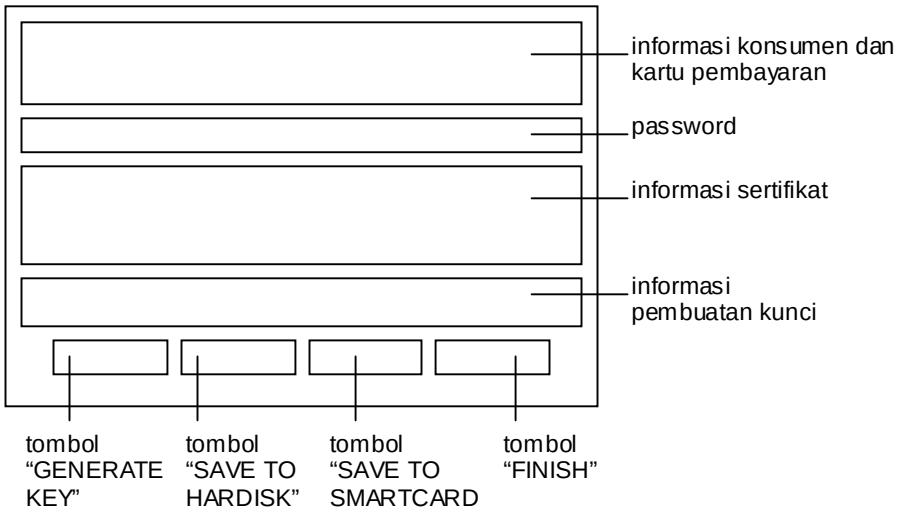
- struktur menu, yaitu struktur yang merupakan link menuju antar muka yang lain.
- informasi konsumen, yaitu informasi tentang konsumen dan transaksi yang terjadi. Informasi ini diperoleh dari *browser* saat konsumen melakukan pemilihan barang.
- informasi kartu pembayaran, yaitu informasi tentang kartu pembayaran yang digunakan. Informasi ini diperoleh dari media penyimpanan.
- deskripsi pembelian (order description), yaitu keterangan mengenai barang-barang yang dibeli.
- tombol "TRANSACTION", yaitu tombol untuk melakukan transaksi dengan pedagang.
- Tombol "EXIT", yaitu tombol untuk keluar dari aplikasi SmartWallet dan kembali ke *browser*.



Gambar V.17. Struktur menu SmartWallet

Antar muka yang kedua adalah antar muka untuk melakukan penambahan identitas pemakai. Bagian-bagian yang terdapat pada antar muka ini adalah:

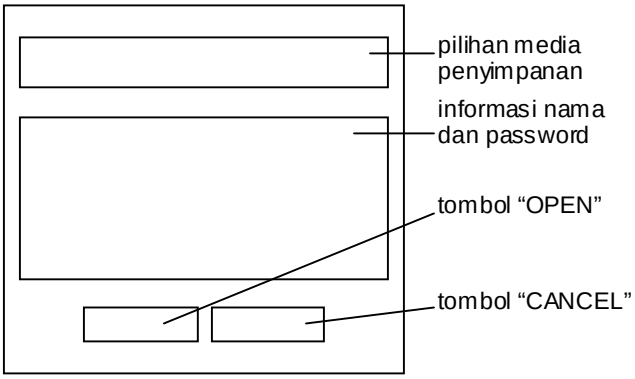
- informasi konsumen dan kartu pembayaran, berisi informasi konsumen dan kartu pembayaran miliknya yang akan disimpan dalam media penyimpanan.
- password, berisi password yang digunakan untuk mengenkripsi file penyimpanan.
- informasi sertifikat, berisi informasi konsumen yang ada dalam sertifikat digital.
- informasi pembuatan kunci, berisi keterangan apakah kunci sudah dihitung atau belum.
- tombol “GENERATE KEY”, yaitu tombol yang digunakan untuk menghitung kunci privat dan sertifikat.
- tombol “SAVE TO HARDISK”, yaitu tombol yang digunakan untuk menyimpan informasi ke dalam hardisk.
- tombol “SAVE TO SMARTCARD, yaitu tombol yang digunakan untuk menyimpan informasi ke dalam smartcard.
- tombol “FINISH”, yaitu tombol yang digunakan untuk menyelesaikan proses penambahan identitas.



Gambar V.18. Perancangan Antar Muka untuk Penambahan Identitas

Antar muka berikutnya adalah antar muka untuk membuka identitas yang sudah disimpan dalam media penyimpanan. Antar muka ini terdiri dari beberapa bagian:

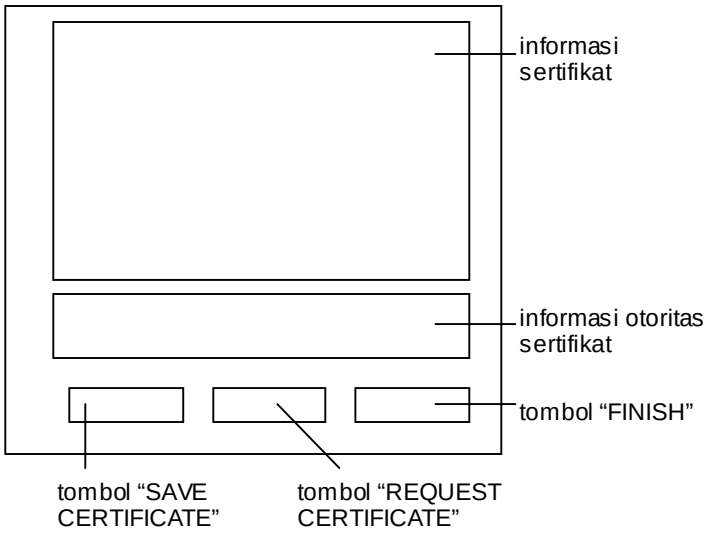
- pilihan media penyimpanan, berisi pilihan media penyimpanan yang akan digunakan.
- informasi nama dan password, berisi informasi nama-nama dari pemakai serta password yang harus dimasukkan untuk membuka informasi dari media penyimpanan.
- tombol “OPEN”, yaitu tombol yang digunakan untuk membuka identitas dari media penyimpanan yang sudah dipilih.
- tombol “CANCEL”, yaitu tombol yang digunakan untuk keluar dari antar muka ini.



Gambar V.19. Perancangan Antar Muka untuk Membuka Identitas

Antar muka yang terakhir adalah antar muka untuk melakukan permintaan sertifikat. Bagian-bagian pada antar muka ini antara lain:

- informasi sertifikat, berisi informasi sertifikat milik konsumen.
- informasi otoritas sertifikat, berisi nama dan alamat dari otoritas sertifikat yang menandatangani sertifikat konsumen.
- tombol “SAVE CERTIFICATE”, yaitu tombol untuk menyimpan sertifikat.
- tombol “REQUEST”, yaitu tombol untuk melakukan permintaan sertifikat.
- tombol “FINISH”, yaitu tombol untuk keluar dari antar muka ini.



Gambar V.20. Perancangan Antar Muka untuk Permintaan Sertifikat

BAB VI
IMPLEMENTASI

Bab ini membahas mengenai implementasi dari aplikasi SmartWallet. Implementasi yang dilakukan adalah implementasi *smartcard* Java API, fasilitas *browsing/shopping*, integrasi *wallet* dengan *browser*, pembuatan identitas baru, membuka *file* identitas, permintaan sertifikat digital ke *Cardholder Certificate Authority* (CCA), transaksi dengan pedagang, dan pembuatan modul *policy*.

6.1 SMARTCARD JAVA API

Smartcard Java API adalah kelas-kelas (*library*) yang digunakan untuk mengoperasikan *smartcard* dengan bahasa pemrograman Java. Smartcard Java API ini digunakan oleh aplikasi SmartWallet untuk menyimpan data-data konsumen di dalam *smartcard*. Fungsi-fungsi primitif yang dibuat dengan bahasa pemrograman C dihubungkan dengan bahasa pemrograman Java menggunakan Java Native Interface agar dapat berkomunikasi dengan aplikasi SmartWallet. Berikut ini adalah langkah-langkah yang diperlukan untuk menghubungkan kedua bahasa pemrograman tersebut:

1. Menuliskan fungsi-fungsi primitif yang diperlukan untuk mengoperasikan *smartcard* menggunakan format APDU (*Application Protocol Data Unit*) yang terdapat pada ISO-7816. Berikut ini adalah fungsi-fungsi primitif yang diimplementasikan (meskipun tidak semua fungsi-fungsi tersebut digunakan oleh aplikasi SmartWallet):
   - driveInit, berfungsi untuk melakukan inisialisasi *smartcard drive* pertama kali, memeriksa bahwa *drive* dan kartu telah terpasang dengan baik.
   - cardOn, berfungsi untuk menyalakan *power* kartu.
   - cardOff, berfungsi untuk memutuskan hubungan *power* kartu..
   - createFile, berfungsi untuk membuat *file* dalam *smartcard* dengan proteksi biasa (PIN).
   - createPurseFile, berfungsi untuk membuat *file purse* yang digunakan untuk menyimpan nilai tertentu.
   - openFile, berfungsi untuk membuka file tertentu dalam smartcard.

- getSerialNumber, berfungsi untuk mendapatkan nomor seri dari smartcard.
- writeFile, berfungsi untuk menuliskan string tertentu dalam kartu.
- readFile, berfungsi untuk membaca string dari kartu.
- eraseCard, berfungsi untuk menghapus isi smartcard (untuk kartu yang diproteksi harus memasukkan kunci lebih dahulu).
- status, berfungsi untuk mendapatkan status keberadaan sebuah kunci dalam kartu.
- React, berfungsi untuk mengaktifkan kembali kartu yang sudah diblok.
- WriteKey, berfungsi untuk menuliskan kunci utama (master key) dalam kartu.
- ChangeKey, berfungsi untuk mengganti kunci utama dalam kartu.
- KeyPress, berfungsi untuk memeriksa kebenaran kunci dalam sebuah kartu.
- WriteSC, berfungsi untuk menuliskan kunci rahasia (secret code) dalam kartu.
- ChangeSC, berfungsi untuk mengganti kunci rahasia dalam kartu.
- SCPress, berfungsi untuk memeriksa kebenaran kunci rahasia dalam sebuah kartu.
- GetRandomNumber, berfungsi untuk mendapatkan sebuah angka acak.

Di bawah ini diberikan contoh implementasi dari fungsi readFile :

```
JNIEXPORT jstring JNICALL Java_javacard_smartcard_readFile
        (JNIEnv *env, jobject obj, jshort recordNumber, jshort offset, jshort dataLength) {
         SHORT rc;
        UCHAR Ack, St1, St2, *str = (UCHAR*)"";

        // panggil APDU dengan ins=0xC6.
        rc = ASE_7816Receive ( 0, 0xC6, recordNumber, offset, dataLength, str, &Ack, &St1, &St2 );
        checkResponse (env, 0xC6, Ack, St1, St2);
        jstring dataBuff = env->NewStringUTF ((char *)str);
        free (str);
        return dataBuff;
}
```

2. Membuat interface dari fungsi-fungsi primitif tersebut dalam bahasa pemrograman Java dan memanggil file library menggunakan kelas System.loadLibrary. Berikut ini adalah interface dari fungsi-fungsi primitif tersebut:

```
private native void driveInit(short port, short timeOut);
```

```
private native char[] cardOn (short GardLen, short ResetLen);
private native void cardOff ();
private native void createFile (short name, short RA1, short numberOfRecord, short recordLength);
private native void createPurseFile (short name, short type, short RA1, short RA2, short secretCodeNumber);
private native void openFile (short name, short type);
private native char[] getSerialNumber();
private native void writeFile (short recordNumber, short offset, short dataLength, String data);
private native String readFile (short recordNumber, short offset, short dataLength);
private native void eraseCard();
private native int[] status (short secretCodeNumber);
private native void react (short crypt, String value);
private native void writeKey (short maxAttempt, String masterKey);
private native void changeKey (String newMasterKey);
private native void keyPress (short crypt, String masterKey);
private native void writeSC (short maxAttempt, short SCNumber, String secretCode);
private native void changeSC (short SCNumber, String newSecretCode);
private native void SCPress (short crypt, short SCNumber, String secretCode);
private native char[] getRandomNumber ();

// baca library smartcard.dll
static {
        System.loadLibrary("smartcard");
}
```

3. Membuat file header yang berfungsi untuk menghubungkan kelas interface di atas dengan file library. File ini dapat dibuat dengan program jni yang terdapat pada Java Development Kit 1.2. Berikut ini adalah format perintah yang digunakan untuk membuat file header tersebut :

   ```
   javah –jni smartcard
   ```

4. Membuat *file dynamic link library* (dll) dari fungsi-fungsi primitif di atas menggunakan compiler untuk *create library* dari Visual C++ dengan *format* sebagai berikut

   ```
   cl /Og /G5 /EHs smartcard.cpp -Ic:\jdk1.2\include -Ic:\jdk1.2\include\win32 -Fesmartcard.dll /MD /LD -nologo /link Ase32.lib
   ```

   Perintah ini membuat sebuah file *library* yaitu smartcard.dll yang dapat dijalankan dari bahasa pemrograman Java. File library ini diletakkan di direktori ${system.library}, misalnya pada windows terletak di direktori c:\windows\system.

5. Buat fungsi-fungsi dalam bahasa pemrograman Java yang mengimplementasikan *interface* yang sudah dibuat sebelumnya. Berikut ini diberikan implementasi dari fungsi ReadFile yang berfungsi untuk membaca isi sebuah *file* dalam *smartcard*:

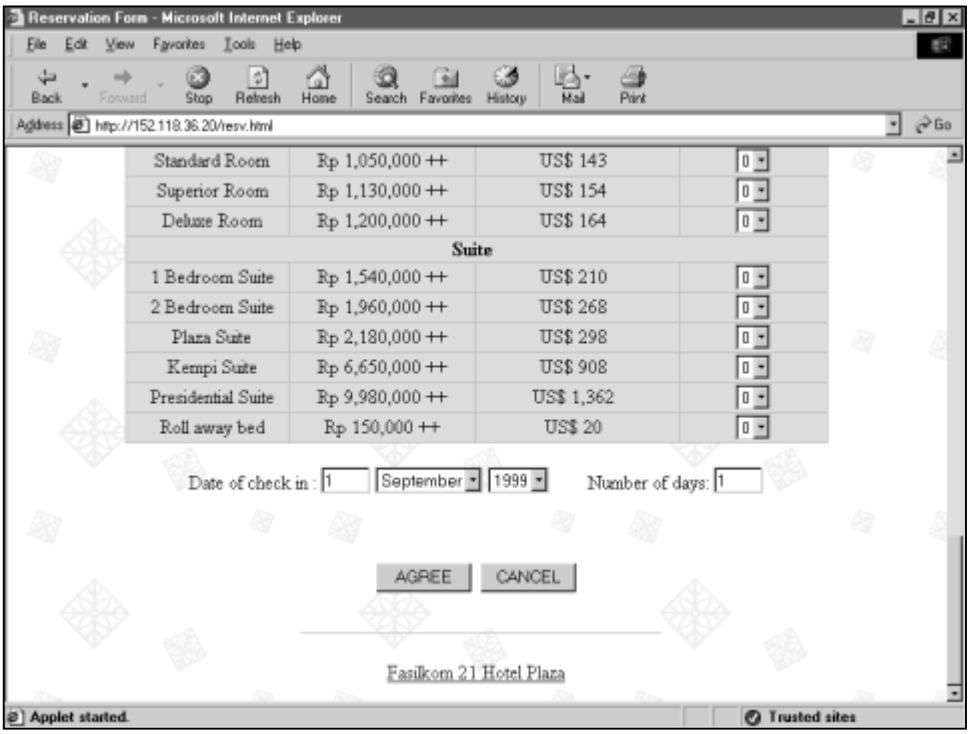
   ```
   public String read_File (short recordNumber, short offset, short dataLength)
           throws offsetException, dataLengthException {
           int i=0, begin=0, end=0, length;
           String data_temp, str="";

           if (offset+dataLength < 0 || offset+dataLength > 255)
                throw new offsetException("Not valid offset !");
           if (dataLength > 255)
                throw new dataLengthException("Not valid data length !");
   ```

```
            if (dataLength <= 8) {
                        str = readFile (recordNumber, offset, dataLength);
            } else {
                        for (i=0; end < dataLength; i++) {
                                    begin = i*8;
                                    offset = (short)(offset+(short)begin);
                                    if ((i+1)*8 < dataLength) end = (i+1)*8;
                                    else end = dataLength;
                                    length = end-begin;
                                    data_temp = readFile (recordNumber, offset, (short)length);
                                    str = str + data_temp;
                        }
            }
            return str;
}
```

6.2 FASILITAS BROWSING / SHOPPING

Konsumen melakukan pemilihan barang yang akan dibelinya melalui sebuah *browser* yang terhubung ke *website* pedagang Internet. Pada pemesanan kamar hotel secara *online*, konsumen dapat memilih kamar-kamar mana saja yang akan dipesannya. Setelah itu, daftar pemesanan dikirimkan ke pedagang dan aplikasi SmartWallet untuk melakukan transaksi.

Gambar VI.1. Halaman Utama dari Fasilitas Browsing



Gambar VI.2. Halaman Pemesanan Kamar Hotel

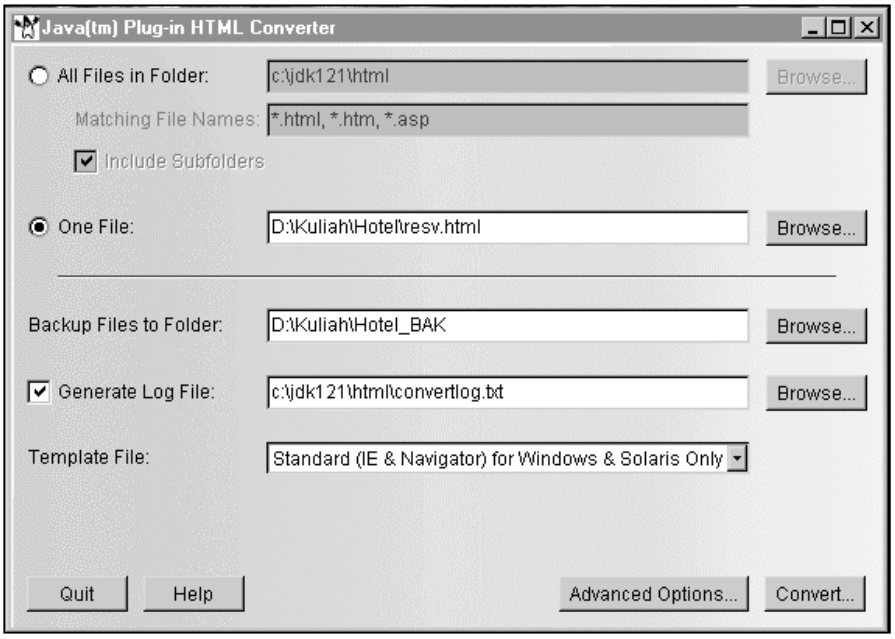
Halaman yang paling penting pada bagian ini adalah halaman yang digunakan untuk melakukan pemesanan kamar hotel. Halaman ini dibuat dengan dengan file HTML yang menggunakan *plug-ins* JRE 1.2.2. *Plug-ins* ini digunakan untuk menjalankan aplikasi SmartWallet setelah konsumen menekan tombol “AGREE”. Pemanggilan *plug-ins* tersebut dilakukan oleh file HTML dengan menggunakan *tag-tag* khusus pada file HTML. Internet Explorer menggunakan *tag* <OBJECT> untuk menggunakan ActiveX yang memanggil *plug-ins* JRE tersebut. Berikut ini pemanggilan *plug-ins* pada *browser* Internet Explorer:

```
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" WIDTH="300" HEIGHT="1"
NAME="menu" codebase="http://java.sun.com/products/plugin/1.1.2/jinstall-112-
win32.cab#Version=1,1,2,0">
<param name="CODE" value="Wallet">
<param name="ARCHIVE" value="Wallet.jar">
<param name="NAME" value="menu">
<param name="type" value="application/x-java-applet;version=1.1.2">
<param name="mayscript" value="true">
<param name="scriptable" value="true">
```

Netscape Navigator menggunakan *tag* <EMBED> untuk memanggil *plug-ins* JRE tersebut. Berikut ini pemanggilan *plug-ins* menggunakan *browser* Netscape Navigator:

```
<embed type="application/x-java-applet;version=1.1.2" java_CODE="Wallet" java_ARCHIVE="Wallet.jar"
NAME="menu" WIDTH="300" HEIGHT="1" mayscript="true" scriptable="true"
pluginspage="http://java.sun.com/products/plugin/1.1.2/plugin-install.html"><noembed>
```

Kedua jenis tag tersebut juga dapat dibuat menggunakan aplikasi tambahan yang ada pada Java 1.2 yaitu HTML Converter 1.2. Aplikasi ini secara otomatis akan menambahkan *tag-tag* tersebut pada file HTML yang dibuat meskipun perlu penambahan beberapa parameter lagi.



Gambar VI.3. Java Plug-in HTML Converter 1.2

SmartWallet menggunakan teknologi Java Applet sehingga Java Script digunakan untuk menjalankannya. Setelah semua pemesanan dihitung maka Java Script memanggil fungsi yang ada pada kelas Wallet. Teknologi ini disebut sebagai LiveConnect yang berfungsi untuk mengintegrasikan Java Script, Java Applet, dan *plug-ins*. Berikut ini adalah perintah yang digunakan untuk memanggil aplikasi SmartWallet:

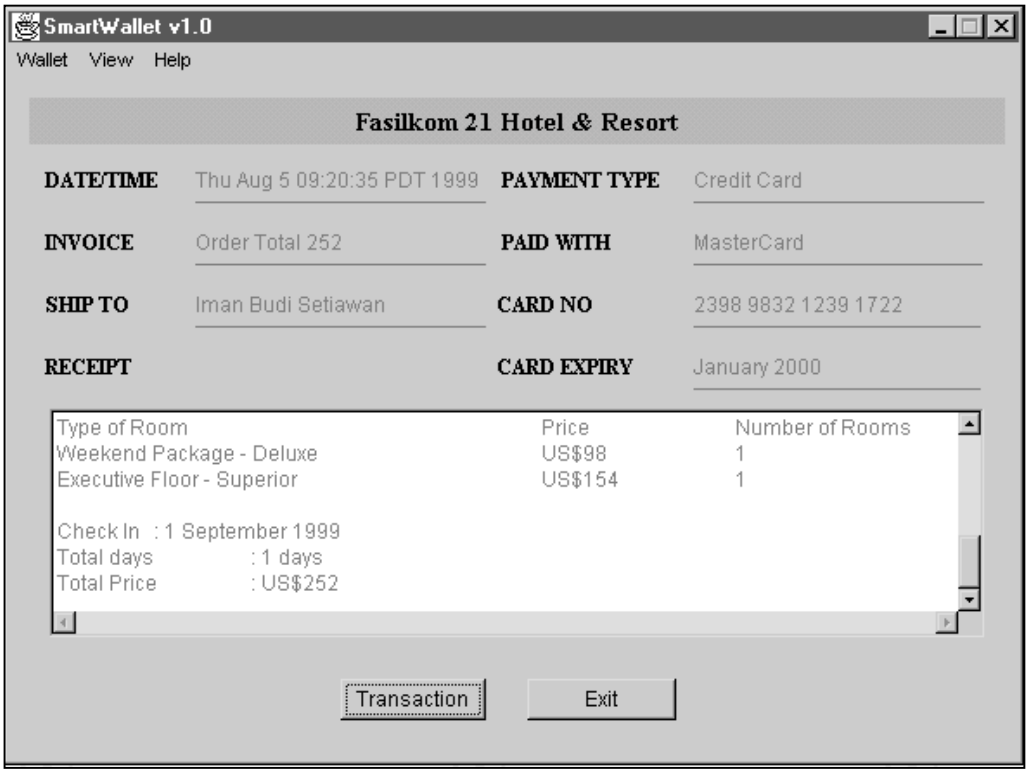
```
<SCRIPT language=”JavaScript”>
        var date, merchant, desc, total
        .....
        formulir.menu.startWallet(date, merchant, desc, total)
</SCRIPT>
```

6.3 INTEGRASI WALLET DENGAN BROWSER

SmartWallet merupakan sebuah applet yang dijalankan setelah konsumen menekan tombol "AGREE" di halaman transaksi pada *browser*. Sebenarnya setelah penekanan tombol tersebut, *browser* menjalankan fungsi startWallet yang terdapat pada kelas Wallet lebih dahulu. Data-data transaksi dikirim sebagai parameter-parameter dari fungsi startWallet. Fungsi ini membuat sebuah objek SmartWallet yang digunakan konsumen untuk melakukan transaksi.

```
public void startWallet(String date, String merchant, String desc, String total) {
        String transid;
        try {
                ….
                // buat kelas smartwallet
                SmartWallet sw = new SmartWallet(date, merchant, desc, total, transid);
                sw.setVisible(true);
        } catch(JSException jse) {System.err.println(jse);}
}
```

Potongan program di atas memperlihatkan objek SmartWallet yang dibuat dalam fungsi SmartWallet. Antar muka utama SmartWallet ditampilkan setelah objek SmartWallet tersebut dijalankan.



Gambar VI.4. Antar Muka Utama SmartWallet

6.4 PEMBUATAN IDENTITAS BARU

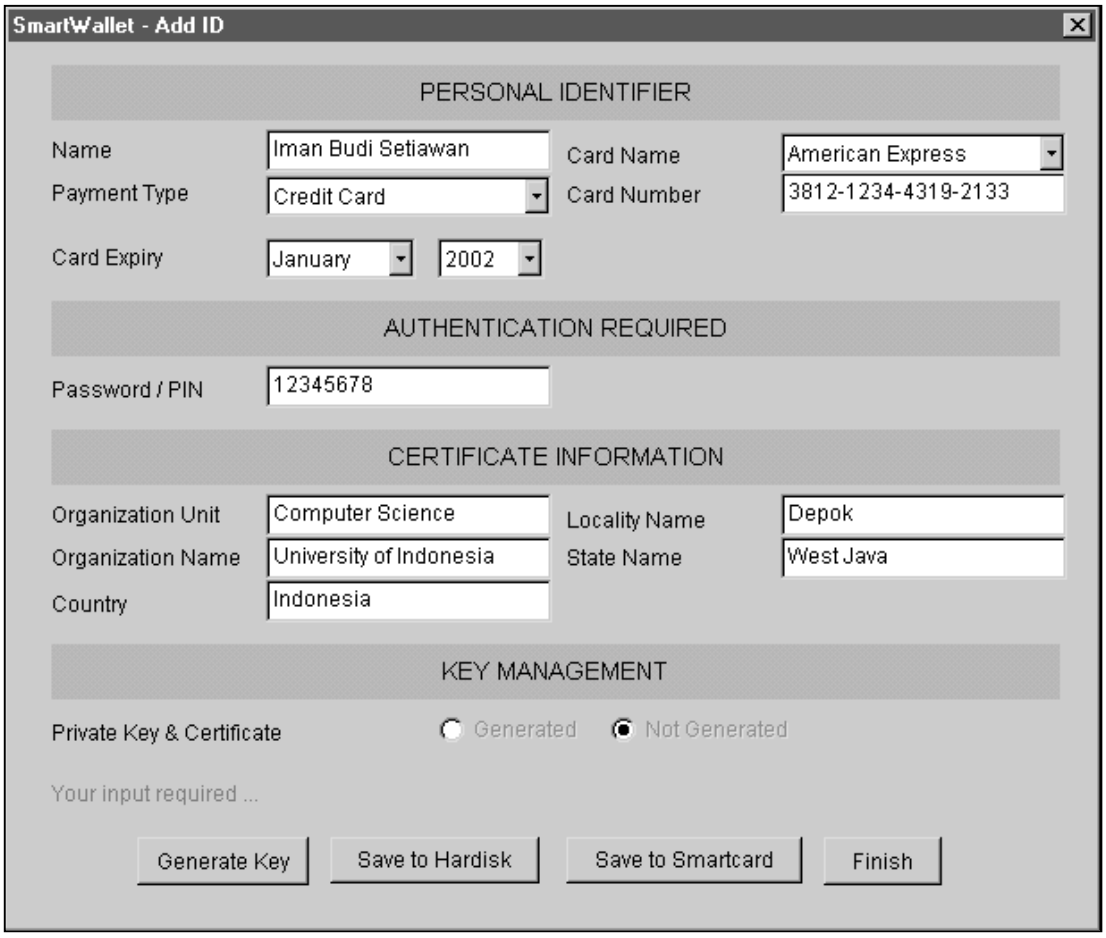
Fasilitas ini digunakan oleh konsumen yang baru pertama kali menggunakan SmartWallet. Konsumen memasukkan informasi tentang identitas dan kartu kreditnya kemudian disimpan dalam media penyimpanan. Data-data yang telah disimpan tersebut dapat digunakan untuk transaksi berikutnya. Sebelum melakukan penyimpanan informasi-informasi tersebut, konsumen harus membuat kunci privat dan sertifikat miliknya. Pembuatannya dilakukan oleh fungsi generateKeys setelah konsumen menekan tombol “GENERATE KEY”.

```
private void generateKeys() {
      try {
            int validity = 360;
            CertAndKeyGen certandkeygen = new CertAndKeyGen("RSA", "MD5WithRSA", "ABA");
            String myname = (txtName.getText()).trim();
            String myOU = (txtOU.getText()).trim();
            String myON = (txtON.getText()).trim();
            String myLN = (txtLN.getText()).trim();
            String mySN = (txtSN.getText()).trim();
            String myCO = (txtCO.getText()).trim();
            X500Name x500name = new X500Name(myname,myOU,myON,myLN,mySN,myCO);
            certandkeygen.generate(1024);
            privatekey = certandkeygen.getPrivateKey();
            X509Certificate ax509certificate = certandkeygen.getSelfCertificate(x500name, validity*24*60* 60);
            certificate = (java.security.cert.Certificate)ax509certificate;
      } catch (Exception e) {
            System.err.println("Caught exception: " + e.toString());
      }
}
```

Konsumen dapat menyimpan informasi-informasi yang sudah dimasukkannya ke dalam media penyimpanan. Konsumen dapat menyimpan informasi-informasi tersebut dalam *hardisk* atau *smartcard* sebagai alternatif lain.

```
void btnHardisk_ActionPerformed(java.awt.event.ActionEvent event) {
   if (isEmpty() == true) return;
   try {
         …..
         byte[] data = strData.getBytes();
         FileOutputStream dataFos = new FileOutputStream(myhome + txtName.getText() + ".swt");
         dataFos.write(data);
         dataFos.flush();
         dataFos.close();
         // simpan kunci privat
         dumpKey(myhome + txtName.getText() + ".key");
         // simpan sertifikat
         dumpCert(myhome + txtName.getText() + ".cer");
         txtMsg.setText("Saving data to hardisk finished ...");
   } catch (Exception e) {
         System.err.println("Caught exception: " + e.toString());
   }
}
```

```
void btnSmartcard_ActionPerformed(java.awt.event.ActionEvent event) {
    if (isEmpty() == true) return;

    card.drive_Init ();
    char[] reset = card.card_On ();

    try {
        // penting : file harus dibuka lebih dahulu sebelum ditulis
        // nomor file=1, tipe=0 (ordinary)
        card.open_File((short)1, (short)0);
        // recordNumber, offset, datalength, data
        card.write_File((short)1, (short)0, (short)3, "SWT");
        card.write_File((short)2, (short)0, (short)20, txtName.getText());
        card.write_File((short)3, (short)0, (short)20, txtPayment.getSelectedItem());
        card.write_File((short)4, (short)0, (short)20, txtCard.getSelectedItem());
        card.write_File((short)5, (short)0, (short)20, txtNumber.getText());
        String myexpiry = getFixedString((String.valueOf(txtExpiry1.getSelectedIndex())).trim(),2) +
                getFixedString((txtExpiry2.getSelectedItem()).trim(),4);
        card.write_File((short)6, (short)0, (short)6, myexpiry);
        card.card_Off();
    } catch(Exception e) {System.out.println(e);}
}
```

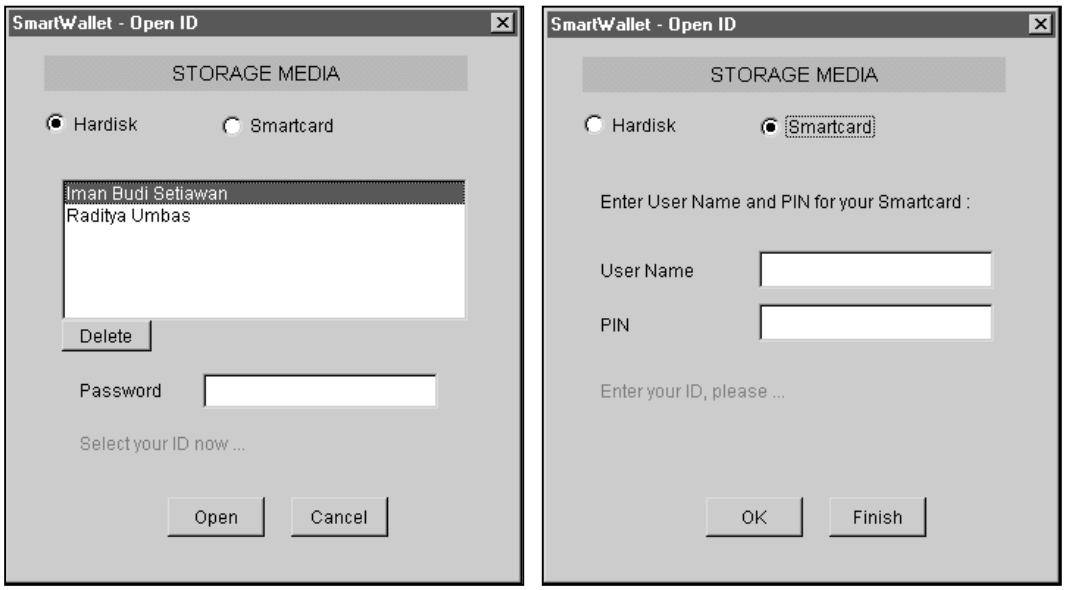


Gambar VI.5. Antar Muka Penambahan Identitas

6.5 MEMBUKA FILE IDENTITAS

Konsumen yang akan melakukan transaksi harus membuka informasi-informasi miliknya yang sebelumnya telah disimpan. Konsumen dapat memilih media penyimpanan yang digunakan tempat menyimpan informasi-informasi yang miliknya. Berikut ini diberikan fungsi readCert yang digunakan untuk membaca sertifikat konsumen yang disimpan dalam hardisk:

```
java.security.cert.Certificate readCert(String filename) {
    X509Certificate cert;
    try {
        InputStream in = new FileInputStream(filename);
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        cert = (X509Certificate)cf.generateCertificate(in);
        in.close();
    } catch (Exception e) {
        System.err.println("Caught exception: " + e.toString());
        return null;
    }
    return cert;
}
```



Gambar VI.6. Antar Muka untuk Membuka Identitas

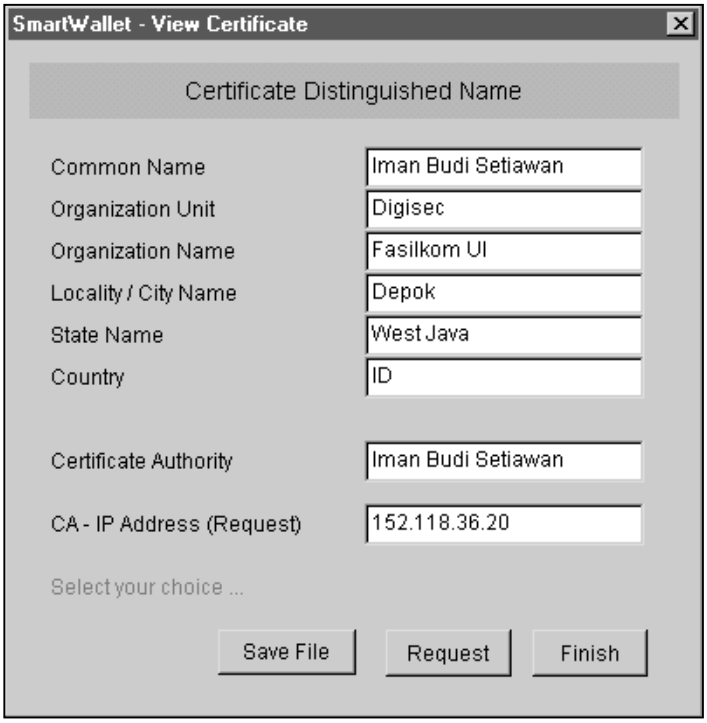
Berikut ini fungsi readPrivKey yang digunakan untuk membaca kunci privat konsumen yang disimpan dalam hardisk:

```
PrivateKey readPrivKey(String filename) {
    PrivateKey priv;
    try {
        InputStream in = new FileInputStream(filename);
```

```
            byte[] encKey = new byte[in.available()];
            in.read(encKey);
            in.close();
            KeyFactory factory = KeyFactory.getInstance("RSA", "ABA");
            PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(encKey);
            priv = factory.generatePrivate(keySpec);
        } catch (Exception e) {
            System.err.println("Caught exception: " + e.toString());
            return null;
        }
        return priv;
}
```

6.6 PERMINTAAN SERTIFIKAT DIGITAL KE CCA

Sertifikat konsumen yang dibuat saat penambahan identitas ditandatangani menggunakan kunci privat miliknya sendiri (*self signed*). Transaksi yang dilakukan dengan pedagang membutuhkan kepercayaan yang tinggi sehingga sertifikat tersebut harus ditandatangani oleh otoritas sertifikat yang berwenang. Bagian ini digunakan konsumen untuk melakukan proses permintaan sertifikat agar memperoleh tanda tangan dari otoritas sertifikat.



Gambar VI.7. Antar Muka untuk Permintaan Sertifikat

Berikut ini fungsi yang digunakan untuk melakukan permintaan tanda tangan sertifikat digital yang dijalankan setelah konsumen menekan tombol “REQUEST”:

```
void btnRequest_ActionPerformed(java.awt.event.ActionEvent event) {
        if ((txtIP.getText()).equals("")) {
                txtMsg.setText("IP Address can't be empty !");
                return;
        }
        x509certificate = CertReq.createRequest(x509certificate);
        txtMsg.setText("Certificate has been signed by CA !");
}
```

Berikut ini fungsi yang digunakan untuk menyimpan sertifikat yang sudah ditandatangani oleh otoritas sertifikat:

```
private void dumpCert(String filename) throws IOException, CertificateException {
        PrintStream printstream = new PrintStream(new FileOutputStream(filename));
        printstream.write(((java.security.cert.Certificate)x509certificate).getEncoded());
}
```

6.7 TRANSAKSI DENGAN PEDAGANG

Konsumen melakukan transaksi dengan menekan tombol “TRANSACTION”. Informasi-informasi dari konsumen dikirim ke proses transaksi antara konsumen dan pedagang dalam bentuk parameter HODInput dan sertifikat X.509. Proses transaksi yang secara langsung melibatkan konsumen dan pedagang diimplementasi oleh Dwinanda Prayudi [DAPI99]. Berikut ini proses untuk mengirimkan HODInput dan sertifikat X.509 ke proses transaksi tersebut:

```
try {
        CurrencyAmount cur = new CurrencyAmount(new BigInt(120),new
        BigInt(Integer.parseInt(total)),new BigInt(1));
        byte[] odSalt = {1,4,33,2,1,67,88,23,12,8,1,4,33,2,1,67,88,23,12,8};

        Calendar cal = Calendar.getInstance();
        cal.set(tahun,bulan,1);
        Recurring rec = new Recurring(new BigInt(28),cal.getTime());

        InstallRecurData install = new InstallRecurData(rec);

        HODInput hod = new HODInput(od,cur,odSalt,install);
        boolean bool = PurchaseReq.processToMerchant(hod, (X509Certificate)certificate);

} catch(Exception e) {System.out.println(e);}
```

6.8 PEMBUATAN MODUL POLICY

Seperti yang telah dijelaskan sebelumnya bahwa modul *policy* digunakan agar *browser* menerima *applet* yang dapat dipercaya untuk mengakses sumber daya di komputer lokal. Hak-hak yang diberikan kepada sebuah *applet* tergantung implementasi dari modul *policy* tersebut. Berikut ini adalah langkah-langkah yang dilakukan untuk menjalankan aplikasi SmartWallet:

1. *Source code* dari aplikasi SmartWallet harus di-compile terlebih dahulu agar dapat dijalankan menggunakan *interpreter* Java.
   Perintah: Javac *.java
2. *Class file* yang diperoleh dari langkah pertama dikompres ke dalam format Jar.
   Perintah: Jar cvf Wallet.jar *.class
3. Membuat sertifikat digital untuk menandatangani file Jar yang telah dibuat sebelumnya.
   Perintah: keytool –alias iman
   (Setelah itu akan diminta memasukkan data-data yang diperlukan untuk membuat sertifikat / *Certificate Distinguished Name*).
4. Melakukan tanda tangan terhadap file Jar tersebut menggunakan JarSigner.
   Perintah: jarsigner Wallet.jar iman
5. Membuat sebuah *policy file* yang digunakan untuk memberikan hak-hak tertentu pada applet. *Policy file* tersebut diletakkan di ${user.home}\.java.policy. Berikut ini isi dari *policy file* yang digunakan untuk memperbolehkan aplikasi SmartWallet agar dapat mengakses komputer *client* :

```
grant SignedBy “Iman Budi Setiawan” codeBase “152.118.36.20” {
    permission java.lang.RuntimePermission “loadLibrary”;
    permission java.security.SystemPermission “insertProvider”;
    permission java.util.PropertyPermission “user.home”,”read”;
    permission java.io.FilePermission “user.home”,”read,write,delete”;
    permission java.net.SocketPermission “152.118.36.20:7000”,“connect,accept”;
};
```

NO	JENIS	FITUR SMARTWALLET
1.	Teknologi	Smartwallet menggunakan teknologi Java Applet sehingga mempunyai kelebihan-kelebihan sebagai berikut: - Aplikasi langsung di-*download* dari *website* pedagang sehingga mudah mengaturnya. - Konsumen tidak perlu melakukan instalasi dan *update* untuk versi yang baru. - Komputer konsumen tidak perlu dipenuhi *wallet-wallet* yang berasal dari berbagai vendor. - Pengguna lain tidak dapat mengubah *wallet* yang digunakan untuk transaksi karena aplikasi tersebut memang tidak diletakkan di komputer konsumen.
2.	Media Penyimpanan	- Penggunaan media penyimpanan *hardisk* sebagaimana *wallet-wallet* lainnya, - Penyimpanan dengan *smartcard* mempunyai nilai tambah sebagai media penyimpanan alternatif yang lebih aman.
3.	Protokol	Protokol SET merupakan protokol yang tepat untuk melakukan transaksi (lihat tabel III.1).
4.	Jenis Pembayaran	SmartWallet memperbolehkan pembayaran menggunakan kartu kredit dan kartu debit.
5.	*Browser*	SmartWallet dapat dijalankan dengan baik di setiap komputer yang mempunyai Java *enabled browser* & JRE 1.2.2. Keduanya dapat diperoleh secara gratis di Internet
6.	Interaksi aplikasi dengan konsumen.	Konsumen dapat melakukan transaksi secara interaktif dan dapat mengatur informasi-informasi yang akan digunakan untuk transaksi.
7.	Sertifikat digital	SmartWallet dapat membuat sertifikat digital yang memenuhi standar X.509 dan dapat digunakan pada *browser* Internet Explorer dan Netscape Navigator.
8.	Keamanan	Keamanan lebih terjamin karena menggunakan *permission* untuk mengatur hak-hak yang diberikan pada aplikasi SmartWallet yang dijalankan.

Tabel VI-1 Fitur-fitur SmartWallet

NO	JENIS	KELEMAHAN SMARTWALLET
1.	Perangkat lunak	Konsumen harus meng-*install* Java Plugins (JRE) 1.2.2 atau lebih agar dapat menggunakan aplikasi SmartWallet.
2.	Pengetahuan konsumen	Konsumen harus mengerti tentang *policy file* atau dapat menggunakan *policy tool* yang berfungsi untuk mengatur *permission* pada aplikasi SmartWallet. Sebagai solusinya konsumen dapat menggunakan *policy file* yang terdapat pada bagian VI.8.
3.	Hubungan antar komputer	SmartWallet menggunakan hubungan *socket*. Pada komputer-komputer yang menggunakan *proxy* hal ini tidak dimungkinkan karena nomor *port* yang diperbolehkan hanya *port-port* umum, misalnya *port* 80 (HTTP) dam *port* 21 (FTP).

Tabel VI-2 Kelemahan-kelemahan SmartWallet

BAB VII
UJI COBA SISTEM

Bab ini membahas mengenai uji coba yang dilakukan terhadap aplikasi SmartWallet. Pengujian ini dilakukan untuk melihat apakah aplikasi SmartWallet dapat berjalan sesuai dengan tujuan yang diharapkan.

7.1 LINGKUNGAN UJI COBA

Pengujian SmartWallet dilakukan dengan simulasi transaksi antara pedagang dan konsumen. Komputer pedagang memberikan pelayanan kepada konsumen melalui sebuah jaringan intranet.

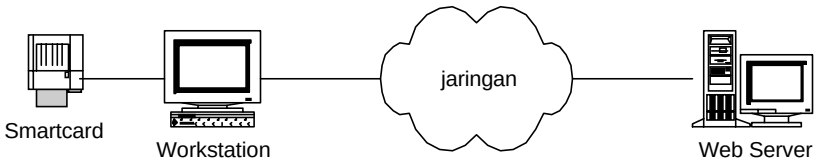
7.1.1 Spesifikasi komputer pedagang

1. Perangkat keras:
   - Komputer Pentium 166 MMX, SDRAM 64 MB, Hardisk 3.2 GB.
2. Perangkat lunak:
   - Sistem operasi Windows NT 4.0.
   - Microsoft Internet Information Server (IIS) 4.0.
   - Java Development Kit v1.2.2.
   - SmartWallet v1.0.

7.1.2 Spesifikasi komputer konsumen

1. Perangkat keras:
   - Komputer Pentium Celeron 333 MMX, SDRAM 160 MB, Hardisk 8.2 GB.
   - ASE Smartcard Reader, Microprocessor Card 2048 bytes.
2. Perangkat lunak:
   - Sistem operasi Windows 98.

- Java Development Kit v1.2.2.
- ASE Smartcard Driver.



Gambar VII.1. Pengujian SmartWallet

7.2 PELAKSANAAN UJI COBA

Berikut ini langkah-langkah yang dilakukan untuk pengujian SmartWallet:

- Komputer pedagang dan komputer konsumen dinyalakan, konsumen pedagang secara otomatis menjalankan *web server*.
- *Smartcard device* telah terpasang dengan baik di komputer konsumen dan *smartcard driver* telah di-*install* sebelumnya.
- Konsumen menjalankan sebuah *browser*, yaitu Internet Explorer atau Netscape Navigator.
- Konsumen membuka *website* pedagang dengan memasukkan URL pedagang pada *browser*.
- Konsumen dapat melakukan *browsing/shopping* di *website* pedagang.
- Konsumen dapat melakukan transaksi lebih lanjut (seperti yang dijelaskan pada bab sebelumnya).

7.3 HASIL UJI COBA

Berikut ini pengujian yang dilakukan terhadap aplikasi SmartWallet:

NO	PENGUJIAN	METODE PENGUJIAN	HASIL
1.	Menjalankan SmartWallet dari *browser*.	- SmartWallet dijalankan setelah konsumen menekan tombol “AGREE” pada halaman pemesanan.	√

2.	Pengiriman informasi pembelian dari *browser*. konsumen ke pedagang.	- Informasi transaksi yang dikirim konsumen ke pedagang sebelum membuka SmartWallet sesuai dengan pemesanan yang dilakukan melalui *browser*.	√
3.	Pengiriman informasi dari *browser* ke SmartWallet.	- Informasi pembelian yang ditampilkan pada SmartWallet sesuai dengan pemesanan yang dilakukan melalui *browser*.	√
4.	Pembuatan sertifikat digital yang sesuai standar X.509.	- Sertifikat digital dibuat dengan menekan tombol “GENERATE KEY” pada saat penambahan identitas. - Informasi *distinguished name* berdasarkan standar X.500 diperoleh dari masukan di SmartWallet. - Objek sertifikat dibuat berdasarkan standar X.509 pada spesifikasi SET. - *File* sertifikat yang telah dibuat dapat dibaca oleh *browser* Internet Explorer dan Netscape Navigator.	√
5.	Pembuatan kunci privat yang sesuai standar PKCS7.	- Kunci privat dibuat dengan menekan tombol “GENERATE KEY” pada saat penambahan identitas. - Kunci privat tersebut dibuat sesuai dengan standar PKCS7 pada spesifikasi SET.	√
6.	Pembuatan HODInput yang sesuai standar SET.	- Informasi transaksi dan informasi konsumen dikirim dalam bentuk objek HODInput. - HODInput merupakan sebuah kelas yang dibuat berdasarkan spesifikasi SET.	√
7.	Menulis informasi kartu pembayaran ke dalam *hardisk*.	- Buka antar muka untuk penambahan identitas. - Isi semua informasi yang dibutuhkan ke	√

		wallet. - Klik tombol “SAVE TO HARDISK”, maka informasi kartu pembayaran akan disimpan ke dalam *hardisk* dengan ekstensi “.swt”. - File ini dapat ditemukan di direktori ${user.home}.	
8.	Membaca informasi kartu pembayaran dari *hardisk*.	- Pengujian pertama dilakukan tanpa menutup aplikasi SmartWallet setelah penulisan ke *hardisk*. - Pengujian kedua dilakukan dengan menutup aplikasi SmartWallet terlebih dahulu untuk menghilangkan informasi yang mungkin masih ada di memori komputer. - Buka antar muka untuk membuka identitas. - Pilih media penyimpanan menggunakan *hardisk*. - Kemudian masukkan *password* yang diminta. - Informasi kartu kredit akan dibuka (*load*) ke *wallet*. Isinya sama dengan informasi yang telah dimasukkan sebelumnya.	√
9.	Menulis kunci privat ke dalam *hardisk*.	- Buka antar muka untuk penambahan identitas. - Kunci privat diperoleh setelah menekan tombol “GENERATE KEY”. - Klik tombol “SAVE TO HARDISK”, maka kunci privat akan disimpan ke dalam *hardisk* dengan ekstensi “.key”. - File ini dapat ditemukan di direktori ${user.home}.	√

10.	Membaca kunci privat dari *hardisk*.	- Pengujian pertama dilakukan tanpa menutup aplikasi SmartWallet setelah penulisan ke *hardisk*. - Pengujian kedua dilakukan dengan menutup aplikasi SmartWallet terlebih dahulu untuk menghilangkan informasi yang mungkin masih ada di memori komputer. - Buka antar muka untuk membuka identitas. - Pilih media penyimpanan menggunakan *hardisk*. - Kemudian masukkan *password* yang diminta. - Kunci privat akan dibuka (*load*) ke *wallet*. - Kunci privat tersebut harus sama dengan yang dimasukkan sebelumnya.	√
11.	Menulis sertifikat digital ke dalam *hardisk*.	- Buka antar muka untuk penambahan identitas. - Sertifikat diperoleh setelah menekan tombol “GENERATE KEY”. - Klik tombol “SAVE TO HARDISK”, maka sertifikat akan disimpan ke dalam *hardisk* dengan ekstensi “.cer”. - File ini dapat ditemukan di direktori ${user.home}.	√
12.	Membaca sertifikat digital dari *hardisk*.	- Pengujian pertama dilakukan tanpa menutup aplikasi SmartWallet setelah penulisan ke *hardisk*. - Pengujian kedua dilakukan dengan menutup aplikasi SmartWallet terlebih dahulu untuk menghilangkan informasi yang mungkin	√

		masih ada di memori komputer. - Buka antar muka untuk membuka identitas. - Pilih media penyimpanan menggunakan *hardisk*. - Kemudian masukkan *password* yang diminta. - Sertifikat akan dibuka (*load*) ke *wallet* dari file ".cer". - Sertifikat tersebut harus sama dengan yang dimasukkan sebelumnya.	
13.	Menulis informasi kartu pembayaran ke dalam *smartcard*.	- Buka antar muka untuk penambahan identitas. - Isi semua informasi yang dibutuhkan *wallet*. - Periksa bahwa *smartcard device* bekerja dengan baik dan *smartcard* telah dimasukkan dengan benar. - Klik tombol "SAVE TO SMARTCARD", maka informasi kartu pembayaran akan disimpan ke dalam *smartcard*.	√
14.	Membaca informasi kartu pembayaran dari dalam *smartcard*.	- Pengujian pertama dilakukan tanpa menutup aplikasi SmartWallet setelah penulisan ke *smartcard*. - Pengujian kedua dilakukan dengan menutup aplikasi SmartWallet terlebih dahulu untuk menghilangkan informasi yang mungkin masih ada di memori komputer. - Buka antar muka untuk membuka identitas. - Pilih media penyimpanan menggunakan *smartcard*. - Kemudian masukkan PIN yang diminta.	√

		- Jika PIN tidak dimasukkan maka SmartWallet akan meminta pemakai untuk memasukkan PIN. - Jika PIN yang dimasukkan benar maka informasi kartu pembayaran akan dibuka (*load*) ke *wallet*. - Jika PIN yang dimasukkan salah maka pemakai diminta mengulangi pemasukan PIN. - Informasi kartu kredit yang berhasil dibuka sama dengan informasi yang telah dimasukkan sebelumnya.	
15.	Menulis kunci privat ke dalam *smartcard*.	- Buka antar muka untuk penambahan identitas. - Sertifikat diperoleh setelah menekan tombol "GENERATE KEY". - Klik tombol "SAVE TO SMARTCARD", maka kunci privat akan disimpan ke dalam *smartcard.*	√
16.	Membaca kunci privat dari *smartcard*.	- Pengujian pertama dilakukan tanpa menutup aplikasi SmartWallet setelah penulisan ke *smartcard*. - Pengujian kedua dilakukan dengan menutup aplikasi SmartWallet terlebih dahulu untuk menghilangkan informasi yang mungkin masih ada di memori komputer. - Buka antar muka untuk membuka identitas. - Pilih media penyimpanan menggunakan *smartcard.* - Pastikan *smartcard* sudah terpasang dengan	√

		baik. - Kemudian masukkan PIN yang diminta. - Jika PIN tidak dimasukkan maka SmartWallet akan meminta pemakai untuk memasukkan PIN. - Jika PIN yang dimasukkan benar maka kunci privat akan dibuka (*load*) ke *wallet*. - Jika PIN yang dimasukkan salah maka pemakai diminta mengulangi pemasukan PIN. - Kunci privat tersebut harus sama dengan yang dimasukkan sebelumnya.	
17.	Menulis sertifikat ke dalam *smartcard*.	- Buka antar muka untuk penambahan identitas. - Sertifikat diperoleh setelah menekan tombol "GENERATE KEY". - Klik tombol "SAVE TO SMARTCARD", maka sertifikat akan disimpan ke dalam *smartcard.*	√
18.	Membaca sertifikat dari *smartcard*.	- Pengujian pertama dilakukan tanpa menutup aplikasi SmartWallet setelah penulisan ke *smartcard.* - Pengujian kedua dilakukan dengan menutup aplikasi SmartWallet terlebih dahulu untuk menghilangkan informasi yang mungkin masih ada di memori komputer. - Buka antar muka untuk membuka identitas. - Pilih media penyimpanan menggunakan *smartcard.*	√

		- Pastikan *smartcard* sudah terpasang dengan baik. - Kemudian masukkan PIN yang diminta. - Jika PIN tidak dimasukkan maka SmartWallet akan meminta pemakai untuk memasukkan PIN. - Jika PIN yang dimasukkan benar maka sertifikat akan dibuka (*load*) ke *wallet*. - Jika PIN yang dimasukkan salah maka pemakai diminta mengulangi pemasukan PIN. - Sertifikat yang diperoleh harus sama dengan yang dimasukkan sebelumnya.	

BAB VIII
PENUTUP

Bab ini membahas mengenai kesimpulan terhadap aplikasi SmartWallet yang telah diimplementasi serta saran dan pengembangannya lebih lanjut.

8.1 KESIMPULAN

Beberapa kesimpulan penting dari penelitian yang dilakukan adalah:

1. SmartWallet adalah Java Wallet yang berbasis *smartcard* dan protokol SET. Smartcard merupakan media alternatif untuk menyimpan informasi-informasi penting, selain aman juga lebih fleksibel dalam pemakainya. Protokol SET dirancang khusus untuk transaksi di Internet sehingga lebih menjamin keamanan transaksi tersebut berdasarkan pengujian yang dilakukan oleh Visa dan MasterCard.
2. Teknologi Java 1.2 dapat mengatur hak-hak yang dimiliki sebuah *applet* sehingga konsumen dapat membatasi kemampuan *applet* yang dijalankan, sedangkan ActiveX menggunakan metode *trusted* atau tidak sehingga setiap ActiveX yang dijalankan selalu mempunyai kekuasaan yang penuh terhadap sumber daya komputer.
3. Beberapa *applet* maupun ActiveX tertentu bersifat merusak komputer (*hostile applet & ActiveX*) sehingga perlu diwaspadai. Konsumen sebaiknya melakukan transaksi dengan pedagang yang dapat dipercaya atau menggunakan *wallet* yang ditandatangani oleh pihak-pihak yang dapat dipercaya.
4. Konsumen harus mengerti tentang *permission* dan *policy file* pada Java 1.2 agar dapat menjalankan SmartWallet sesuai dengan kebutuhannya. Pengetahuan ini dibutuhkan konsumen untuk membatasi hak pada *wallet* yang mempunyai akses ke dalam sistem komputer.
5. SmartWallet menggunakan *socket* untuk menghubungkan konsumen dengan pedagang. *Proxy-proxy* tertentu ada yang hanya memperbolehkan akses terhadap *port-port* umum seperti port 80 (HTTP) dan port 21 (FTP) akibatnya SmartWallet tidak dapat digunakan pada komputer-komputer yang memanfaatkan *proxy-proxy* tersebut.

8.2 SARAN DAN PENGEMBANGAN

Beberapa saran, kemungkinan pengembangan, dan penelitian lanjutan yang dapat dilakukan adalah:

1. SmartWallet dapat ditingkatkan dengan mendukung protokol SET versi 2.0 yang saat ini sedang dibuat spesifikasinya oleh Visa dan MasterCard. Beberapa perbaikan dan penambahan dilakukan pada protokol SET versi 2.0 terhadap versi sebelumnya.
2. Java Card dapat digunakan sebagai alternatif untuk media penyimpanan. Teknologi Java Virtual Machines (JVM) yang sudah terintegrasi di dalamnya serta proses kriptografi yang terjadi di dalam kartu memberikan keamanan yang lebih terjamin dibandingkan jenis *smartcard* yang lain saat ini.
3. Menggunakan teknologi enkripsi yang terbaru karena tidak ada teknologi enkripsi yang benar-benar aman, suatu saat nanti pasti ada cara untuk menjebolnya. Salah satu teknologi enkripsi simetri yang telah dijebol adalah DES 56-bit yang dapat dijebol dalam beberapa hari dengan *brute force attack* menggunakan program DESChall yang dibuat oleh Rocke Verser, Matt Curtin, dan Justin Dolske [MCJD98].
4. SmartWallet dapat dikembangkan dengan mendukung protokol pembayaran selain SET sehingga dapat digunakan untuk pembayaran *micropayment*, uang digital (*digital money*), dan kupon digital.
5. SmartWallet sebaiknya menggunakan teknologi Java terbaru yang merupakan perbaikan dari versi-versi sebelumnya sehingga dapat meningkatkan kinerja dan performanya di masa yang akan datang.
6. Sistem keamanan pada *browser* perlu lebih ditingkatkan, terutama adanya larangan ekspor AS yang hanya memperbolehkan *browser* menggunakan kunci simetri RC2 dan RC4 sebatas 40-bit di luar AS dapat menghambat kemajuan ilmu itu sendiri.

LAMPIRAN 1
FORMAT PERINTAH SMARTCARD API

Standarisasi ISO-7816 mengatur *format* perintah / instruksi yang digunakan untuk menjalankan fungsi-fungsi yang ada di dalam *smartcard*. Instruksi-instruksi ini dikirimkan secara lengkap dengan parameter-parameter yang dibutuhkannya ke *smartcard* menggunakan standar APDU (*Application Protocol Data Unit*) seperti yang telah dijelaskan pada landasan teori.

PERINTAH	INS	P1	P2	L	DATA
CREATE	E0	name	RA1	2	n, l
CREATE_PURSE	E8	name	type	6	RA1,RA2,RA3,n , l, SC1_2
OPEN	E4	name	type	0	/
WRITE_MAN	50	/	/	1 to 8	serial number to write
READ_MAN	54	/	/	1 to 8	serial number to read
WRITE	C4	rec no	*offset*	1 to 8	values to write
READ	C6	rec no	*offset*	1 to 8	values read
ERASE	C8	/	/	0	/
STATUS	C2	sc no	/	2	max_attempts, remaining
REACT	4C	crypt	/	8	value to enter (*key*)
WRITE_*KEY*	46	max	/	8	*master key* value
CHANGE_*KEY*	4A	/	/	8	new *master key* value
*KEY*_PRES	48	crypt	/	8	value to enter
WRITE_SC	40	max	sc no	8	*secret code* value
CHANGE_SC	44	/	sc no	8	new *secret code* value
SC_PRES	42	crypt	sc no	8	value to enter

INHIB_DES	AA	/	/	0	/
DES_ENCRYPT	A0	rec no	rec no	8	data to encrypt
DES_DECRYPT	A4	rec no	rec no	8	data to decrypt
MAC_GEN	A8	rec no	rec no	8	data to encrypt
RANDOM	AC	/	/	8	random number
READ_RESULT	A2	/	/	8	result read
CREDIT	32	/	/	3	MC
DEBIT	34	/	/	3	MD
CREDIT_CERTIF	36	/	/	14	MC(3), NT(3), Certif(8)
DEBIT_CERTIF	38	/	/	6	MD(3), NT(3)
READ_PURSE	30	/	/	6	SR(3), NT(3)

Tabel 9.1. Format perintah APDU

Keterangan :

INS : kode instruksi (perintah)
P1 : parameter 1
P2 : parameter 2
L : panjang data
name : nama *file*
rec no : nomor *record*
sc no : nomor *secret code*
crypt : *encryption indicator*, apakah pemasukkan key perlu dienkripsi
type : tipe *file*
n : jumlah *record*
l : panjang *record*
max : jumlah maksimal pemasukkan kunci yang salah
MC : jumlah yang akan dikredit
MD : jumlah yang akan didebit
NT : nomor transaksi
SR : jumlah yang tersisa.

Certif : *certificate*

RA1, RA2, RA3 : byte yang digunakan sebagai *rule access*

SC1_2 : nomor *secret code* pertama dan kedua

Return code :

9000 : OK

9002 : perintah tidak dieksekusi (dengan 4 alasan: kesalahan parameter, penolakan akses, dan sebagainya)

9006 : masalah penulisan di EEPROM

9008 : *disable* (jumlah maksimal pemasukan kode yang tidak benar dicapai).

6F00 : Perintah tidak dikenal (kode tidak ada, atau fungsi tertentu tidak boleh digunakan)

LAMPIRAN 2
PERMISSION DALAM JDK 1.2

Sebuah *permission* merepresentasikan hak untuk mengakses sumber daya lokal. Sebuah applet yang mempunyai permission diperbolehkan untuk mengakses ke sumber daya tertentu. *Permission* ini disimpan dalam *file policy* yang akan dibaca oleh *Java Runtime Environment* (JRE) saat menjalankan applet tersebut.

Format *file policy* :

```
grant SignedBy "[nama]" codeBase "[URL]" {
        permission [jenis-permission] "[lokasi-file]","[hak-permission]";
};
```

Berikut ini adalah jenis-jenis permission yang ada dalam JDK 1.2 :

Jenis Permission	Hak Permission
java.security.AllPermission	[tidak perlu]
java.awt.AWTPermission	accessClibboard, accessEventQueue, listenToAllAWTEvents, readDisplayPixels, showWindowWithoutWarningBanner
java.io.FilePermission	read, write, execute, delete
java.net.NetPermission	setDefaultAuthenticator, requestPasswordAuthentication, specifyStreamHandler
java.util.PropertyPermission	read, write
java.lang.reflectReflectPermission	suppressAccessChecks
java.lang.RuntimePermission	createClassLoader, getClassLoader, setContextClassLoader, setSecurityManager, createSecurityManager, exitVM, setFactory, setIO, modifyThread, stopThread, modifyThreadGroup, getProtectionDomain, readFileDescriptor, writeFileDescriptor, loadLibrary.{library name}, accessClassInPackage.{package name}, defineClassInPackage.{package name}, accessDeclaredMembers, queuePrintJob
SecurityPermission	getPolicy, setPolicy, getProperty.{key}, setProperty.{key},

	insertProvider.{provider name}, removeProvider.{provider name}, setSystemScope, setIdentityPublicKey, SetIdentityInfo, addIdentityCertificate, removeIdentityCertificate, printIdentity, clearProviderProperties.{provider name}, putProviderProperty.{provider name}, removeProviderProperty.{provider name}, getSignerPrivateKey, setSignerKeyPair
SerializablePermission	enableSubclassImplementation, enableSubstitution
SocketPermission	accept, listen, connect, resolve

LAMPIRAN 3
FORMAT PERINTAH JAR

Jar (*Java Archive*) Tool adalah sebuah aplikasi Java yang berfungsi untuk menggabungkan beberapa file menjadi sebuah *archive file* berdasarkan format kompresi ZIP dan ZLIB. Applet-applet dikompresi dalam file Jar untuk mempercepat waktu *download*. Disamping itu, file Jar juga dapat diberi tanda tangan oleh pemiliknya untuk proses autentikasi di *browser*.

Perintah : jar [options] [manifest] destination input-file

Berikut ini argumen-argumen yang digunakan pada Jar Tool :

c	Membuat sebuah *file* kosong
t	Melihat isi sebuah *file* Jar
x	Membuka *file* dari *file* Jar ke *file* aslinya
f	Memberikan informasi tentang nama *file*
v	Membuat *verbose output* di *stderr*
m	Menambah *file manifest* dalam *file Jar*
o	Menyimpan dalam *file* Jar tanpa kompresi
M	Tidak membuat *file manifest* dalam *file* Jar
u	Mengganti isi *file* Jar dengan mengubah *file manifest*
-C	Mengganti direktori selama perintah Jar

Keterangan :

- File manifest : file yang berisi informasi dari file-file yang ada dalam file Jar. File manifest menggunakan format penyimpanan ascii RFC822.
- Kompresi ZLIB : format kompresi yang dibuat oleh Jean-Loup Gailly (kompresi) dan Mark Adler (dekompresi). Kompresi ini merupakan sebuah kompresi yang *lossless data* dan dapat digunakan pada berbagai macam *platform*.

LAMPIRAN 4
FORMAT PERINTAH KEYTOOL

Keytool adalah aplikasi yang digunakan untuk manajemen kunci dan sertifikat. Aplikasi ini memperbolehkan penggunanya untuk membuat tanda tangan sendiri (*self signed*) menggunakan tanda tangan digital (*digital signature*). Keytool menyimpan kunci dan sertifikat dalam sebuah *file* yang disebut *keystore* yang diproteksi *password.*

Perintah : keytool [perintah]

Berikut ini perintah-perintah yang digunakan pada Keytool :

-help	Menampilkan perintah-perintah yang ada
-printcert	Menampilkan sebuah sertifikat digital di layar
-alias	Nama penyimpanan kunci dan sertifikat dalam *file keystore*
-keyalg	Algoritma kunci publik yang dipakai
-keysize	Ukuran kunci publik yang dipakai
-validity	Masa berlakunya kunci dan sertifikat
-keystore	Nama *file* penyimpanan dari hasil *keytool*
-file	*File* yang akan dibaca
-list	Melihat informasi-informasi dalam sebuah keystore
-storetype	Jenis media penyimpanan yang dipakai
-storepass	Password yang digunakan untuk memproteksi keystore
-genkey	Membuat kunci dan sertifikat baru
-selfcert	Membuat sertifikat yang diberi tanda tangan sendiri
-identitydb	Membaca identitas dari basis data identitas (.idb)
-import	Menyimpan sebuah file sertifikat ke dalam keystore
-export	Membuat sebuah file sertifikat (.cer)
-certreq	Membuat sebuah file permintaan sertifikat (.csr)
-keyclone	Menggandakan sebuah informasi dalam keystore
-delete	Menghapus sebuah informasi dalam keystore

LAMPIRAN 5
FORMAT PERINTAH JARSIGNER

JarSigner adalah sebuah aplikasi Java yang berfungsi untuk menandatangani file Jar dan melakukan verifikasi untuk menjamin keutuhan dari file Jar yang telah diberi tanda tangan. JarSigner akan menambahkan *file signature* (.SF) dan *file* blok *signature* (.DSA) ke dalam *file* Jar yang ditandatangani tersebut.

Perintah : jarsigner [options] jar-file alias

jarsigner –verify [options] jar-file

Berikut ini argumen-argumen yang digunakan pada JarSigner :

-keystore	Memberikan lokasi dari file keystore
-storetype	Jenis penyimpanan keystore
-storepass	Password yang digunakan untuk memproteksi keystore
-keypass	Password yang digunakan untuk memproteksi kunci dan sertifikat
-sigfile	Nama file signature (.SF) dan file blok signature (.DSA)
-signedjar	Nama file hasil dari JarSigner
-verify	Melakukan verifikasi terhadap file Jar
-certs	Informasi sertifikat dari orang yang menandatangani file Jar
-verbose	Menjalankan JarSigner dalam mode *verbose* (*debug*)
-internalsf	Menggunakan *file signature* (.SF) dalam *file* blok *signature* (.DSA)
-sectionsonly	Tidak menyertakan header (*hash file manifest*) dalam *file signature* (.SF)
-J	Melewatkan sebuah string ke Java interpreter

Keterangan :

- *File signature* : sebuah file (.SF) yang terletak dalam *file* Jar yang dibuat saat JarSigner menandatangani *file* Jar tersebut. *File signature* ini berisi nama *file*, algoritma *digest*, dan nilai *digest* dari *file manifest*.
- *File* blok *signature* : sebuah file (.DSA) yang berisi tanda tangan dari *file signature* (.SF) dan sertifikat yang digunakan untuk menandatanganinya.

DAFTAR ACUAN

[ARRI97] Arrianto Mukti Wibowo : *Studi Perbandingan Sistem-Sistem Perdagangan di Internet dan Desain Protokol Cek Bilyet Digital*; Skripsi, Fakultas Ilmu Komputer UI, Depok, 1997.

[ARCL97] Arthur L. Coleman : *Giving Currency to The Java Card API*; JavaWorld, February 1997.

[ARCL98] Arthur L. Coleman : *Java Commerce Technologies, An Introduction*; Javasoft, Sun Microsystem Inc, 1998.

[AFPA99] Arif Prihasanta : *Protokol SET antara Pedagang dan Gerbang Pembayaran*; Skripsi, Fakultas Ilmu Komputer UI, Depok, 1999.

[ASDK95] ASE, *ASE Development Kit Documentation*; Alladin Smartcard Development, 1995.

[CDEX97] CardEx 97 : *Reducing fraud in cards*; Motorola, 1997.

[DHMM96] Thomas Dawkins, Justin Higgins, Sean Mathias, Joel Millecan, Michael Rice, Paul Tso: *Unlocking Microsoft Internet Information Server, New Riders Publishing*, Indianapolis, 1996.

[DALE96] Andrew Dahl, Leslie Lesnick: *Internet Commerce*; New Riders Publishing, Indianapolis, 1996.

[DAPI99] Dwinanda Prayudi : *Protokol SET antara Konsumen dan Pedagang*; Skripsi, Fakultas Ilmu Komputer UI, Depok, 1999.

[ELTC 96] Taher Elgamal, Jeff Treuhaft, Frank Chen: *Securing Communications on the Intranet and Over the Internet*; Netscape Communications Corp., 1996.

[ETMG96] Eric Tall and Mark Ginsburg : *Late Night ActiveX*; Macmillan Computer Publishing, USA, 1996.

[HSFI99] Haris Fauzi *: Protokol SET antara Konsumen dan Cardholder Certificate Authority*; Skripsi, Fakultas Ilmu Komputer UI, Depok, 1999.

[IETF96] Internet Engineering Task Force : *Secure Socket Layer 3.0 Specification*; USA, 1996.

[ISAP98] Istofani Api Diany : *Java API pada Smartcard ME2000*; Skripsi, Fakultas Ilmu Komputer UI, Depok, 1998.

[ISEC95] ISO/IEC 7816, *Interindustry Commands for Interchange*, ISO/IEC, 1995.

[JDKD98] Javasoft : *Java Development Kit Documentation*; Sun Microsystem Inc, 1998.

[JLVB94] Jeffrey L Whitten, Lonnie D.Bentley, Victor M Barlow : *System Analysis & Design Methods 3rd ed*, Richard D Irwin Inc, USA, 1994.

[MCJD98] Matt Curtin, Justin Dolske : *A Brute Force Search of DES Keyspace*, Computer Based Learning Unit, University of Leeds, USA, 1998.

[MIMO97] Mike Morgan : *Developing for Netscape One*; Que Corporation, Indianapolis, USA, 1997.

[MRRK98] McCullagh, Rusell, and Raid, Kathleen, *Component 2000: For Smartcard Technology International 1998*, Motorola, 1998.

[PAMI98] Paterson, Mike, *Smartcard and Security*, Motorola, 1998.

[RDSI99] RSA Data Security, Inc : *Security Protocols Overview*; An RSA Data Security Brief, San Mateo, USA, 1999.

[SCHL94] Schlumberger : *ME2000 Smartcard Documentation*; Schlumberger Technology, 1994.

[SCHN96] Bruce Schneier: *Applied Cryptography, 2nd ed;* John Wiley & Sons, Inc. New York, 1996.

[SNMC97] Sun Microsystem, *Java Native Interface Specification*, Sun Press, 1997.

[STAR97] Thom Stark: *Encryption for a Small Planet*; *BYTE* 22/3 (1997).

[TOHO94] Sheila Tomkowiak, Peter Hofland: *A Computer In Your Wallet;* BYTE 21/6 (1996).

[WERN96] Warrant Ernst : *Presenting ActiveX-TM;* Sams Net Publishing, Indianapolis, USA, 1996.