

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

DOCUMENTOS E DINHEIRO ELETRÔNICO COM *SMART*
CARDS* UTILIZANDO TECNOLOGIA *JAVA CARD

CLEBER GIOVANNI SUAVI

BLUMENAU
2005

2005/2-06

CLEBER GIOVANNI SUAVI

DOCUMENTOS E DINHEIRO ELETRÔNICO COM *SMART*

CARDS UTILIZANDO TECNOLOGIA JAVA CARD

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Marcel Hugo - Orientador

**BLUMENAU
2005**

2005/2-06

**DOCUMENTOS E DINHEIRO ELETRÔNICO COM *SMART*
CARDS UTILIZANDO A TECNOLOGIA *JAVA CARD***

Por

CLEBER GIOVANNI SUAVI

Trabalho aprovado para obtenção dos créditos
na disciplina de Trabalho de Conclusão de
Curso II, pela banca examinadora formada
por:

Presidente _____
Prof. Marcel Hugo – Orientador, FURB

Membro _____
Prof. Jomi Fred Hübner – FURB

Membro _____
Prof. Paulo Fernando da Silva – FURB

RESUMO

Smart cards (cartões inteligentes) são dispositivos com formato e dimensões muito semelhantes a um cartão de crédito sendo que a principal diferença está no fato deste dispositivo possuir um *chip* com capacidade de processamento e armazenamento de informações. O presente material aborda o desenvolvimento de aplicações para *smart cards* objetivando a utilização de dinheiro eletrônico, o armazenamento de documentos de forma eletrônica e mecanismos de segurança utilizando chaves assimétricas e certificado digital. Os *applets*, programas Java que executam em *smart cards* foram desenvolvidos utilizando-se a tecnologia *Java Card* que é uma arquitetura que possibilita que programas escritos na linguagem Java sejam executados nestes pequenos dispositivos. Os testes foram feitos utilizando um *smart card* real juntamente com a leitora e gravadora de *smart cards*.

Palavras-chave: *Smart card*. *Java card*.

ABSTRACT

Smart cards are small devices with shape and dimensions very similar to a credit card but the big difference between them, is that smart cards have a microchip with process and storage information capacity. This work describes the development of applications for smart cards with the purpose of using electronic cash, electronic documents storage and security mechanisms using asymmetric keys and digital certificate. The applets, programs that run into smart cards, was developed using the Java card technology, an architecture that enables codes written in Java language to run into these little devices. Tests were made using a real smart card with a smart card reader device.

Key-words: Smart card. Java card.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de <i>smart card</i> e a disposição dos contatos no <i>chip</i>	19
Figura 2 – Estrutura de mensagens APDU entre <i>host</i> e <i>smart card</i>	25
Figura 3 - Estrutura do sistema de pagamento e compras	30
Figura 4 - Diagrama de casos de uso – ator <i>host</i> (agência bancária ou estabelecimento comercial)	35
Figura 5 - Diagrama de casos de uso – ator <i>host</i> (orgão emissor).....	36
Figura 6 – Diagrama de classes: <i>package senha</i>	37
Figura 7 – Diagrama de classes: <i>package dinheiroEletronico</i>	38
Figura 8 – Diagrama de classes: <i>package documentosEletronicos</i>	39
Figura 9 – Diagrama de classes: <i>package servicosSeguranca</i>	40
Figura 10 – Diagrama de seqüência: <i>package senha</i>	43
Figura 11 – Diagrama de seqüência: <i>package dinheiroEletronico</i>	44
Figura 12 - Diagrama de seqüência: <i>package documentosEletronicos</i>	48
Figura 13 - Diagrama de seqüência: <i>package servicosSeguranca</i>	50
Figura 14 – <i>Smart card</i> utilizado para testes (JCOP <i>tools smart card</i>).....	57
Figura 15 – Leitor e gravador de <i>smart cards</i> utilizado para testes	57
Figura 16 – Kit <i>smart card</i> + CAD utilizado para testes com JCOP <i>Shell</i>	70
Figura 17 – JCOP <i>Explorer</i>	71
Figura 18 – JCOP <i>Shell</i> : comando <i>card-info</i>	72
Figura 19 – JCOP <i>Shell</i> : instalação do <i>package senha</i>	73
Figura 20 – <i>Applet CGSSenhaApplet</i> : seleção e gravação da senha.....	74
Figura 21 – <i>Applet CGSSenhaApplet</i> : validação da senha incorreta e posterior bloqueio	75
Figura 22 – <i>Applet CGSSenhaApplet</i> : desbloqueio e validação da senha	75
Figura 23 – <i>Applet CGSDinheiroEletronicoApplet</i> : seleção do applet e validação da senha ..	76
Figura 24 – <i>Applet CGSDinheiroApplet</i> : saque e saldo	77
Figura 25 – <i>Applet CGSDinheiroApplet</i> : depósito, saque e saldo.....	78
Figura 26 – <i>Applet CGSDocumentosEletronicosApplet</i> : gravação das informações referentes ao CPF	79
Figura 27 – <i>Applet CGSDocumentosEletronicosApplet</i> : leitura do documento CPF e dos dados comuns entre os documentos	80

Figura 28 – <i>Applet CGSCertificadoDigitalApplet</i> : envio ao <i>host</i> da chave pública do usuário	81
Figura 29 – <i>Applet CGSCertificadoDigitalApplet</i> : comandos para envio do certificado ao <i>smart card</i>	82
Figura 30 – <i>Applet CGSCertificadoDigitalApplet</i> : respostas do <i>applet</i> à solicitação de envio do certificado digital armazenado	83

LISTA DE QUADROS

Quadro 1 – Exemplos de casos para comandos e respostas APDU	26
Quadro 2 – Comandos APDU do <i>applet CGSDinheiroEletronicoApplet</i>	51
Quadro 3 – Comandos APDU do <i>applet CGSDocumentosEletronicosApplet</i>	52
Quadro 4 – Respostas APDU do <i>applet CGSDocumentosEletronicosApplet</i>	53
Quadro 5 – Comandos APDU do <i>applet CGSSenhaApplet</i>	53
Quadro 6 – Comandos APDU do <i>applet CGSCertificadoDigitalApplet</i>	55
Quadro 7 – Respostas aos comandos APDU do <i>applet CGSCertificadoDigitalApplet</i>	55
Quadro 8 – Estrutura básica de um <i>applet Java card</i>	58
Quadro 9 – Seleção da instrução (INS) no método <i>process()</i>	60
Quadro 10 – Declaração do método <i>getShareableInterfaceObject</i>	61
Quadro 11 – Método <i>getSenhaInterface()</i> que busca objeto da <i>interface CGSSenhaApplet</i> ...	62
Quadro 12 – Chamada do método <i>getSenhaInterface()</i>	62
Quadro 13 – Início do processo para persistência de dados no <i>smart card</i>	64
Quadro 14 – Método <i>setCpf()</i> da classe <i>CGSDocumentosEletronicos</i>	65
Quadro 15 – Enviando dados ao <i>host</i>	66
Quadro 16 – Método <i>install()</i> do <i>applet CGSCertificadoDigitalApplet</i>	68
Quadro 17 – Construtor da classe <i>CGSCertificadoDigital</i>	68
Quadro 18 – Método <i>getChavePublica()</i> da classe <i>CGSCertificadoDigital</i>	69
Quadro 19 – Exemplo de script JCWDE.....	90
Quadro 20 – Exemplo de <i>script</i> APDUTool	92
Quadro 21 – Resultado do <i>script</i> submetido à ferramenta APDUTool.....	93

LISTA DE SIGLAS

3DES – *Triple Data Encryption Standard*

AES – *Advanced Encryption Standard*

AID – *Applet Identifier*

APDU – *Application Protocol Data Units*

API – *Application Programming Interface*

CAD – *Card Acceptance Device*

CAP – *Converted Applet*

CEF – *Caixa Econômica Federal*

CNH – *Carteira Nacional de Habilitação*

CNPJ – *Cadastro Nacional de Pessoa Jurídica*

CPF – *Cadastro de Pessoa Física*

CPU – *Central Processing Unit*

CRT – *Chinese Remainder Theorem*

EEPROM – *Erasable Electrically Programmable ROM*

GSM – *Global System for Mobile Communications*

IDE – *Integrated Development Environment*

ISO – *International Standard Organization*

J2ME – *Java 2 Micro Edition*

JCRE – *Java Card Runtime Environment*

JCVM – *Java Card Virtual Machine*

JCWDE – *Java Card Workstation Development Environment*

JVM – *Java Virtual Machine*

OCF – *Open Card Framework*

PDA – *Personal Digital Assistant*

PIX – *Proprietary Identifier Extension*

RAM – *Random Access Memory*

RG – *Registro Geral (Identidade)*

RID – *Resource Identifier*

ROM – *Read Only Memory*

RSA – *Rivest-Shamir-Adleman*

SDK – *Software Development Kit*

SGBD – *Sistema Gerenciador de Banco de Dados*

SIM – *Subscriber Identity Module*

USB – *Universal serial Bus*

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 OBJETIVOS DO TRABALHO	15
1.2 ESTRUTURA DO TRABALHO	15
2 FUNDAMENTAÇÃO TEÓRICA	17
2.1 SMART CARDS	17
2.2 TECNOLOGIA JAVA CARD	19
2.2.1 Subconjunto da linguagem Java: limitações	20
2.2.2 <i>Java card virtual machine</i> - JCVM	21
2.2.3 <i>Java card runtime enviroment</i> - JCRE	22
2.3 PROTOCOLO APDU – <i>APPLICATION PROTOCOL DATA UNITS</i>	23
2.4 JAVA CARD SDK – <i>SOFTWARE DEVELOPMENT KIT</i>	26
2.5 IBM JCOP TOOLS	27
2.6 MECANISMOS DE SEGURANÇA.....	28
2.7 TRABALHOS CORRELATOS	28
2.8 ESTADO DA ARTE	31
3 DESENVOLVIMENTO DO TRABALHO	32
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	32
3.1.1 Permitir o armazenamento e leitura de documentos eletrônicos (RF)	32
3.1.2 Permitir o controle de débitos e créditos com relação ao dinheiro eletrônico (RF).....	33
3.1.3 Permitir o acesso às informações mediante uso de senha pessoal (RF).....	33
3.1.4 Utilizar a tecnologia <i>Java card</i> (RNF - implementação)	33
3.1.5 Permitir o armazenamento de um par de chaves e certificado digital (RNF - segurança)	34
3.2 ESPECIFICAÇÃO	34
3.2.1 Diagramas de casos de uso.....	34
3.2.2 Diagramas de classes.....	36
3.2.2.1 Package senha	36
3.2.2.2 Package dinheiroEletronico	37
3.2.2.3 Package documentosEletronicos	38
3.2.2.4 Package servicosSeguranca	39
3.2.3 Diagramas de seqüência.....	41

3.2.3.1 Package senha	41
3.2.3.2 Package dinheiroEletronico	43
3.2.3.3 Package documentosEletronicos	46
3.2.3.4 Package servicosSeguranca	49
3.2.4 Comandos e respostas APDU dos <i>applets</i>	50
3.2.4.1 CGSDinheiroEletronicoApplet.....	51
3.2.4.2 CGSDocumentosEletronicosApplet	51
3.2.4.3 CGSSenhaApplet.....	53
3.2.4.4 CGSCertificadoDigitalApplet.....	53
3.2.4.5 Status Word específicos dos applets desenvolvidos	55
3.3 IMPLEMENTAÇÃO	55
3.3.1 <i>Software e Hardware</i> utilizados.....	56
3.3.2 Estrutura básica de um <i>applet</i>	58
3.3.3 Autenticação com senha.....	60
3.3.4 Persistência de dados e transações	63
3.3.5 Enviando dados ao <i>host</i>	66
3.3.6 Chaves assimétricas e certificado digital	67
3.3.7 Estudos de caso com <i>JCOP Shell</i>	70
3.3.7.1 Applet CGSSenhaApplet	73
3.3.7.2 Applet CGSDinheiroEletronicoApplet.....	76
3.3.7.3 Applet CGSDocumentosEletronicosApplet	78
3.3.7.4 Applet CGSCertificadoDigitalApplet.....	80
3.4 RESULTADOS E DISCUSSÃO	83
3.4.1 Dificuldades encontradas	84
4 CONCLUSÕES	86
4.1 EXTENSÕES	86
REFERÊNCIAS BIBLIOGRÁFICAS	88
APÊNDICE A – Script JCWDE (<i>senhaApplet.app</i>)	90
APÊNDICE B – Script APDUTool (<i>senhaApplet.src</i>)	91
APÊNDICE C – Resultado do script submetido ao APDUTool (<i>saidasenhaApplet.txt</i>) ...	93
APÊNDICE D – Status Words específicos dos applets desenvolvidos.....	94
D.1. Package documentosEletronicos.....	94
D.2. Package dinheiroEletronico	95
D.3. Package servicosSeguranca	96

D.4. Package senha	96
D.5. Status words de uso geral	96
APÊNDICE E – Descrição dos casos de uso - Host	97
E.1. Cadastrar senha pessoal	97
E.2. Efetua depósito	97
E.3. Realizar saque	98
E.4. Visualizar Saldo	98
E.5. Visualizar documentos eletrônicos	99
E.6. Visualizar certificado digital	99
E.7. Validar senha pessoal	100
APÊNDICE F – Descrição dos casos de uso – Host órgão emissor	101
F.1. Armazenar certificado digital	101
F.2. Armazenar chave pública de CAs confiáveis	101
F.3. Gerar par de chaves para usuário do smart card	102
F.4. Emitir documento CPF	102
F.5. Emitir documento RG	103
F.6. Emitir documento Título de Eleitor	103
F.7. Validar senha pessoal	104

1 INTRODUÇÃO

Atualmente, o cidadão deve portar inúmeros documentos. Cada um desses documentos possui uma ou algumas finalidades. Essa variedade de documentos não oferece muita praticidade ao cidadão. Pode-se citar, por exemplo, o Título de Eleitor, que é usado esporadicamente. Outros, como o Cadastro de Pessoa Física (CPF) e Registro Geral (RG), são usados com mais frequência e, geralmente, apresentados juntamente para identificação do cidadão. Entre outros documentos, pode-se mencionar ainda Carteira Nacional de Habilitação (CNH), Carteira de Reservista, Carteira de Estudante.

Para substituir todos esses documentos, os *smart cards* ou cartões inteligentes poderiam ser utilizados de forma a tornar muito mais prático e rápido a abertura de uma conta no banco ou a compra de algum produto no comércio, por exemplo. Mas como isso seria possível?

Muito parecido tanto em formato quanto tamanho aos cartões de crédito com tarja magnética, um *smart card* possui como grande diferencial um pequeno *microchip* com capacidade de processamento e armazenamento de dados. Assim, um *smart card* seria capaz de armazenar não só as informações constantes nos documentos pessoais de um cidadão, mas também, o valor de cédulas e moedas que se conhece hoje, de tal forma que seria possível extinguir as mesmas. Além disso, outras formas de identificação, como por exemplo, a assinatura digital, como lembra Hong e Chun (2001, p. 1370), poderiam ser adicionadas ao *smart card*.

Assim, imagina-se um cenário onde o cidadão, com seu *smart card*, teria a possibilidade de efetuar um saque em qualquer caixa eletrônico de qualquer banco onde possua uma conta bancária de forma que, o valor sacado, seria armazenado e contabilizado no próprio *smart card*. Por sua vez, o cidadão poderia se dirigir a uma loja, fazer suas compras,

abrir sua conta nesta loja e efetuar o pagamento. Tudo isso bastando ao cidadão apresentar seu *smart card*, que neste momento, estaria contendo todos seus documentos e seu dinheiro, de forma totalmente digital. Desta forma, conforme lembra Buse (1998, p. 01), não haveria mais as lembranças da procura desajeitada pela última pequena moeda no fundo do bolso.

Neste cenário, um aspecto muito importante que deve ser considerado, pelo fato de haver envolvimento com dinheiro e documentos pessoais, é a segurança das informações presentes no *smart card*. Autenticação, integridade e confidencialidade são os serviços de segurança que devem estar presentes para garantir, respectivamente, a identidade do indivíduo, garantir que as informações envolvidas estão corretas e não sofreram alterações e, por fim, que o acesso aconteça somente por pessoas autorizadas.

Tendo em vista o cenário descrito, este trabalho propõe o desenvolvimento de uma aplicação (para uso em um *smart card* as aplicações chamam-se *applet* ou *cardlet*) que possa ser instalada em um *smart card*, onde sejam possíveis a utilização de dinheiro eletrônico e documentos pessoais, respectivamente, em substituição ao papel moeda e documentos pessoais em papel, além de garantir a segurança das informações pertencentes ao usuário do cartão.

Para alcançar esse propósito, é feito uso da tecnologia chamada *Java card* que possibilita o desenvolvimento de *applets* para serem utilizados em *smart cards*. Para garantir os serviços de segurança anteriormente citados é empregado o uso de chaves assimétricas, ou seja, chaves privada e pública e o uso de certificado digital.

Smart card aliado com a tecnologia *Java card*, oferece uma série de vantagens e benefícios. Além de compacto e do poder computacional, podem ser citados ainda segurança, portabilidade e facilidade de uso. Buse (1998, p. iv) descreve que *smart cards* oferecem também flexibilidade, que possibilita o compartilhamento de produtos e serviços num mesmo cartão. Paludo (2003, p. 85) lembra que essa característica torna o *smart card* dinâmico, ou

seja, novos *applets* podem ser instalados e registrados mesmo depois que um cartão foi emitido, se adaptando às necessidades de mudança do usuário do cartão.

Tendo em vista todas as vantagens e a praticidade das tecnologias citadas e o fato dessas tecnologias não terem sido ainda muito exploradas no Brasil, esse trabalho se propõe a servir de exemplo para que futuros trabalhos venham a serem desenvolvidos utilizando essas tecnologias.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é o desenvolvimento de *applets*, onde serão possíveis a utilização de dinheiro eletrônico e documentos pessoais, que poderão ser instalados e utilizados em *smart cards* (cartões inteligentes).

Os objetivos específicos são:

- a) utilizar a tecnologia Java *card* para o desenvolvimento dos *applets*;
- b) possibilitar o armazenamento e leitura dos seguintes documentos: CPF, RG e Título de Eleitor;
- c) permitir débitos e créditos com relação ao dinheiro eletrônico;
- d) permitir o armazenamento de um par de chaves (pública e privada) e um certificado digital para garantir confidencialidade, autenticação e integridade no acesso às informações presentes no *smart card*.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta a fundamentação teórica sobre *smart card*, a tecnologia Java *card* e o protocolo APDU. Além desses assuntos são abordados ainda o Java *card Software*

Development Kit (SDK) e o *plugin IBM JCOP Tools* que foram os softwares utilizados para os testes da implementação. Por fim, o capítulo apresenta mecanismos de segurança, trabalhos correlatos e o estado da arte.

Todos os diagramas de casos de uso, classes e diagramas de seqüência, juntamente com a especificação do protocolo APDU para os *applets* e detalhes sobre o desenvolvimento do trabalho são apresentados no capítulo 3.

Concluindo, no capítulo 4 são apresentadas conclusões finais e sugestões para extensões do presente trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Com relação à fundamentação teórica serão apresentados assuntos que, apesar de serem distintos entre si num primeiro momento, são de grande importância para o entendimento geral do trabalho principalmente no que diz respeito à implementação do mesmo. Sendo assim, os assuntos abordados nesta seção são os seguintes:

- a) *smart cards*: é apresentado conceito, configurações e pontos de contato do chip e tipos de memória utilizados;
- b) tecnologia Java *card*: peculiaridades e limitações;
- c) protocolo *Application Protocol Data Units* (APDU): o protocolo utilizado para comunicação entre aplicações *host* e *smart cards*;
- d) Java *card* SDK: *kit* para desenvolvimento e testes de *applets*;
- e) IBM JCOP *tools*: *plugin* para a ambiente Eclipse que permite o desenvolvimento e testes de *applets*;
- f) mecanismos de segurança;
- g) trabalhos correlatos;
- h) estado da arte.

2.1 SMART CARDS

Conforme descrito na introdução deste trabalho, *smart cards* são cartões semelhantes a cartões com tarja magnética (cartões de crédito comum), sendo que a principal diferença está no fato de possuírem um *chip* com capacidade de processamento e armazenamento de dados.

Para padronização das características de um *smart card* pelos diversos fabricantes, foram definidas normas que estão descritas no padrão ISO 7816 como lembra Lisboa (2003).

Esse padrão define características físicas como material e dimensões do cartão, posição dos contatos do *chip*, sinais elétricos e protocolos de transmissão, comandos para intercâmbio entre empresas, entre algumas outras características.

Conforme descreve Medeiros (2004b), como configuração típica, o *chip* de um *smart card* pode possuir uma Unidade Central de Processamento (CPU) de 16 ou 32 *bits* que funciona a 3.0 ou 5.0 MHz. Sua *Erasable Electrically Programmable Read Only Memory* (EEPROM) pode chegar a 80Kb. Como item opcional em um *smart card*, pode haver um co-processador utilizado somente para funções de criptografia.

O *chip* de um *smart card* possui oito pontos de contato, conforme descrevem Chen (2000, p. 14-15) e Buse (1998, p. 14-15):

- a) contato Vcc (*supply voltage*) é o responsável pela recepção da fonte de alimentação que pode ser de três ou cinco volts;
- b) contato GND (*ground*) é utilizado como voltagem de referência possuindo geralmente o valor zero;
- c) contato RST (*reset*) é o contato utilizado para enviar um sinal de *reset* ao *chip*;
- d) contato Vpp (*programming voltage*) é o contato utilizado para enviar um sinal de alta voltagem necessária para a programação da EEPROM;
- e) contato CLK (*clock*) é o responsável pelo recebimento do sinal lógico usado como referência para o sincronismo da comunicação serial. *Smart cards* não possuem fonte de *clock* interno, logo, este sinal deve ser recebido de uma fonte externa, neste caso um *Card Acceptance Device* (CAD), ou seja, o dispositivo onde o cartão é inserido para leitura e/ou gravação dos dados;
- f) contato I/O tem a função de transmitir dados entre o *smart card* e o CAD em modo *half duplex*, ou seja, a transmissão ocorre somente em uma direção em um determinado momento;

g) contatos RFU são, na verdade, dois contatos que foram definidos para uso futuro.

Na Figura 1, à direita, pode ser verificada a disposição dos contatos no *chip* e à esquerda, pode ser visualizado o local onde o *chip* está presente no *smart card* (pode-se verificar um pequeno quadrado escuro com cantos convexos, abaixo do nome *Kredyt Bank*).



Fonte: adaptado de Medeiros (2004b).

Figura 1 – Exemplo de *smart card* e a disposição dos contatos no *chip*

Com relação aos tipos de memória, comenta Chen (2000, p. 16) que um *smart card*, geralmente, possui três tipos que seriam:

- a) memória *Read Only Memory* (ROM) que é aquela utilizada para manter *applets* desenvolvidos pelo fabricante;
- b) memória EEPROM, assim como a ROM, pode persistir dados e *applets*. A diferença está no fato de que, neste tipo de memória, pode-se alterar o conteúdo durante o uso normal do cartão, assim como *applets* desenvolvidos podem ser instalados e registrados;
- c) memória *Random Access Memory* (RAM) é utilizada como memória temporária para modificar e manter dados, não sendo capaz de persistir os mesmos.

2.2 TECNOLOGIA JAVA CARD

Conforme explica Ortiz (2003a), *Java card* é uma tecnologia que adapta a plataforma Java para ser utilizada em *smart cards* e outros dispositivos, onde memória e processamento

são muito mais limitados que nos dispositivos que utilizam a tecnologia Java 2 *Micro Edition* (J2ME) como, por exemplo, celulares e *Personal Digital Assistents* (PDA). Como utiliza Java como linguagem, a tecnologia possibilita à legião de programadores Java a usar suas habilidades com a linguagem para começar a desenvolver *applets* sem grande conhecimento da tecnologia.

Como citado anteriormente, aplicativos que são interpretados e executados no *smart card* são denominados *applets*. Conforme explica Paludo (2003, p. 84), um *applet* é escrito em Java e compilado em um compilador Java comum.

Quando usado comercialmente, de acordo com Chen (2000, p.168), um *applet* deve possuir um identificador único chamado *Applet Identifier* (AID), que deve possuir entre 5 e 16 *bytes* sendo formados pelo *Resource Identifier* (RID), com tamanho 5 *bytes* e o *Proprietary Identifier Extension* (PIX) com tamanho 11 *bytes*. O RID identifica a organização e deve ser obtido através da *International Standards Organization* (ISO) enquanto que o PIX deve ser atribuído a cada *package* de acordo com gerenciamento que deve ser efetuado pelo proprietário do RID.

2.2.1 Subconjunto da linguagem Java: limitações

Por se tratar de um subconjunto da linguagem Java, conforme descrevem SUN MICROSYSTEMS (2003b, p. 7-10) e Alonso e Medeiros (2005, p.56), a tecnologia Java *card* possui algumas limitações, sendo citadas a seguir, as principais:

- a) não permite o carregamento dinâmico de classes, ou seja, um *applet* executando em um *smart card* pode referenciar somente classes que já estejam presentes no *smart card*;
- b) não é possível a utilização de *threads* (processos concorrentes) e *object cloning*

- (clonagem de objetos), assim como qualquer palavra reservada relacionada;
- c) não há suporte aos tipos *char*, *double*, *float*, *long* e *arrays* multi-dimensionais;
 - d) algumas classes como *String*, *Boolean* e *Integer* também não são suportadas, assim como a classe *System* do pacote *java.lang*;
 - e) palavras reservadas relativas a determinados recursos tais como *synchronized* e *volatile*, que dizem respeito a *threads* (processos concorrentes); *strictfp* e *native*, que dizem respeito à dependência de plataforma; e *transient* que diz respeito a serialização de objetos;
 - f) não há o mecanismo *garbage collection* na tecnologia Java card, mecanismo este que seria responsável por eliminar da memória objetos que não possuem referências para eles mesmos.

2.2.2 Java card virtual machine - JCVM

A tecnologia Java card possui sua máquina virtual implementada em duas partes. A primeira parte da JCVM é denominada de conversor, parte esta responsável por efetuar o pré-processamento das classes desenvolvidas pelo programador em um computador pessoal. Além de otimizar o *bytecode* a ser gerado, no pré-processamento é verificado se não houve a utilização de algum comando não pertencente ao subconjunto da linguagem Java utilizada na tecnologia Java card. O resultado do pré-processamento é um arquivo chamado *Converted Applet* (CAP).

Uma vez gerado o arquivo CAP pelo conversor é utilizado um dispositivo chamado CAD que diz respeito ao leitor/gravador onde o *smart card* é inserido e, então, a transmissão do arquivo CAP do computador pessoal para o *smart card* é efetuada. A partir deste momento o *applet* pode ser instalado e registrado, função esta executada pela segunda parte da JCVM,

que também é responsável pela execução das instruções em *bytecode* dos *applets* presentes no *smart card*.

Segundo Alonso e Medeiros (2005, p. 56) uma aplicação que utiliza a tecnologia Java *card* é considerada uma aplicação distribuída já que parte do processamento ocorre no *host*, e a outra no *smart card*.

Diferente de uma Java *Virtual Machine* (JVM) que executa como um serviço do sistema operacional em um computador pessoal, Chen (2000, p. 37) lembra que a JCVM executa no *smart card* somente durante sessões CAD, ou seja, o período a partir do momento em que o cartão é inserido em um CAD até o momento em que é removido do mesmo. Sempre que o *smart card* é inserido em um CAD o estado da JCVM é reiniciado, permanecendo em um *loop* e aguardando por comandos do protocolo APDU, que será visto adiante.

2.2.3 Java *card runtime environment* - JCRE

A JCRE consiste de todos os componentes Java *card* que executam no *smart card*. Além de efetuar a inicialização da JCVM e o gerenciamento de recursos entre sessões CAD, também é responsável por algumas funções realizadas em tempo de execução dos *applets*, funções estas presentes apenas na tecnologia Java *card*:

- a) manter objetos persistentes e transientes: objetos que mantêm dados persistentes através de sessões CAD são alocados na memória persistente (EEPROM), enquanto que objetos transientes, que não tem seu estado salvo durante sessões CAD retendo, portanto, dados temporários, são alocados na memória RAM do *smart card*;
- b) operações atômicas e transações: para manter a integridade dos dados gravados na

memória persistente do *smart card*, a tecnologia dispõe de transações, semelhante ao conceito de transações em um Sistema Gerenciador de Banco de Dados (SGBD), e atomicidade, onde cada processo de escrita em um campo de um objeto persistente é feito atomicamente;

- c) *applet firewall*: é um mecanismo que garante que *applets* não interfiram de alguma maneira (acesso, leitura ou escrita de dados) em outros *applets* presentes no *smart card*, exceto no caso a seguir;
- d) mecanismo de compartilhamento de *applets*: caso seja necessário, é possível um *applet* implementar uma *interface* compartilhada (*Shareable*) de modo a permitir o acesso de outros *applets* aos métodos e propriedades definidos na *interface*.

2.3 PROTOCOLO APDU – *APPLICATION PROTOCOL DATA UNITS*

APDU é o protocolo utilizado entre *host* e *smart card* para troca de mensagens de modo a efetuar o envio e recebimento de dados. O mesmo pode ser dividido em comando (enviado do *host* para o *smart card*) e resposta (enviado do *smart card* para o *host*).

Um comando APDU pode ser subdividido em cabeçalho e corpo. O cabeçalho é formado por quatro componentes, sendo cada componente formado por um *byte*, logo, o cabeçalho é formado por quatro *bytes*. Cada componente, sua função e código identificador é especificado a seguir:

- a) CLA: classe;
- b) INS: código da instrução a ser executada no *smart card*;
- c) P1 e P2: representam parâmetros que podem ser utilizados de maneira a alterar o processamento a ser efetuado em uma instrução.

Por sua vez, o corpo de um comando APDU é formado por três componentes

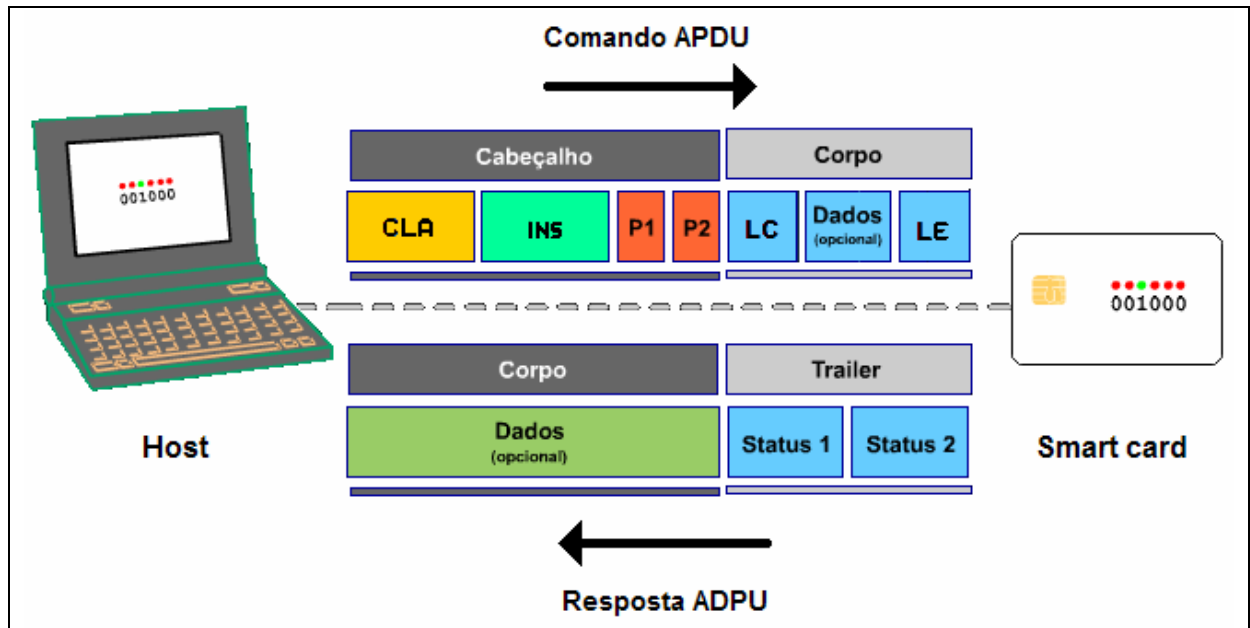
opcionais, conforme descritos a seguir:

- a) LC: formado por um *byte*, indica a quantidade de *bytes* no comando APDU, enviado ao *smart card* pelo *host*, ou seja, o tamanho do dado enviado;
- b) dados opcionais: *bytes* enviados do *host* para o *smart card* sendo seu tamanho informado em LC;
- c) LE: quantidade de *bytes* esperados pelo *host* como resposta a um comando enviado ao *smart card*.

Com relação à resposta APDU, a mesma pode ser subdividida em duas partes, conforme listado a seguir:

- a) dados opcionais: como resposta a um comando APDU, o *smart card* pode enviar *bytes* ao *host* que representam o resultado do processamento solicitado;
- b) SW: formado por dois bytes, representa o *status word* (ou *trailer*) do comando processado no *smart card*, ou seja, indica o resultado da execução de um comando APDU. Por padrão, todo comando executado com sucesso pelo *smart card* irá retornar um *status word* com valor hexadecimal 90 00.

Na Figura 2 é ilustrado a estrutura de comandos e respostas APDU entre *host* e *smart card*.



Fonte: adaptado de Alonso e Medeiros (2005)

Figura 2 – Estrutura de mensagens APDU entre *host* e *smart card*

Para melhor entendimento do protocolo APDU, o mesmo pode ser visualizado em quatro casos distintos, conforme exemplos apresentados no Quadro 1, sendo que em todos, o cabeçalho do comando APDU deve estar presente, ou seja, os valores CLA, INS, P1 e P2, devem obrigatoriamente, serem informados:

- caso 1: não são enviados dados, logo LC possui o valor 0x00, e nenhuma *byte* como resposta é esperado;
- caso 2: idem caso 1, exceto que LE é informado, ou seja, espera-se que o *smart card* envie ao *host* como resposta, uma quantidade de *bytes* igual a LE;
- caso 3: LC é informado indicando o número de *bytes* enviados ao *smart card*, ou seja, o tamanho do dados enviados;
- caso 4: idem caso 3, exceto que é esperado uma resposta do *smart card* com número de *bytes* igual a LE.

	<i>COMANDO APDU</i>							<i>RESPOSTA APDU</i>	
	Cabeçalho				Corpo				
	CLA	INS	P1	P2	LC	Dados	LE	Dados	Status Word
caso 1	0x07	0x06	0x05	0x04	0x00	-	-	-	0x90 0x00
caso 2	0x07	0x06	0x05	0x04	-	-	0x01	0x0A	0x90 0x00
caso 3	0x07	0x06	0x05	0x04	0x03	0x02 0x02 0x02	-	-	0x90 0x00
caso 4	0x07	0x06	0x05	0x04	0x03	0x02 0x02 0x02	0x01	0x0A	0x90 0x00

Quadro 1 – Exemplos de casos para comandos e respostas APDU

2.4 JAVA CARD SDK – *SOFTWARE DEVELOPMENT KIT*

Desenvolvido e disponibilizado gratuitamente pela *Sun Microsystems*, o *Java card SDK* consiste de documentação sobre as classes e *interfaces* utilizadas pela tecnologia *Java card* e de algumas ferramentas, onde *applets* podem ser desenvolvidos e testados antes de serem instalados e registrados em um *smart card*.

O processo de desenvolvimento de um *applet* com o *Java card SDK* consiste, basicamente, de três passos. O primeiro deles é a escrita e compilação do *applet*. No segundo passo, é necessário utilizar a ferramenta *Java Card Workstation Development Enviroment* (JCWDE) que permite testar e simular o *applet* diretamente, sem a necessidade de haver um *smart card* e um CAD. Porém, conforme explica Ortiz (2003b) e SUN MICROSYSTEMS (2003a, p. 31), este simulador não suporta a instalação e exclusão de *packages* e *applets*, persistência de dados, transações e *applet firewall* (e, conseqüentemente, compartilhamento entre *applets*).

Conforme explica Medeiros (2004a), para utilizar o JCWDE para efetuar os testes, é necessário que seja criado um arquivo texto contendo referências aos *applets* disponíveis para execução conforme demonstrado no apêndice A. Pode-se fazer uma analogia deste arquivo com o registro de *applets* mantidos pela JCRE no *smart card*.

Em seguida, no terceiro passo, deve ser criado um *script* com uma ou mais requisições utilizando o protocolo APDU que será submetido pela ferramenta APDUTool ao *applet*, de modo a testar suas funcionalidades e verificar os resultados da execução. Um exemplo desse *script* e o resultado de sua execução podem ser verificados respectivamente nos apêndices B e C.

Outras ferramentas no *kit* permitem que os arquivos com extensão *class* sejam convertidos em um único arquivo com extensão *cap* que será o arquivo instalado e registrado no *smart card*.

2.5 IBM JCOP TOOLS

JCOP *Tools* trata-se de um *plugin* desenvolvido pela empresa IBM (subsidiária da Suíça) para ser utilizado em conjunto com a *Integrated Development Environment* (IDE) Eclipse para o desenvolvimento e simulação de *applets Java card*.

Como diferencial em relação ao Java *card* SDK, o *plugin* dispõe de ferramenta de linha de comando chamada JCOP *Shell* onde é possível o envio de comandos APDU para o *smart card* real, sendo necessário neste caso, um CAD e um *smart card* reconhecido pelo *plugin*.

JCOP *Shell* permite ainda acesso ao *smart card*, *upload*, instalação e exclusão de *packages* e *applets* de maneira rápida, sem que o desenvolvedor necessite utilizar comandos APDU para tais tarefas, bastando alguns cliques em determinados botões na interface do *plugin*.

Com o JCOP *Tools* é possível verificar características do *applet* como código AID, tamanho que irá ocupar na memória EEPROM do *smart card*, versão, *bytecode* gerado, entre outras.

Além de toda a *Application Programming Interface* (API) da tecnologia *Java card* disponibilizada pela *Sun Microsystems*, o *plugin* disponibiliza ainda APIs adicionais, tanto para o desenvolvimento de *applets* quanto aplicações *host* que irão comunicar-se com o *smart card*.

2.6 MECANISMOS DE SEGURANÇA

De acordo com Chen (2000, p. 129), criptografia é a ciência da escrita secreta que objetiva manter mensagens seguras ou ocultas. Um dos serviços de segurança que a criptografia pode oferecer é a autenticação que tem como objetivo garantir a identidade de pessoas ou organizações.

Uma forma de prover autenticação entre *smart card* e *host*, é a utilização de certificados digitais, que possuem em seu conteúdo, além da identidade do usuário do *smart card*, sua chave pública (K_U). Este certificado, por sua vez, é assinado digitalmente com a chave privada ($K_{R_{auth}}$) da entidade certificadora, que garante a identidade do usuário.

Para ser autêntico para o *host*, o *smart card* deve armazenar um certificado digital assinado por uma autoridade certificadora confiável. Desta forma basta enviar o certificado digital para o *host* que deverá verificar se a assinatura no certificado é de uma entidade confiável. Em caso positivo, o *smart card* será autêntico. O mesmo processo deve ocorrer do *host* para o *smart card* que irá verificar o certificado digital do *host*.

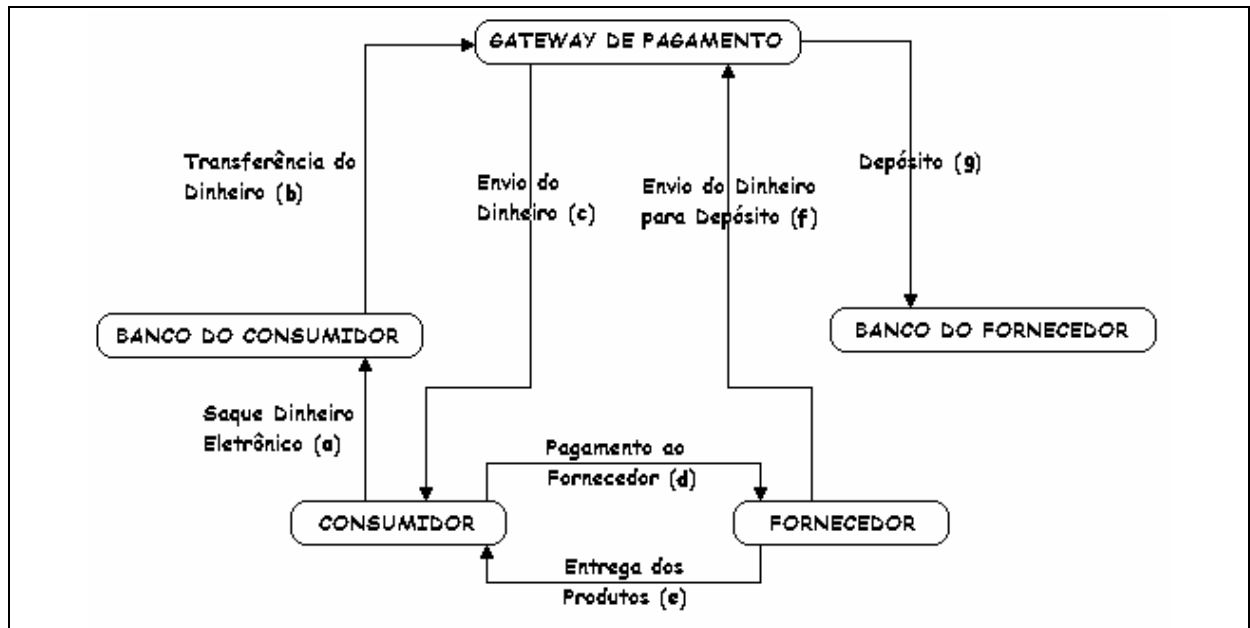
Para prover autenticação de um usuário com relação ao *smart card*, a forma mais comum é a utilização de uma senha que deverá ser informada pelo usuário em cada sessão CAD de forma a autorizar determinada operação entre *host* e *smart card*.

2.7 TRABALHOS CORRELATOS

Buse (1998, p. 59-63) descreve uma aplicação utilizando *smart cards* para ser empregada em cooperativas médicas independente do sistema de informação utilizado. Exames feitos nos laboratórios, compras de medicamentos em farmácias e serviços prestados em hospitais, consultórios e clínicas poderiam fazer parte do cenário proposto, que seria gerenciado pelo *smart card*, gerando os pagamentos aos prestadores dos serviços diretamente no banco da cooperativa, onde o cliente possui sua conta bancária.

Paludo (2003, p. 83-85) faz uma breve descrição de várias tecnologias para aplicações móveis, entre as quais está o *Java card*. Neste trabalho, são citados alguns exemplos de aplicações utilizando a tecnologia. Entre esses exemplos, está um projeto piloto da Caixa Econômica Federal (CEF) que tem por objetivo utilizar *smart cards* para registrar os dados pessoais de pelo menos cinquenta mil clientes. Além disso, a CEF poderá obter rendimentos alugando espaço disponível na EEPROM para parceiros que desejam inserir seus *applets* no mesmo *smart card*.

Para uma aplicação de *e-commerce*, Hong e Chun (2001, p. 1369-1372) apresentam uma estrutura de um sistema de pagamento e compras utilizando *smarts cards* com a tecnologia *Java card*. Nesta estrutura podem ser identificados o consumidor, o fornecedor e suas respectivas contas bancárias, sendo que ambas contas estão situadas em diferentes bancos, conforme mostrado na Figura 3, onde o objeto de estudo neste trabalho pode ser comparado ao objeto consumidor da figura.



Fonte: adaptado de Hong e Chun (2001, p. 1371)

Figura 3 - Estrutura do sistema de pagamento e compras

Na estrutura há também o chamado *gateway* de pagamento, responsável por conectar todos os bancos e companhias de cartão de crédito, fornecedores e consumidores. Esta estrutura, portanto, funciona da seguinte maneira:

- a) o consumidor, de posse de seu *smart card* contendo apenas as informações necessárias para *e-commerce*, requisita um saque de sua conta bancária;
- b) o dinheiro eletrônico referente a este saque é então enviado ao *gateway* de pagamento;
- c) por sua vez, o *gateway* de pagamento envia o dinheiro eletrônico ao *smart card* do consumidor onde o dinheiro é contabilizado e armazenado;
- d) deste modo, o consumidor efetua o pagamento de suas compras ao fornecedor;
- e) o fornecedor então, efetua a entrega ao consumidor dos produtos solicitados;
- f) agora, de posse do dinheiro eletrônico pago pelo consumidor, o fornecedor encaminha este dinheiro ao *gateway* de pagamento para que seja efetuado o depósito em sua conta bancária;

o *gateway* de pagamento, por sua vez, efetua o depósito na conta do fornecedor.

2.8 ESTADO DA ARTE

Atualmente os *smarts cards* são encontrados em grande escala em celulares com a tecnologia *Global System for Mobile Communications* (GSM). Porém, o formato dos mesmos não está de acordo com os padrões ISO7816, logo, são mais conhecidos por cartões *Subscriber Identity Module* (SIM), ou simplesmente *chip*, como é popularmente chamado pelos consumidores destes aparelhos.

Smart cards, conforme o padrão ISO7816, estão sendo utilizados no Brasil pelo governo federal na emissão de CPF, Cadastro Nacional de Pessoa Jurídica (CNPJ) e Identidade Digital a fim de facilitar o relacionamento entre contribuintes e a Secretaria da Receita Federal além de permitir o acesso a vários outros serviços oferecidos. Segundo a CERTISIGN (2005), esses documentos são utilizados para garantir a autenticidade dos remetentes de documentos e dados que trafegam na internet.

No ramo de vales refeição, *smart cards* são usados como cartões pré-pagos que armazenam um saldo referente ao valor disponível para pagamento de refeições pelo usuário do cartão.

3 DESENVOLVIMENTO DO TRABALHO

Para detalhar o processo de desenvolvimento do trabalho serão abordados e/ou apresentados os seguintes temas:

- a) análise e especificação dos requisitos do problema a ser trabalhado;
- b) especificação através da apresentação dos diagramas de casos de uso, classes e diagramas de seqüência;
- c) especificação dos comandos e respostas APDU dos *applets* desenvolvidos;
- d) *software* e *hardware* utilizados;
- e) estrutura básica de um *applet*;
- f) estudos de caso: implementação e testes em ambiente real;
- g) resultados obtidos e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos apresentados a seguir são divididos em requisitos funcionais (RF) e requisitos não-funcionais (NF).

3.1.1 Permitir o armazenamento e leitura de documentos eletrônicos (RF)

O *applet* deve permitir o recebimento e posterior armazenamento de informações referentes aos documentos CPF, RG e Título de Eleitor, assim como disponibilizar métodos de leitura pelo *host*.

As informações comuns ao usuário do *smart card*, como nome e data de nascimento, são armazenadas de maneira comum a todos os documentos de forma a evitar redundância de

tais informações em cada documento eletrônico.

3.1.2 Permitir o controle de débitos e créditos com relação ao dinheiro eletrônico (RF)

O *applet* deve permitir o armazenamento de dinheiro em formato eletrônico controlando débitos e créditos sobre um saldo disponível no *smart card*, sendo que o limite estabelecido está entre zero e 29.999,99 unidades de moeda. Este valor máximo foi escolhido arbitrariamente.

3.1.3 Permitir o acesso às informações mediante uso de senha pessoal (RF)

O acesso às informações contidas no *smart card*, com relação a dinheiro e documentos eletrônicos, ocorre somente mediante a validação de uma senha pessoal que deverá ser previamente cadastrada para posterior verificação.

Para tal, um *applet* deve gerenciar o armazenamento, a validação e o bloqueio, caso necessário, da senha. Além disso, deve oferecer uma *interface* de acesso compartilhada que permita, a outros *applets* instalados no *smart card*, a validação de uma senha recebida com a senha já armazenada.

3.1.4 Utilizar a tecnologia Java *card* (RNF - implementação)

Utilizar Java *card* de modo a verificar o funcionamento da tecnologia para *smart cards*, visto que se trata das poucas, senão a única tecnologia que permite o desenvolvimento de aplicações para tais dispositivos.

Além disso, o *hardware* disponível para testes dos *applets*, é compatível com a

tecnologia.

3.1.5 Permitir o armazenamento de um par de chaves e certificado digital (RNF - segurança)

O *applet* deve permitir o armazenamento de um par de chaves assimétricas e de um certificado digital, pois os mesmos possibilitam a implementação de serviços de segurança de modo a garantir a confidencialidade, autenticação e integridade no acesso às informações presentes no *smart card*.

Deve ser possível também, armazenar chaves públicas de entidades certificadoras confiáveis, no intuito de possibilitar a verificação da autenticidade do *host*.

3.2 ESPECIFICAÇÃO

A seguir, são ilustrados os diagramas de casos de uso, classes e diagramas de seqüência e, mais adiante, a especificação dos comandos e respostas APDU referentes aos *applets* desenvolvidos.

Para o desenvolvimento de tais diagramas foi utilizada a ferramenta *Enterprise Architect* versão 4.0 da empresa *Sparx Systems*.

3.2.1 Diagramas de casos de uso

Os diagramas de casos de uso a seguir partem do princípio que o ator envolvido será um computador (*host*) que irá submeter comandos APDU ao *smart card* que então efetuará o processamento necessário.

Para os casos de uso iniciados pela palavra visualizar, entende-se como sendo o envio

de informações ao *host* pelo *smart card* após processamento solicitado pelo *host*.

Na Figura 4 tem-se a imagem do ator *host*, que trata-se de um computador (agência bancária ou estabelecimento comercial, por exemplo) que estará efetuando o acesso ao *smart card* para consulta de informações ou requisições de processamento.

No diagrama observa-se que todas as operações relacionadas a dinheiro e documentos eletrônicos serão feitas somente mediante a validação da senha pessoal do usuário do *smart card*. Para registro da senha pela primeira vez e consulta ao certificado digital do usuário do cartão, a verificação da senha não é necessária.

As descrições dos casos de uso apresentados no diagrama da Figura 4 podem ser verificadas no apêndice E.

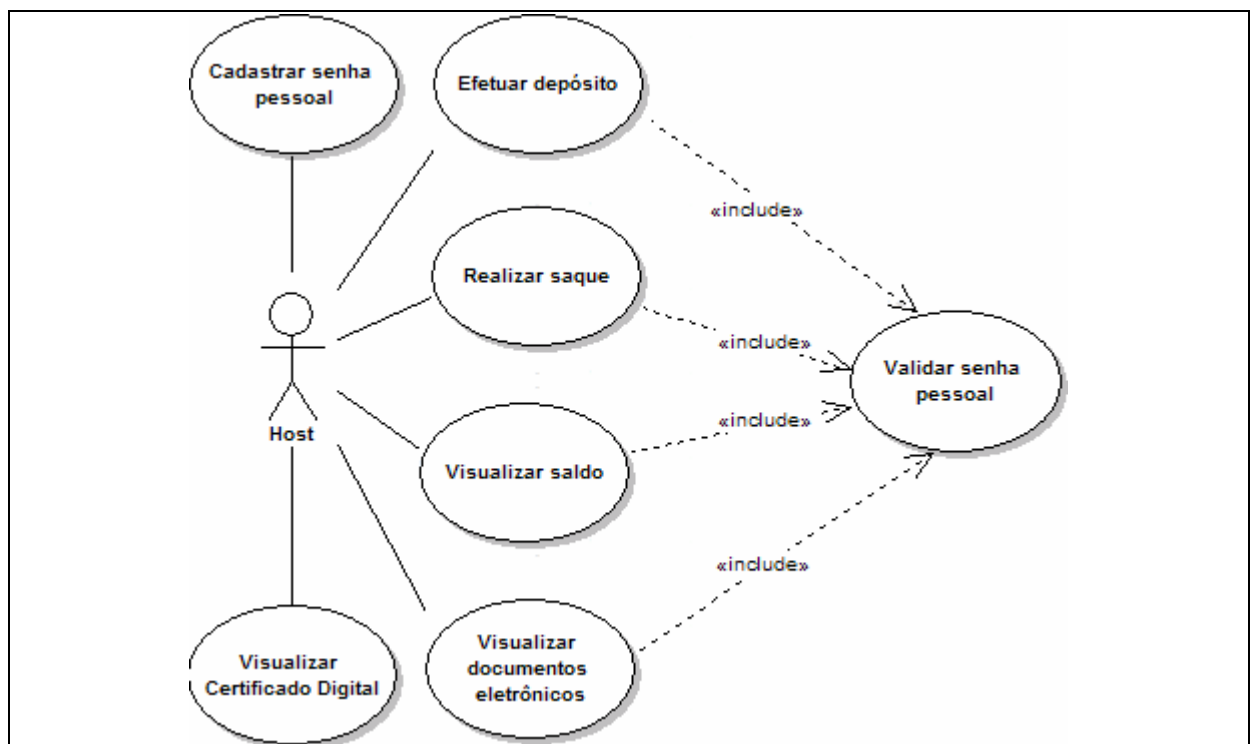


Figura 4 - Diagrama de casos de uso – ator *host* (agência bancária ou estabelecimento comercial)

Na Figura 5 tem-se novamente a imagem do ator *host*, porém, imagina-se o mesmo como sendo o órgão (entidade certificadora ou órgão do governo) responsável pela emissão de documentos e certificados digitais.

Para a emissão dos documentos do usuário do *smart card*, o mesmo deve ter sua senha

previamente cadastrada que deverá, portanto, ser validada.

Para a geração do par de chaves, o armazenamento de chaves públicas de entidades certificadoras confiáveis e o armazenamento do certificado digital do usuário do *smart card*, não é necessária a autenticação com senha pessoal.

As descrições dos casos de uso apresentados no diagrama da Figura 5 podem ser verificadas no apêndice F.

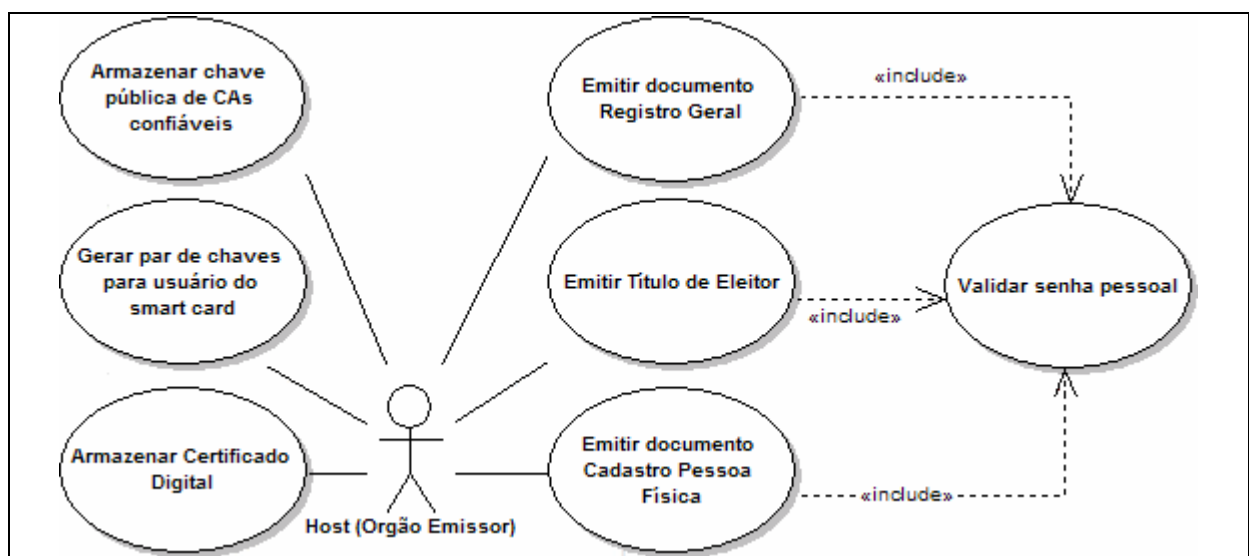


Figura 5 - Diagrama de casos de uso – ator *host* (orgão emissor)

3.2.2 Diagramas de classes

Os diagramas de classes foram organizados em *packages* (pacotes) de modo que, cada qual, represente um *applet* e suas classes auxiliares.

3.2.2.1 *Package senha*

O diagrama de classes visto na Figura 6 apresenta a classe *CGSSenhaApplet* que efetua o gerenciamento da senha pessoal do usuário do *smart card*. Esta classe implementa a

interface *CGSSenhaInterface* que possibilita outros *applets* registrados efetuarem a verificação da senha quando esta é informada.

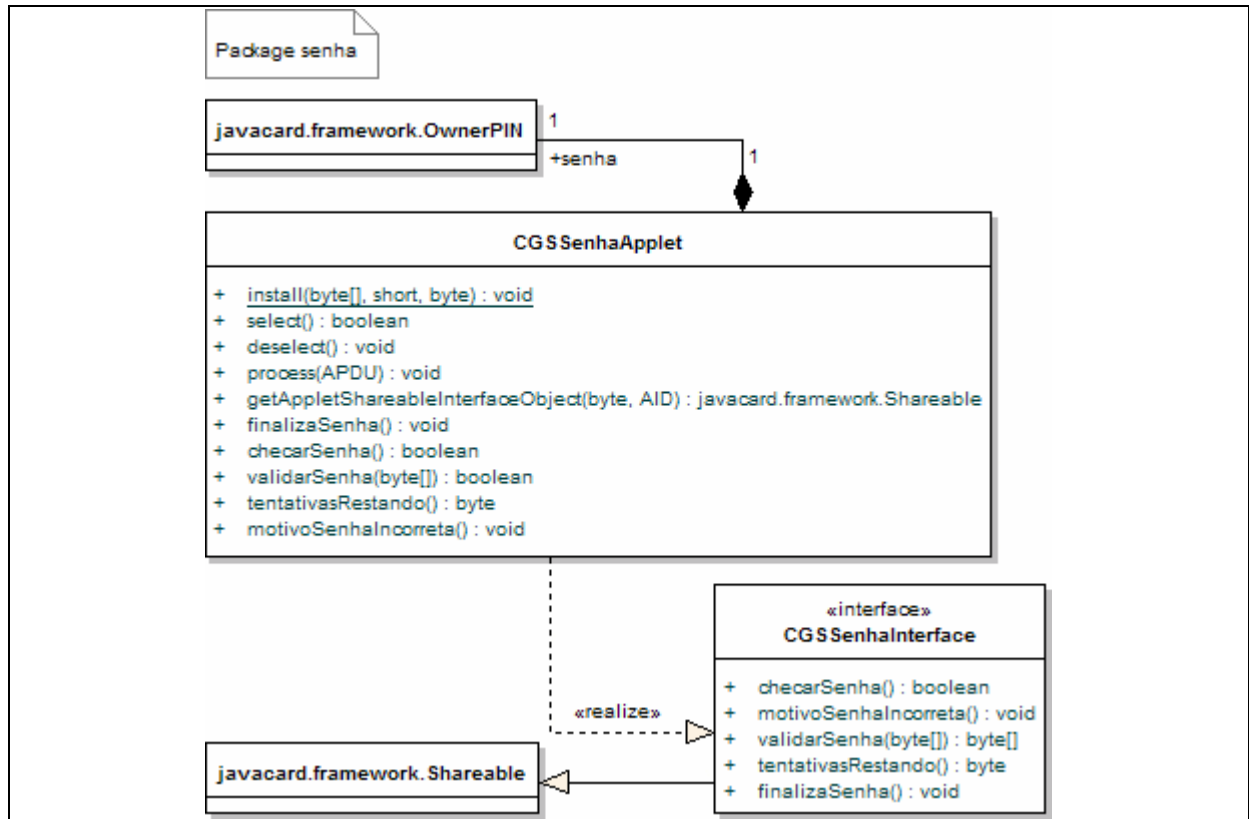


Figura 6 – Diagrama de classes: *package* senha

3.2.2.2 *Package dinheiroEletronico*

Na Figura 7 é apresentada a classe *CGSDinheiroEletronicoApplet*, classe esta que representa o *applet* responsável pelo dinheiro eletrônico. Uma vez instalada no cartão, o objeto desta classe possuirá uma instância da classe *CGSDinheiroEletrônico* onde o dinheiro eletrônico é efetivamente armazenado e as operações de soma e subtração sobre o saldo disponível ocorram. É também função desta classe retornar o saldo disponível no *smart card*.

Na classe *CGSDinheiroEletronicoApplet* pode ser observada uma referência à interface *CGSSenhaInterface* que se trata do objeto pela qual chamadas aos métodos da classe *CGSSenhaApplet* acontecem.

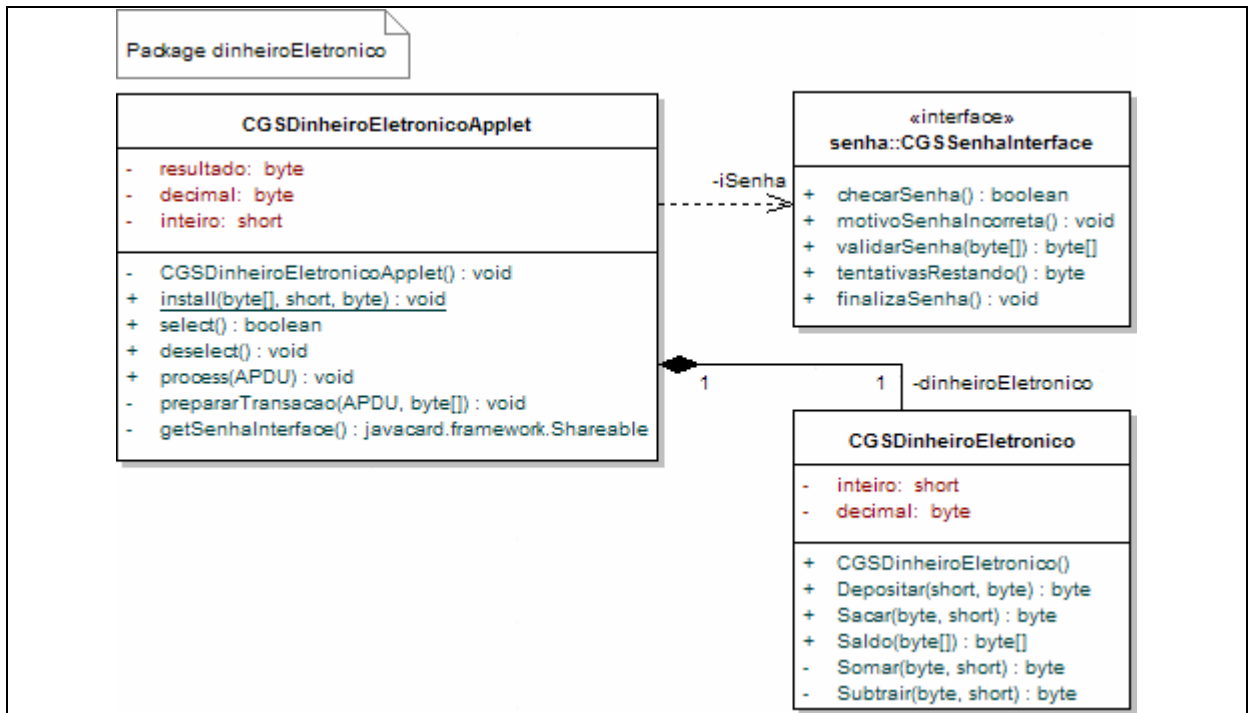


Figura 7 – Diagrama de classes: *package dinheiroEletronico*

3.2.2.3 *Package documentosEletronicos*

Na Figura 8 observa-se o conjunto de classes responsável pelo gerenciamento de documentos eletrônicos. A classe *CGSDocumentosEletronicosApplet*, a exemplo da classe *CGSDinheiroEletronicoApplet* possui referência à interface *CGSSenhaInterface* possibilitando as mesmas funções com relação a utilização da senha.

Agregada (através de uma composição) à classe *CGSDocumentosEletronicosApplet*, está a classe *CGSDocumentosEletronicos* que irá possuir as instâncias das classes responsáveis pelo gerenciamento de cada tipo de documento eletrônico. Essa estratégia é adotada para que haja somente um ponto de acesso às informações referentes aos documentos, que irão ocorrer somente a partir da classe *CGSDocumentosEletronicos*.

Nesta mesma classe ficam armazenadas as informações comuns entre todos os documentos como, por exemplo, nome e data de nascimento do usuário do *smart card*.

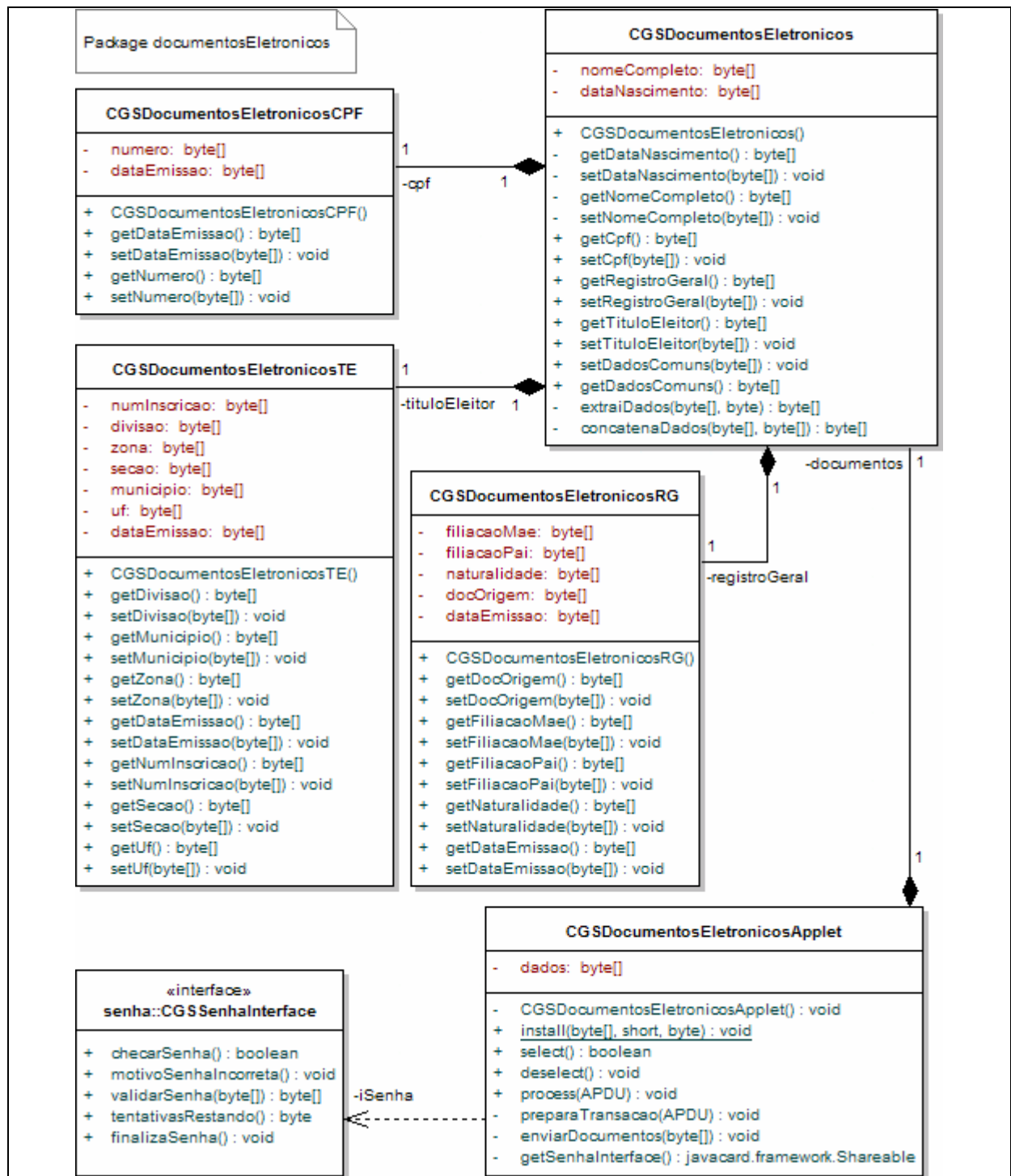


Figura 8 – Diagrama de classes: *package documentosEletronicos*

3.2.2.4 *Package servicosSeguranca*

Na Figura 9 é apresentada a estrutura de classes que gerenciam as chaves assimétricas e certificado digital do usuário e as chaves públicas das autoridades certificadoras confiáveis.

A classe *CGSCertificadoDigital*, que está agregada através de uma composição à classe *CGSCertificadoDigitalApplet*, mantém as chaves e o certificado digital do usuário e possui a função de instanciar as chaves, enviar a chave pública e, receber e enviar o certificado digital do usuário quando solicitado pelo *host*.

A classe *CGSChavePublicaCA*, por sua vez, é responsável por manter uma lista de chaves públicas de autoridades certificadoras confiáveis.

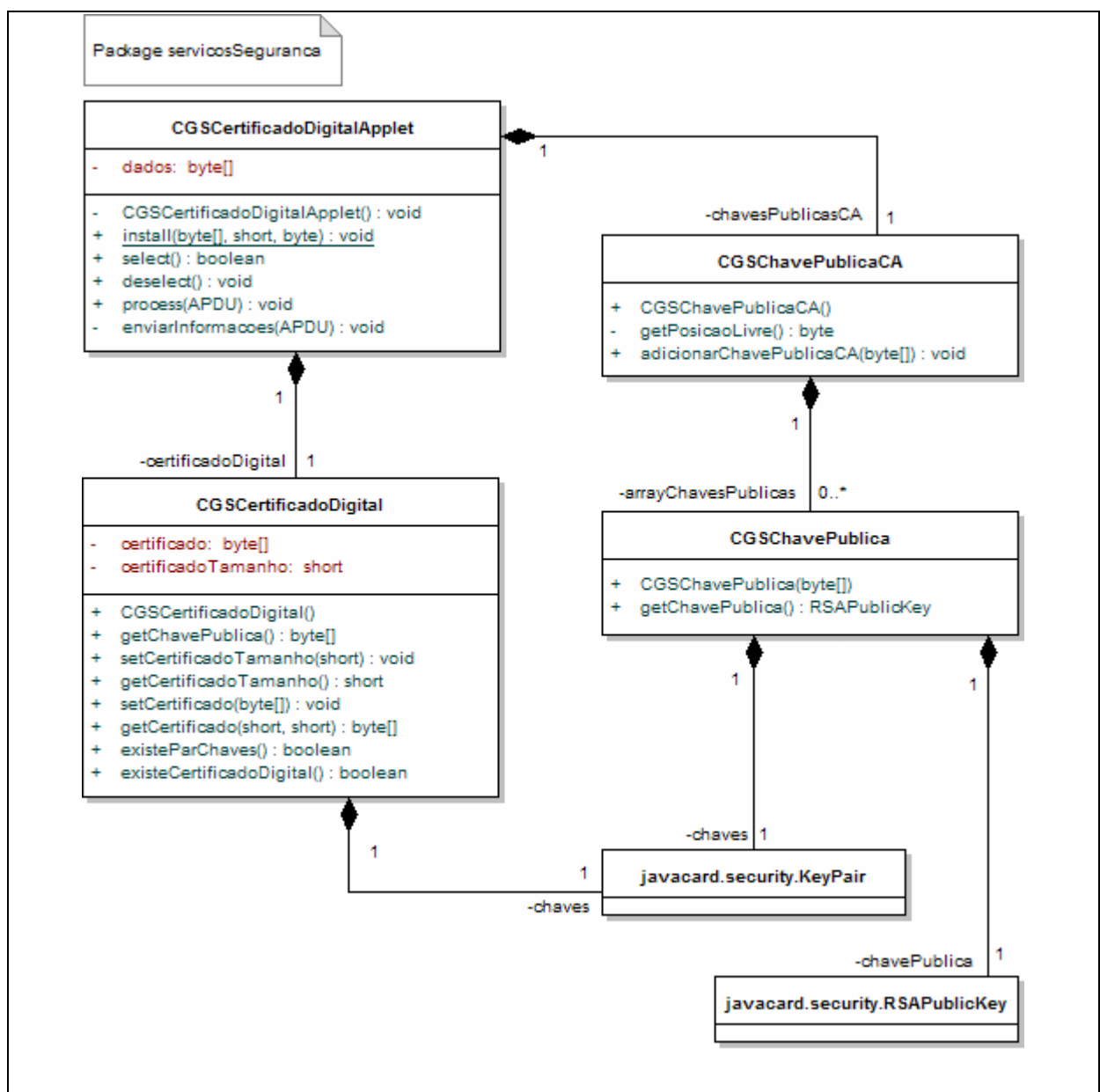


Figura 9 – Diagrama de classes: *package servicosSeguranca*

3.2.3 Diagramas de seqüência

Os diagramas de seqüência, a exemplo dos diagramas de classes, estão organizados em *packages*, sendo feito o uso de classes da API Java *card* em alguns casos, para melhor ilustrar as mensagens enviadas entre as classes.

O ator *host* foi empregado para demonstrar o início da troca de mensagens entre as classes, partindo-se dos métodos *install()*, *select()*, *deselect()* e *process()*.

Em todos os diagramas, após chamada do método *process()* verifica-se que determinadas instruções serão executadas dependendo do valor contido em INS que, neste caso, refere-se à instrução a ser executada no *smart card*.

3.2.3.1 *Package senha*

No diagrama de seqüência do *package senha*, o ator *host* pode efetuar chamadas aos métodos *install()*, *select()*, *deselect()* e *process()* ao *applet CGSSenhaApplet*.

Além de instalar e registrar o *applet* no *smart card*, o método *install()* instancia um objeto da classe *OwnerPIN* que será o objeto responsável pelo gerenciamento da senha, no que diz respeito a validação, bloqueio e armazenamento da mesma. A classe *OwnerPIN* pertence ao *package javacard.framework*.

Nos métodos *select()* e *deselect()* nenhuma operação é efetuada, porém, no método *process()* é verificado o comando APDU enviado pelo *host*. Caso a instrução (*byte* INS) desse comando tenha o valor 0x01, significa que uma nova senha deverá ser armazenada no objeto da classe *OwnerPIN* através da chamada ao método *update()*. Caso a instrução possua o valor 0x02, o método *resetAndUnblock()* será invocado de modo a efetuar o desbloqueio da senha e reiniciar a contagem de entradas inválidas para a mesma. Por fim, caso a instrução tenha o

valor 0x03, significa que uma senha recebida no comando APDU deverá ser validada com a senha previamente gravada, através da chamada ao método *check()*.

Disponibilizando uma interface compartilhada, o *applet CGSSenhaApplet* permite que outros *applets* sejam capazes de acessar determinados métodos da instância da classe *OwnerPIN* possibilitando a validação de uma senha com a senha já existente neste objeto. Para ilustrar o uso desta interface, pode-se observar no diagrama o ator *Applet* que, possuindo uma instância de classe que implementa a interface *CGSSenhaInterface*, poderá enviar mensagens ao *applet CGSSenhaApplet* de modo a efetuar verificações com relação a senha gravada no objeto da classe *OwnerPIN*.

Os métodos disponibilizados pela *interface CGSSenhaInterface* para acesso ao objeto da classe *OwnerPIN* que está agregado à classe *CGSSenhaApplet* são listados a seguir:

- a) *checarSenha()*: esse método faz chamada ao método *isValidated()* de modo a verificar se uma senha válida foi previamente informada;
- b) *finalizarSenha()*: efetua chamada ao método *resetAndUnblock* de modo a reiniciar o contador interno que mantém a quantidade de senhas informadas incorretamente;
- c) *motivoSenhaIncorreta()*: caso o método *checarSenha()* retorne falso (senha inválida), o método *motivoSenhaIncorreta()* informa se a senha recebida no comando APDU enviado pelo *host* está bloqueada ou incorreta;
- d) *tentativasRestando()*: efetua chamada ao método *getTriesRemaining()* que retorna o número de tentativas restantes para informar a senha corretamente;
- e) *validarSenha()*: esse método invoca o método *check()* afim de validar a senha recebida no comando APDU enviado pelo *host* com a senha previamente cadastrada.

O diagrama de seqüência do *package senha*, com todas as funcionalidades descritas anteriormente, pode ser observado na Figura 10.

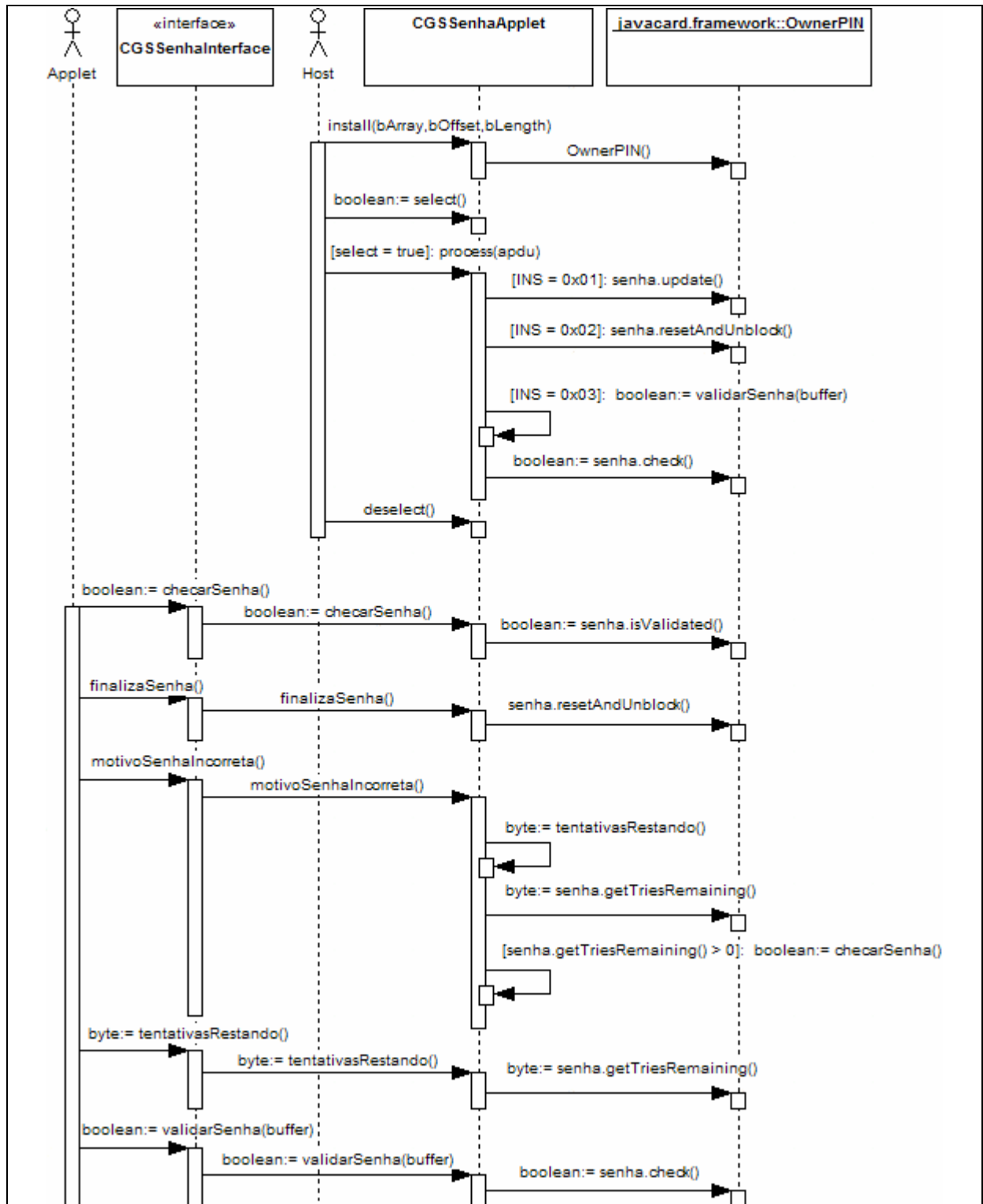


Figura 10 – Diagrama de seqüência: *package senha*

3.2.3.2 *Package dinheiroEletronico*

Na Figura 11 é ilustrado o diagrama de seqüência do *package dinheiroEletronico*.

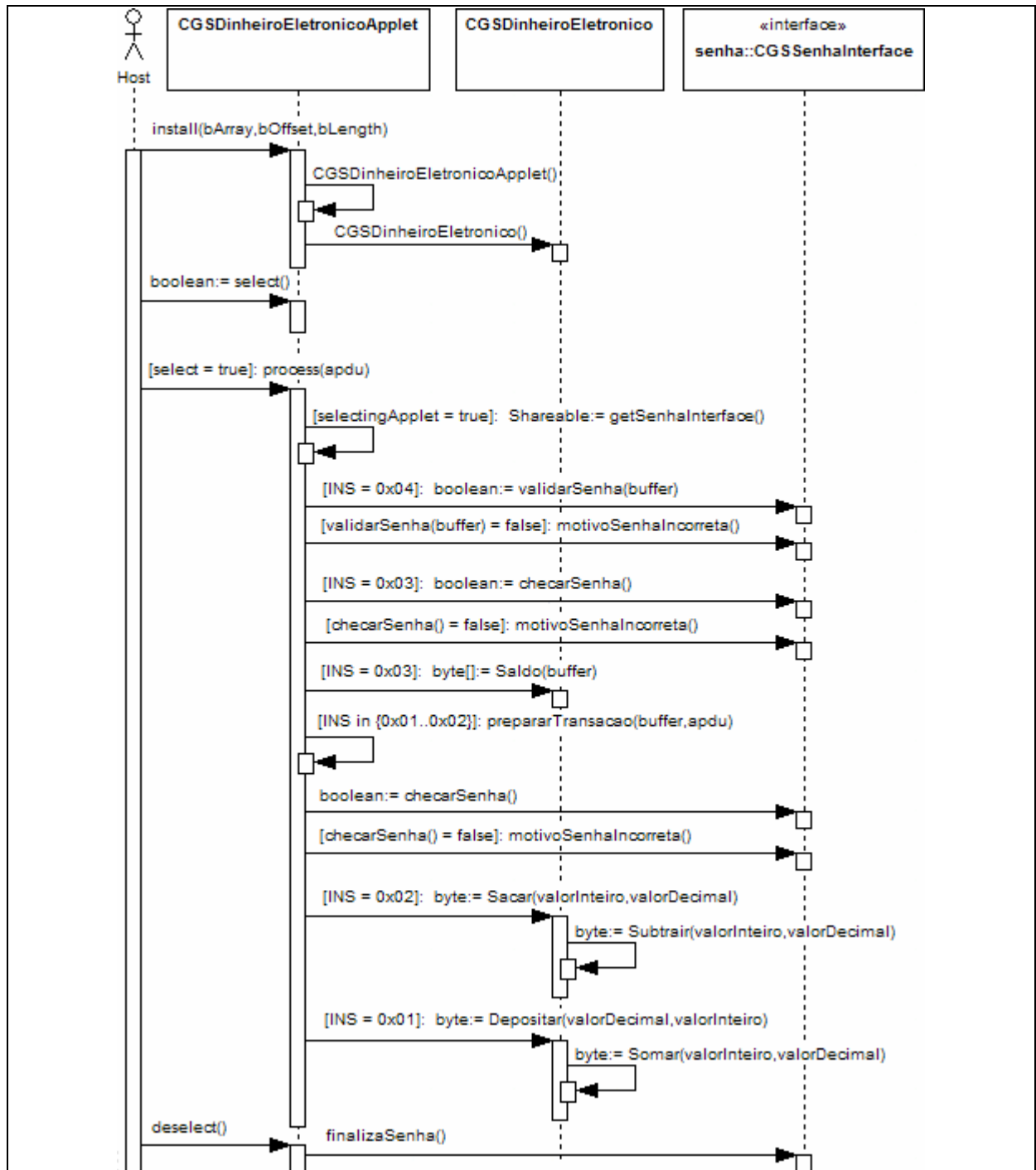


Figura 11 – Diagrama de seqüência: *package dinheiroEletronico*

Neste diagrama, verifica-se que no método *install()* é instanciado um objeto da classe *CGSDinheiroEletronicoApplet* que, por sua vez, instancia um objeto da classe *CGSDinheiroEletronico* que irá conter os valores em dinheiro eletrônico. No momento da instalação esses valores são inicializados com zero.

No método *select()* nenhuma ação é tomada, porém, no método *deselect()*, caso uma

senha válida tenha sido informada ocorre a chamada ao método *finalizaSenha()* da interface *CGSSenhaInterface*.

Quando a JCRE invoca o método *select()*, o método *process()* é chamado em seguida. Através da propriedade lógica *javacard.framework.Applet.selectingApplet* é possível identificar que o comando APDU enviado pelo *host* é um comando de seleção para o *applet*. Neste momento, conforme diagrama apresentado, é feita chamada ao método *getSenhaInterface()* que irá retornar uma referência à interface *CGSSenhaApplet* que estende da classe *Shareable*.

Ainda no método *process()*, caso a instrução recebida (*byte* INS) possua os valores 0x01 ou 0x02, será feita chamada ao método *prepararTransação()* que irá verificar através de chamada ao método *checarSenha()* da interface *CGSSenhaInterface* se uma senha válida foi previamente informada. Se esse método retornar falso será feita chamada ao método *motivoSenhaIncorreta()* da interface *CGSSenhaInterface* que irá informar se a senha é inválida ou está bloqueada. Caso contrário será extraído do comando APDU recebido, o valor a ser depositado ou sacado e, então, chamadas aos métodos *Depositatar()* e *Somar()* ou *Sacar()* e *Subtrair()*, da classe *CGSDinheiroEletronicos*, serão invocados.

Caso a instrução recebida no comando APDU enviado pelo *host* tenha o valor 0x03 e uma senha válida tenha sido previamente informada, o método *Saldo()* da classe *CGSDinheiroEletronico* será chamado e irá retornar o valor do dinheiro eletrônico existente.

Caso a instrução possua valor 0x04, será invocado o método *validarSenha()* da interface *CGSSenhaInterface* que irá verificar se a senha informada no comando APDU é válida. Caso senha seja inválida, chamada ao método *motivoSenhaIncorreta()* será efetuada.

3.2.3.3 *Package documentosEletronicos*

No diagrama de seqüência da Figura 12 é apresentado, inicialmente, o método *install()* que instancia um objeto da classe *CGSDocumentosEletronicosApplet* e, na seqüência, um objeto da classe *CGSDocumentosEletronicos*, sendo que este último irá conter as instâncias de cada classe responsável por cada documento. No diagrama de seqüência, não foram utilizadas as classes específicas para cada documento, pelo fato destas possuírem apenas métodos para gravação e leitura (*getters* e *setters*) das informações, sendo que estas chamadas partiriam da classe *CGSDocumentosEletronicos* que agrega, portanto, as classes *CGSDocumentosEletronicosRG*, *CGSDocumentosEletronicosTE*, *CGSDocumentosEletronicosCPF*, cada qual responsável respectivamente, pelos documentos RG, Título de Eleitor e CPF.

No método *process()*, caso a instrução (*byte* INS) enviada no comando APDU pelo *host* possua os valores 0x01, 0x02, 0x03 ou 0x04, serão chamados do objeto *CGSDocumentosEletronicos*, respectivamente, para cada instrução, os métodos *getDadosComuns()*, *getCpf()*, *getTituloEleitor()* e *getRegistroGeral()*. Caso a instrução tenha os valores 0x06, 0x07, 0x08 ou 0x09, será feita, respectivamente, para cada instrução, chamada aos métodos *setDadosComuns()*, *setCPF()*, *setTituloEleitor()* e *setRegistroGeral()* passando como parâmetros os dados a serem armazenados nos objetos responsáveis por cada documento. Anterior à chamada desses métodos é invocado o método *preparaTransacao()*, que extrai do comando APDU enviado pelo *host*, as informações referentes ao documento a ser armazenado.

Para os métodos *getDadosComuns()*, *getCpf()*, *getTituloEleitor()* e *getRegistroGeral()* é feita chamada adicional ao método *enviarDocumentos()* que tem a função de enviar ao host as informações retornadas pelos métodos invocados.

Os métodos *select()* e *deselect()* na classe *CGSDocumentosEletronicosApplet* têm funcionalidades idênticas aos métodos de mesmo nome apresentados no *applet CGSDinheiroEletronicoApplet*, assim como a autenticação com senha pessoal que é feita da mesma maneira, apenas diferenciando o número da instrução (0x05).

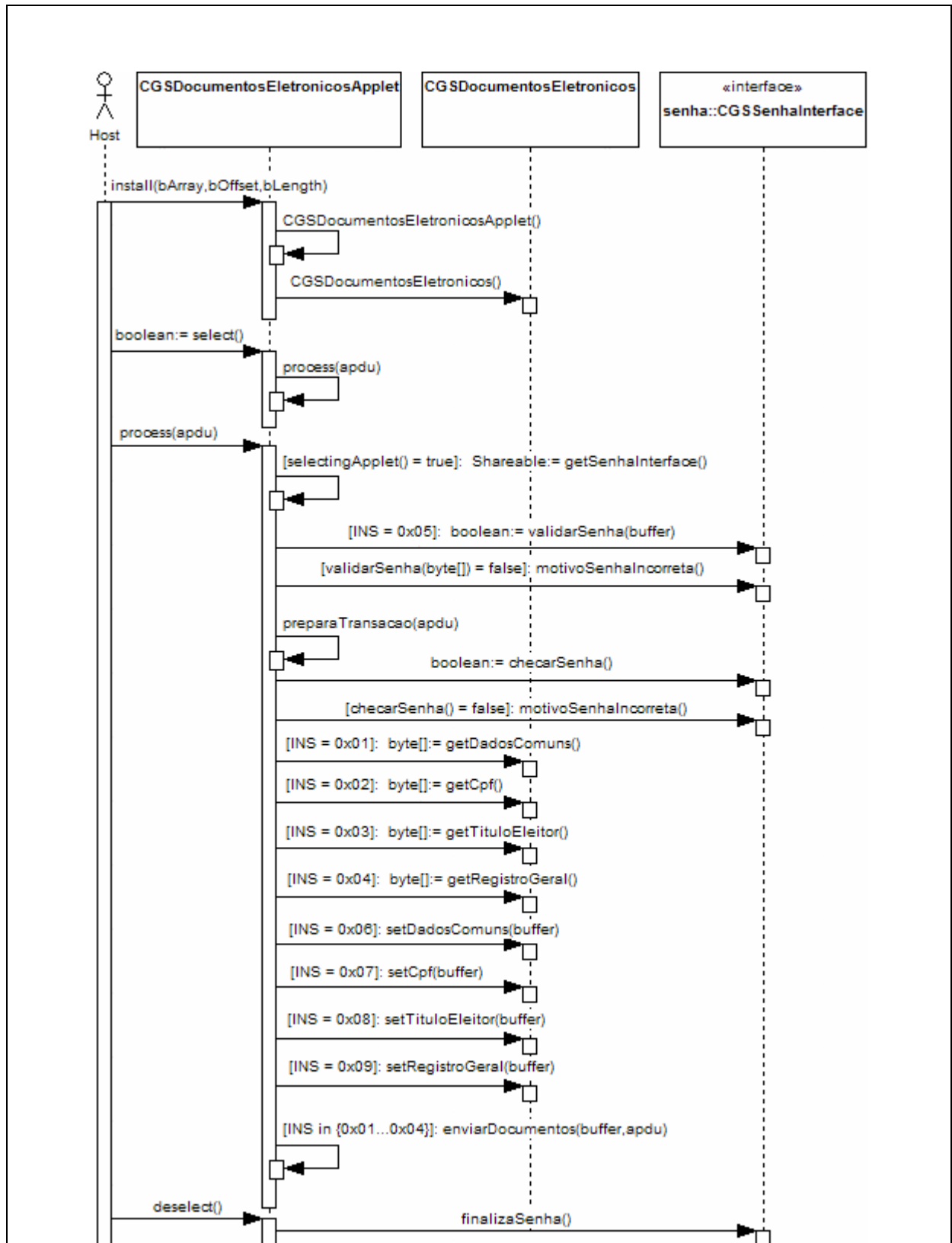


Figura 12 - Diagrama de seqüência: *package documentosEletronicos*

3.2.3.4 *Package servicosSeguranca*

No diagrama de seqüência da Figura 13, o método *install()* instancia o objeto da classe *CGSCertificadoDigitalApplet* e, na seqüência, objetos da classe *CGSChavesPublicasCA* (responsável por manter chaves públicas de entidades certificadoras confiáveis) e *CGSCertificadoDigital* (responsável por manter o par de chaves assimétricas e certificado digital do usuário).

Com relação ao método *process()*, caso a instrução do comando APDU enviado pelo *host* tenha o valor 0x01 será feita chamada ao método *adicionarChavePublicaCA()*. Neste método é feita chamada ao construtor da classe *CGSChavePublica* que instancia uma chave publica RSA, e atribui a esta chave o módulo e expoente passados como parâmetro no método construtor.

A instrução 0x02 tem duas funções sendo que a primeira delas é gravar o tamanho do certificado digital a ser armazenado, através da chamada do método *setCertificadoTamanho()* da classe *CGSCertificadoDigital()*. Esse método será invocado caso os parâmetros P1 e P2, contidos no comando APDU enviado pelo *host*, possuam, cada qual, o valor 0x00. A segunda função da instrução armazena fragmentos do certificado digital enviados pelo *host* através de chamadas ao método *setCertificado()*. Neste contexto, P1 representa o número do fragmento a ser gravado e P2 o tamanho, em *bytes*, deste fragmento.

A instrução 0x05 funciona de forma semelhante à instrução 0x02, porém, neste caso, os métodos *getCertificadoTamanho()* e *getCertificado()* tem a função de extrair os dados do certificado digital previamente armazenado. Estes métodos podem ser invocados desde que o método *existeCertificadoDigital()* retorne o valor verdadeiro, indicando a existência de um certificado digital.

Por fim, a instrução 0x03 tem a função de retornar a chave pública do usuário do *smart*

card através do método *getChavePublica()* da classe *CGSCertificadoDigital*, desde que o par de chaves assimétricas desde usuário exista, o que pode ser verificado através do método *existeParChaves()*.

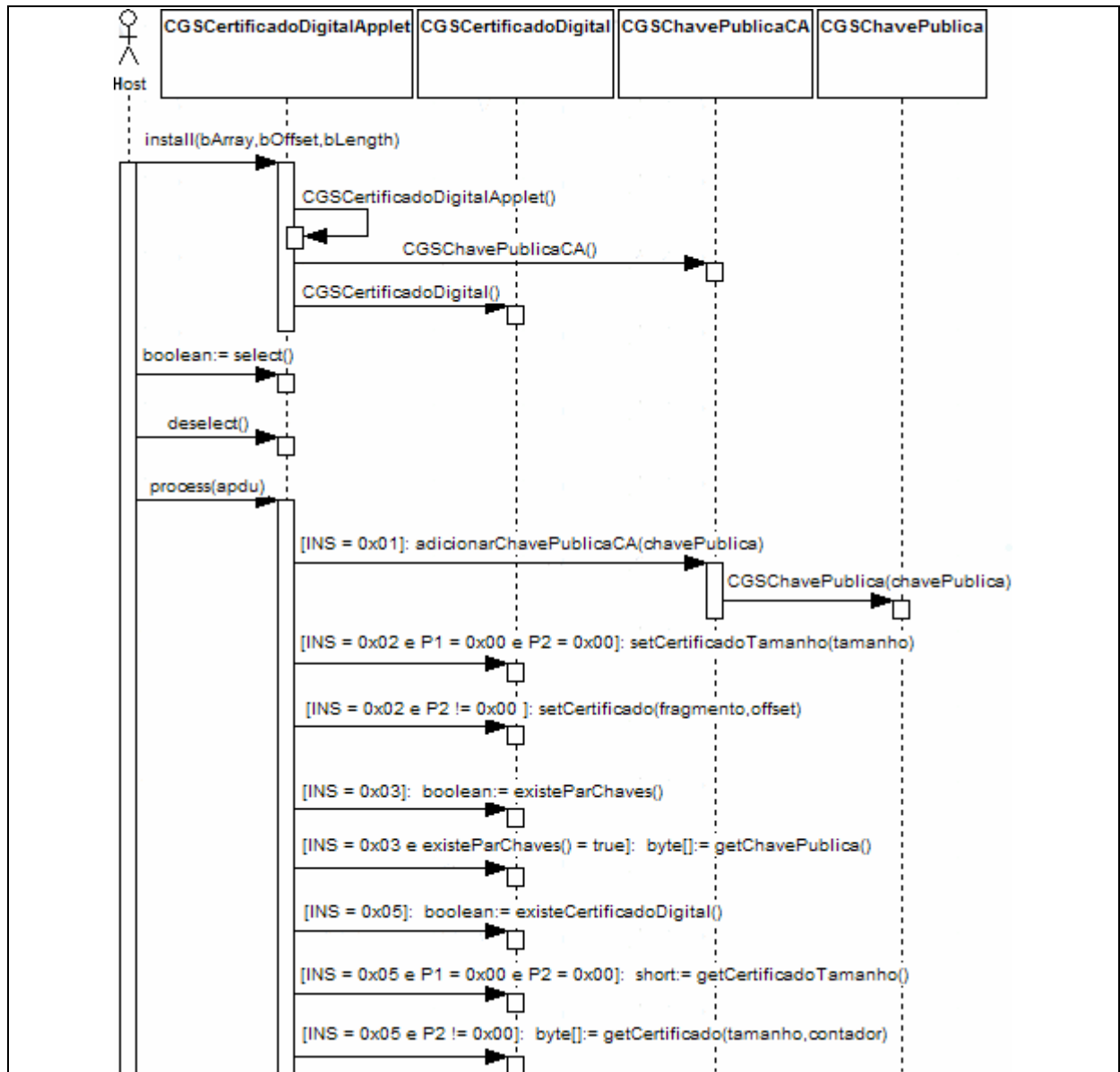


Figura 13 - Diagrama de seqüência: *package servicosSeguranca*

3.2.4 Comandos e respostas APDU dos *applets*

Cada um dos *applets* atende a comandos e respostas APDU específicos, conforme especificado na seqüência.

3.2.4.1 CGSDinheiroEletronicoApplet

O *applet* para gerenciamento do dinheiro eletrônico possui quatro funções básicas que seriam depósito, saque, consulta de saldo e autenticação com senha.

Nas funções depósito e saque o *host* envia ao *smart card* três *bytes* referentes à quantidade a ser depositada ou sacada. Como a tecnologia Java *card* não possui suporte a tipos *float* ou *currency* é necessário que a parte inteira do valor seja armazenada em um *short* (2 *bytes*) e a parte decimal seja armazenada em um *byte*, logo, esses três bytes representam o valor. O mesmo ocorre na função que recupera o saldo existente no *smart card* retornando três *bytes*.

No Quadro 2 são apresentados os comandos APDU suportados pelo *applet*:

Comando APDU							
Função	CLA	INS	P1	P2	LC	Dados	LE
depósito	0x00	0x01	0x00	0x00	3 bytes	valor para depósito	-
saque	0x00	0x02	0x00	0x00	3 bytes	valor sacado	-
saldo	0x00	0x03	0x00	0x00	-	-	3 bytes
validar senha	0x00	0x04	0x00	0x00	tamanho da senha	senha do usuário	-

Quadro 2 – Comandos APDU do *applet* CGSDinheiroEletronicoApplet

3.2.4.2 CGSDocumentosEletronicosApplet

Para o gerenciamento de documentos eletrônicos, o *applet* dispõe de funções para gravação e leitura, além de autenticação do usuário do *smart card* através da senha pessoal.

Nas funções de leitura, responsáveis por buscar as informações dos documentos armazenados no *smart card*, observa-se que o tamanho do dado esperado (LE) pelo *host* é zero. Como as informações podem ter tamanhos variados para cada usuário de cada *smart card*, o *host* deverá assumir um tamanho variável a ser recebido depois de invocadas quaisquer dessas funções.

Para as funções de gravação dos dados dos documentos é necessário observar que

todas as informações de cada comando são enviadas de uma só vez. Desta forma, para que o *smart card* possa distinguir cada dado, após cada um deles, o *host* deverá concatenar o byte 0x00 (que representa um caractere nulo) possibilitando que a gravação de cada campo ocorra corretamente em cada documento. Da mesma forma, na leitura dos dados, estes também estarão separados por tal caractere.

Para que o método adotado seja viável, as informações devem ser enviadas ao *smart card* na ordem definida em cada comando presente no Quadro 3.

Comando APDU							
Função	CLA	INS	P1	P2	LC	Dados	LE
ler dados comuns	0x00	0x01	0x00	0x00	-	-	0x00
ler CPF	0x00	0x02	0x00	0x00	-	-	0x00
ler Título de Eleitor	0x00	0x03	0x00	0x00	-	-	0x00
ler RG	0x00	0x04	0x00	0x00	-	-	0x00
validar senha	0x00	0x05	0x00	0x00	tamanho da senha	senha do usuário	-
gravar dados comuns	0x00	0x06	0x00	0x00	tamanho dados	nome completo, 0x00, data nascimento, 0x00	-
gravar CPF	0x00	0x07	0x00	0x00	tamanho dados	número, 0x00, data emissão, 0x00	-
gravar Título de Eleitor	0x00	0x08	0x00	0x00	tamanho dados	número de inscrição, 0x00, zona eleitoral, 0x00, seção, 0x00, município, 0x00, UF, 0x00, data emissão, 0x00	-
gravar RG	0x00	0x09	0x00	0x00	tamanho dados	nome mãe, 0x00, nome Pai, 0x00, naturalidade, 0x00, documento origem, 0x00, data emissão, 0x00	-

Quadro 3 – Comandos APDU do *applet CGSDocumentosEletronicosApplet*

Outra característica do *applet* em questão é que sempre que a leitura de um documento for efetuada, os dados retornados pelo *applet* incluirão os dados comuns previamente armazenados. Sendo assim, as respostas aos comandos de leitura tem o formato conforme mostrado no Quadro 4.

Resposta APDU	
Função	Dados retornados
ler dados comuns	nome completo, 0x00, data nascimento, 0x00
ler CPF	nome completo, 0x00, data nascimento, 0x00, número, 0x00, data emissão, 0x00
ler Título de Eleitor	nome completo, 0x00, data nascimento, 0x00, número de inscrição, 0x00, divisão, 0x00, zona eleitoral, 0x00, seção, 0x00, município, 0x00, UF, 0x00, data emissão, 0x00
ler RG	nome completo, 0x00, data nascimento, 0x00, nome mãe, 0x00, nome Pai, 0x00, naturalidade, 0x00, documento origem, 0x00, data emissão, 0x00

Quadro 4 – Respostas APDU do *applet CGSDocumentosEletronicosApplet*

3.2.4.3 *CGSSenhaApplet*

O *applet* responsável por gerenciar a senha possui três funções sendo, gravação da senha, desbloqueio e verificação, nas quais os respectivos comandos APDU são mostrados no

Quadro 5:

Comando APDU							
Função	CLA	INS	P1	P2	LC	Dados	LE
gravar senha	0x00	0x01	0x00	0x00	tamanho dados	nova senha	-
desbloqueio	0x00	0x02	0x00	0x00	0x00	-	-
validar senha	0x00	0x03	0x00	0x00	tamanho dados	senha	-

Quadro 5 – Comandos APDU do *applet CGSSenhaApplet*

É definido no *applet* que a senha pode ter até dez caracteres e que o limite de entradas inválidas antes que a senha seja bloqueada é de quatro tentativas.

3.2.4.4 *CGSCertificadoDigitalApplet*

O *applet* responsável pelo gerenciamento de chaves e certificado digital possui uma série de peculiaridades em seus comandos e respostas APDU, com relação aos demais *applets*.

Antes da análise dos comandos apresentados no Quadro 6 é importante mencionar que o par de chaves assimétricas do usuário, que utiliza o algoritmo Rivest-Shamir-Adleman

(RSA) com 512 *bytes*, é instanciado e inicializado quando o *applet* é instalado e registrado. Logo, não há comandos APDU que façam a gravação das chaves no *smart card*, ou seja, uma vez instalado este *applet*, o par de chaves automaticamente passa a existir.

Para efetuar a gravação de chaves públicas das autoridades certificadoras no *smart card*, o *host* deverá montar um vetor de bytes onde, os dois primeiros *bytes* representam o módulo da chave, os dois *bytes* seguintes representam o expoente da chave e, em seguida, esse vetor deverá conter respectivamente, o módulo e o expoente. Isso se faz necessário para que seja possível reconstruir a chave pública no *smart card* (utilizando um objeto *RSAPublicKey*).

Com relação à gravação e leitura de certificado digital, nota-se que há dois comandos APDU em cada uma das funções. Verifica-se também que existem diferenças entre os cabeçalhos de ambos comandos para cada função, pelo fato de utilizar-se dos parâmetros P1 e P2.

É necessário o uso desses parâmetros, pois no *smart card* é possível o envio e/ou recebimento de no máximo 256 *bytes* de uma só vez. Como um certificado digital ultrapassa esse limite, é necessário que a gravação e/ou leitura se faça em fragmentos até que o processo esteja completo, ou seja, até que a leitura e/ou gravação de todo certificado digital aconteça.

Nessas funções, portanto, quando os parâmetros P1 e P2 possuírem valor nulo (0x00), é assumido que a rotina de leitura ou gravação do tamanho (em *bytes*) do certificado digital é solicitado. Caso contrário o valor de P1 multiplicado pelo valor de P2, representa a posição no vetor de *bytes* a partir da qual, a quantidade de *bytes* equivalente ao valor de P2 deve ser lido e/ou gravado.

Comando APDU							
Função	CLA	INS	P1	P2	LC	Dados	LE
gravar K_U autoridade certificadora	0x00	0x01	0x00	0x00	tamanho dados	tamanho módulo (2 bytes), tamanho expoente (2 bytes), módulo, expoente	-
grava certificado digital usuário	0x00	0x02	0x00	0x00	0x02	2 bytes representando tamanho do certificado	-
	0x00	0x02	contador	0x7F	0x7F	fragmento do certificado digital	-
ler K_U usuário	0x00	0x03	0x00	0x00	-	-	0x00
ler certificado digital	0x00	0x05	0x00	0x00	-	-	0x02
	0x00	0x05	contador	0x7F	-	-	0x00

Quadro 6 – Comandos APDU do *applet CGSCertificadoDigitalApplet*

De acordo com o Quadro 7, a chave pública do usuário é enviada pelo *smart card* ao *host*, com o mesmo formato na qual as chaves públicas das autoridades certificadoras são enviadas do *host* para o *smart card*. Neste mesmo quadro, observa-se também a resposta com relação ao certificado digital ao comando de leitura de um de seus fragmentos ou tamanho.

Resposta APDU	
Função	Dados retornados
ler K_U usuário	Tamanho módulo (2 bytes), tamanho expoente (2 bytes), módulo, expoente
ler certificado digital	2 bytes representando o tamanho do certificado digital
	Fragmento do certificado digital com tamanho equivalente ao valor de P2, a partir da posição $P1 * P2$

Quadro 7 – Respostas aos comandos APDU do *applet CGSCertificadoDigitalApplet*

3.2.4.5 Status Word específicos dos applets desenvolvidos

Conforme citado, com relação ao protocolo APDU, o *smart card* sempre irá retornar um código de dois bytes que representa o *status word* do comando executado. Para todos os *applets*, portanto, foram definidos vários códigos conforme apêndice D.

A classe que define estes códigos chama-se *constantes.CGSStatusWords*.

3.3 IMPLEMENTAÇÃO

Nesta seção do trabalho são abordadas características dos softwares e hardwares

utilizados. Além deste assunto são apresentados detalhes de implementação com relação à estrutura básica de um *applet*, utilização de *interface* compartilhada entre *applets*, o funcionamento da persistência de dados e transações, o envio de dados ao *host*, geração de chaves assimétricas e armazenamento do certificado digital e, por fim, os testes efetuados com o uso do *JCOP Shell* utilizando o *smart card* real.

3.3.1 *Software e Hardware* utilizados

A ferramenta utilizada para codificação foi o IDE IBM Eclipse versão 3.1 (executando sobre Java 2 SDK versão 1.4.2) com processo de desenvolvimento e testes iniciais de todos os *applets* sendo efetuado utilizando-se o *Java card SDK* versão 2.2.1.

No decorrer do desenvolvimento foram adquiridos, o *plugin* para o IDE Eclipse chamado IBM *JCOP Tools* versão 2.2 41 juntamente com um *smart card* (onde este último libera a licença de uso para o *plugin*) e também, um CAD da empresa *Perto Software*. A partir de então, tanto desenvolvimento como testes foram voltados à aplicação utilizando o *hardware e software* adquiridos.

Segundo PHILIPS SEMICONDUCTORS (2004), o *smart card* utilizado para os testes (visto na Figura 14) possui 72 *Kbytes* de memória EEPROM, 160 *Kbytes* de ROM, 4608 *bytes* (4.5 *Kbytes*) de RAM, co-processador RSA, *Advanced Encryption Standard* (AES) e *Triple Data Encryption Standard* (3DES) (independentes), interface dos contatos conforme a norma ISO/IEC 7816 sendo que o tempo de retenção da EEPROM é de vinte anos, no mínimo.



Figura 14 – *Smart card* utilizado para testes (*JCOP tools smart card*)

De acordo com PERTO SOFTWARE (2005), o CAD utilizado para os testes, visto na Figura 15, possui dimensões 13 x 66 x 62mm e interface com o *host* do tipo *Universal Serial Bus* (USB), suportando apenas os sistemas operacionais Windows 98, 2000, Me e XP. A alimentação necessária de 5v é fornecida através da própria interface. A interface do CAD com o *smart card* é compatível com as normas ISO/IEC 7816.



Figura 15 – Leitor e gravador de *smart cards* utilizado para testes

O algoritmo RSA com chaves de 512 *bits* foi utilizado para gerar as chaves assimétricas no *applet* responsável por esta função. A escolha deste algoritmo deu-se pelo fato do mesmo estar disponível no *smart card* adquirido, além de ser mundialmente utilizado.

3.3.2 Estrutura básica de um *applet*

Um *applet* Java card estende da classe abstrata *javacard.framework.Applet*, logo, deverá implementar determinados métodos que serão chamados pela JCVM. A estrutura básica de um *applet* pode ser visualizada no Quadro 8.

```
import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;

public class CGSCLasseTeste extends Applet {

    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new CGSCLasseTeste().register();
    }

    public boolean select() {
        return true;
    }

    public void deselect() {
    }

    public void process(APDU apdu) {

        byte[] buf = apdu.getBuffer();

        if (buffer[ISO7816.OFFSET_CLA] != 0x00) {
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        }

        switch (buf[ISO7816.OFFSET_INS]) {
            case (byte) 0x00:
                break;
            default:
                ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }
}
```

Quadro 8 – Estrutura básica de um *applet* Java card

O método *install()* é o método invocado quando o *applet* é instalado no *smart card*, logo, será chamado apenas uma vez. Neste método, é importante que todos os objetos que serão utilizados pelo *applet* sejam instanciados e inicializados sempre que possível, de modo a garantir o recurso de memória disponível no *smart card* para o *applet* evitando assim, possíveis problemas com falta de memória durante o uso normal do *smart card*. Depois de feitas as inicializações necessárias, o método *register()* deve ser invocado para finalmente registrar o *applet* no *smart card* que estará então, apto a ser selecionado e utilizado.

O método *select()* é invocado imediatamente quando o *applet* é selecionado pela JCVM enquanto que o método *deselect()* é chamado imediatamente antes do *applet* ser desselecionado, possibilitando nestes métodos a inicialização ou finalização de variáveis ou objetos.

O método *process()* é o método responsável pelo recebimento de comandos APDU enviados pelo *host*, sendo o comando passado como parâmetro no objeto *apdu* da classe *APDU*.

Para extrair efetivamente o comando do objeto *apdu*, utiliza-se o método *getBuffer()* que retorna um vetor de *bytes* contendo o comando APDU no formato enviado pelo *host*.

Como boas técnicas de programação, neste caso, deve-se inicialmente verificar no comando APDU recebido se a classe (CLA) enviada pelo *host* é uma classe válida. Caso não seja deve ser lançada uma exceção indicando ao *host* essa divergência. Para tal, é utilizado um *status word* padrão definido na classe *ISO7816* (*SW_CLA_NOT_SUPPORTED*).

Caso a classe seja válida, deve-se verificar qual instrução (INS) foi solicitada pelo *host* ao *applet*. Caso a instrução seja inválida pode-se lançar uma exceção indicando o problema ao *host*, mais uma vez utilizando-se de um *status word* padrão definido na classe *ISO7816* (*SW_INS_NOT_SUPPORTED*).

Pelo fato da análise do comando APDU se dar em nível de *byte*, para melhor leitura e análise do código, cria-se variáveis estáticas cada qual representando uma instrução a ser executada pelo *applet*, conforme código parcial do *applet* *CGSDinheiroEletronicoApplet* mostrado no Quadro 9.

```

.
.
public class CGSDinheiroEletronicoApplet extends Applet {
.
.
.
    //constantes que definem instruções válidas
    final static byte DEPOSITAR      = (byte)0x01; //efetua depósito
    final static byte SACAR          = (byte)0x02; //efetua saque
    final static byte SALDO          = (byte)0x03; //retorna saldo
    final static byte INFORMA_SENHA  = (byte)0x04; //informa senha
.
.
.
    public void process(APDU apdu) throws ISOException {
.
.
.
        switch (buffer[ISO7816.OFFSET_INS]) {
            case DEPOSITAR:      ... break;
            case SACAR:          ... break;
            case SALDO:          ... break;
            case INFORMA_SENHA: ... break;
            default:
                ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
                break;
        }
    }
.
.
.
}

```

Quadro 9 – Seleção da instrução (INS) no método *process()*

3.3.3 Autenticação com senha

O *applet CGSSenhaApplet* disponibiliza aos demais *applets* registrados no *smart card* uma *interface* compartilhada de modo que esses *applets* possam acessar determinados métodos que dizem respeito a senha. Para que isso seja possível, é necessário que o *applet CGSSenhaApplet* declare o método *getShareableInterfaceObject(AID clienteAID, byte segredo)* e retorne a referência ao próprio objeto, conforme Quadro 10.

O parâmetro *clienteAID* refere-se ao *applet* que está solicitando o objeto da interface para acesso à senha. Desta forma, é possível analisar o parâmetro *clienteAID* recebido e decidir entre retornar ou não a referência à *interface*. Além deste parâmetro, há ainda o parâmetro *segredo* que se trata de um *byte* que pode ser utilizado como uma segunda verificação. Como exemplo, poderia ser acordado o compartilhamento de *applets* entre empresas parceiras. Assim cada uma delas poderia ceder o AID e o segredo à parceira, de

modo a disponibilizar o acesso à *interface* de seus *applets* compartilhados.

```

.
.
. public class CGSSenhaApplet extends Applet implements CGSSenhaInterface {
    final static byte BYTEVERIFICACAO = (byte)0x00;
.
.
. public Shareable getShareableInterfaceObject(AID clienteAID, byte segredo) {
    if (BYTEVERIFICACAO != segredo) {
        return null;
    }
    return (this);
}
.
.
.
}

```

Quadro 10 – Declaração do método *getShareableInterfaceObject*

Para que os *applets* responsáveis pelo gerenciamento do dinheiro e documentos eletrônicos possam obter o objeto dessa interface (*CGSSenhaInterface*), é necessário que estes efetuem uma chamada ao método *getAppletShareableInterfaceObject* da classe *JCSyssem* passando como parâmetro o AID do *applet* que contém a *interface* e o *byte* de verificação (segredo), conforme Quadro 11, método *getSenhaInterface()*.

Para tal, é necessário que seja efetuada chamada do método *JCSyssem.lookupAID* passando como parâmetro o AID do *applet* procurado. Esse método efetua uma busca no registro no JCRE, retornando um objeto AID do *applet* solicitado caso seja encontrado. Feito isso é possível invocar o método *getAppletShareableInterfaceObject* afim de receber a referência da *interface* do *applet*.

```

.
.
. private Shareable getSenhaInterface() {

    final byte BYTEVERIFICACAO = (byte)0x00;

    //AppletID da Interface CGSSenhaApplet - senhaApplet
    byte[] senhaAppletInterfaceAID = {
        0x73, 0x65, 0x6E, 0x68, 0x61, 0x41, 0x70, 0x70, 0x6C, 0x65, 0x74 };

    //busca de entrada no JCRE com o AID informado
    AID senhaInterfaceAID = JCSysyem.lookupAID(
        senhaAppletInterfaceAID, (short)0, (byte)senhaAppletInterfaceAID.length );

    if (senhaInterfaceAID == null) {
        ISOException.throwIt(CGSStatusWords.CGS_SW_AID_NAOEXISTE);
    }

    //retorna Interface Shareable do applet CGSSenhaApplet
    Shareable senhaInterface =
        JCSysyem.getAppletShareableInterfaceObject(senhaInterfaceAID,
                                                    BYTEVERIFICACAO );

    if (senhaInterface == null) {
        ISOException.throwIt(CGSStatusWords.CGS_SW_INTF_NAOEXISTE);
    }

    return senhaInterface;
}
.
.
.

```

Quadro 11 – Método *getSenhaInterface()* que busca objeto da interface *CGSSenhaApplet*

A referência a interface *CGSSenhaInterface* é utilizada durante toda a sessão CAD.

Por esse motivo, a chamada ao método *getSenhaInterface()* ocorre durante a seleção do *applet*, no método *process()*, sendo a referência descartada no método *deselect()*, conforme mostrado no Quadro 12.

```

.
.
. public void process(APDU apdu) throws ISOException {

    if (selectingApplet()) {
        iSenha = (CGSSenhaInterface)(getSenhaInterface());
        return;
    }
    .
    .
}
.
.
. public void deselect () {
    .
    .
    iSenha = null;
}
.
.

```

Quadro 12 – Chamada do método *getSenhaInterface()*

3.3.4 Persistência de dados e transações

Applets mantém, entre sessões CAD, as referências de objetos instanciados assim como os valores de suas propriedades. Assim sendo, qualquer alteração em qualquer propriedade é feita de forma atômica.

Para garantir que um grupo de propriedades seja alterado de forma atômica, Java *card* oferece o conceito de transação, que funciona da mesma forma que um SGBD, onde uma transação é aberta, operações são realizadas e, em caso de sucesso a transação será efetivada, caso contrário todas as operações serão desfeitas.

Para manter dados no *smart card*, o processo inicia-se no recebimento do comando APDU no método *process()*. Para demonstrar esse processo será utilizado o código parcial do *applet* responsável pelo gerenciamento de documentos eletrônicos, utilizando o documento CPF conforme mostrado no Quadro 13.


```

.
.
public class CGSDocumentosEletronicosApplet extends Applet{

    private CGSDocumentosEletronicos documentos;
    private byte[] dados = null;
.
.
.
    public void process(APDU apdu) throws ISOException {
        .
        .
        byte[] buffer = apdu.getBuffer();
        .
        .
        .
        case W_CPF:
            preparaTransacao(apdu, buffer);
            try {
                documentos.setCpf(dados);
            }
            catch (ISOException e) {
                ISOException.throwIt(e.getReason());
            }
            break;
        .
        .
    }//fim process
.
.
.
    private void preparaTransacao(APDU apdu, byte[] buffer) {
        .
        .
        .
        apdu.setIncomingAndReceive();
        dados = new byte[buffer[ISO7816.OFFSET_LC]];
        Util.arrayCopy(
            buffer, ISO7816.OFFSET_CDATA, dados, (short)0, buffer[ISO7816.OFFSET_LC]);
    }//fim preparaTransacao
.
.
.
} //fim classe

```

Quadro 13 – Início do processo para persistência de dados no *smart card*

Inicialmente é efetuada a chamada do método *preparatransacao()*. Neste método, através do objeto *APDU apdu* é invocado o método *setIncomingAndReceive()* que indica que há *bytes* a serem recebidos pelo *applet*. Esses *bytes* serão, portanto, copiados para a variável *buffer*.

Depois da chamada ao método *setIncomingAndReceive()* é associada à variável *dados* um número de posições igual ao número de *bytes* recebidos. Por fim, será copiado para *dados* o conteúdo de *buffer* a partir da posição indicada por *ISO7816.OFFSET_CDATA*, ou seja, a posição a partir da qual estão os dados enviados pelo *host*.

Com isso o método *preparaTransacao()* é encerrado permanecendo os *bytes* recebidos na variável *dados* que será passada como parâmetro para o objeto *documentos* no método

setCpf().

A seguir, no método *setCPF()* da classe *CGSDocumentosEletronicos* é necessário separar as informações (neste exemplo, número do documento e data de emissão) conforme definido na especificação dos comandos APDU para o *applet* *CGSDocumentosEletronicosApplet*. Essa separação é feita com a chamada do método *extraiDados()*, conforme demonstrado no Quadro 14, passando como parâmetro o *buffer* que contém os dados e a posição inicial da qual serão extraídos os *bytes* até que seja encontrada o *byte* nulo 0x00.

```

.
.
public class CGSDocumentosEletronicos {

    private CGSDocumentosEletronicosCPF cpf;
    .
    .
    public void setCpf(byte[] buffer) {
    .
    .
        byte inicio          = (byte)0x00;
        byte[] numero        = null;
        byte[] dataEmissao   = null;

        try {
            numero = extraiDados(buffer, inicio);
            inicio += numero.length + (byte)0x01;
            dataEmissao = extraiDados(buffer, inicio);
        }
        catch (Exception e) {
            ISOException.throwIt(CGSStatusWords.CGS_SW_SET_CPF);
        }

        try {
            JCSysytem.beginTransaction();
            this.cpf.setNumero(numero);
            this.cpf.setDataEmissao(dataEmissao);
            JCSysytem.commitTransaction();
        }
        catch (ISOException e) {
            JCSysytem.abortTransaction();
            ISOException.throwIt(e.getReason());
        }
        catch (Exception e1) {
            JCSysytem.abortTransaction();
            ISOException.throwIt(CGSStatusWords.CGS_SW_SET_CPF_TRANSACAO);
        }
        .
        .
    } //fim setCpf
    .
    .
}

```

Quadro 14 – Método *setCpf()* da classe *CGSDocumentosEletronicos*

Com isso tem-se nas variáveis *numero* e *dataEmissao* os dados já preparados para

armazenamento na classe *CGSDocumentosEletronicosCPF*. Para finalizar, é necessário iniciar uma transação através do método *JCSystem.beginTransaction()*, efetuar chamadas aos métodos *setters* das propriedades *numero* e *dataEmissao* e, por fim, finalizar a transação através da chamada do método *JCSystem.commitTransaction()*. Caso algum erro ocorra durante a transação é possível abortá-la através do método *JCSystem.abortTransaction()*.

3.3.5 Enviando dados ao *host*

Para enviar dados é necessário que os *bytes* a serem enviados ao *host* sejam copiados para a variável *buffer* utilizada no método *apdu.getBuffer()* no método *process()*. Para explanação, será utilizada a instrução que envia para o *host* o saldo presente no *applet* responsável pelo gerenciamento do dinheiro eletrônico, demonstrado no Quadro 15.

```

.
.
. public class CGSDinheiroEletronicoApplet extends Applet {
    private CGSDinheiroEletronico dinheiroEletronico;
    final static byte POSICOES_MOEDA = (byte)0x03;
    .
    .
    public void process(APDU apdu) throws ISOException {
        .
        .
        byte[] buffer = apdu.getBuffer();
        .
        .
        case SALDO:
            .
            .
            try {
                dinheiroEletronico.Saldo(buffer);
                apdu.setOutgoing();
                apdu.setOutgoingLength(POSICOES_MOEDA);
                apdu.sendBytes((short)0, (short)POSICOES_MOEDA);
            }
            catch (ISOException e) {
                ISOException.throwIt(CGSStatusWords.CGS_SW_ERRO_SALDO);
            }
            break;
            .
            .
        } //fim process
        .
        .
    } //fim classe

```

Quadro 15 – Enviando dados ao *host*

O método *Saldo()* da classe *CGSDinheiroEletronico* recebe como parâmetro a variável *buffer*, copiando para a mesma, o saldo, conforme especificação nas respostas APDU do *applet CGSDinheiroEletronicoApplet* (seção 3.2.4.1).

A seguir, é chamado o método *setOutgoing()* da classe APDU que informa ao *smart card* que deverá permanecer em modo de envio de dados já que, *smart cards*, operam em modo *half-duplex*, ou seja, num determinado momento ou enviam ou recebem dados, nunca ambos ao mesmo tempo.

Por sua vez, o método *setOutgoingLength(length)*, também da classe APDU, deve ser chamado informando a quantidade de *bytes* que será enviado ao *host*.

Por fim, o método *sendBytes(offset,length)* envia as informações de *buffer* a partir da posição inicial *offset* enviando o número de *bytes* igual a *length*.

3.3.6 Chaves assimétricas e certificado digital

Pelo fato da API Java *card* não possuir classes para manipulação de certificado digital é necessário que o certificado do usuário seja armazenado num vetor de *bytes* no *applet* responsável pelo gerenciamento das chaves assimétricas e certificado digital.

As chaves assimétricas do usuário são criadas no ato da instalação e registro do *applet CGSCertificadoDigitalApplet* no *smart card*, conforme mostrado no Quadro 16.

```

.
.
. public class CGSCertificadoDigitalApplet extends Applet{
    private CGSCertificadoDigital certificadoDigital;
    .
    .
    public static void install(byte[] bArray, short bOffset, byte bLength)
                                throws IOException {
        (new CGSCertificadoDigitalApplet()).register();
    }
    private CGSCertificadoDigitalApplet() {
        super();
        certificadoDigital = new CGSCertificadoDigital();
        .
        .
    }
    .
    .
}

```

Quadro 16 – Método *install()* do applet *CGSCertificadoDigitalApplet*

Conforme mostrado no Quadro 17 é instanciado um objeto de *KeyPair* informando-se o algoritmo e o tamanho das chaves. Neste caso, é utilizado o algoritmo RSA, tendo a chave privada gerada na forma *Chinese Remainder Theorem* (CRT) e o tamanho das chaves é de 512 bits. Em seguida, é invocado o método *genKeyPair()* que gera, efetivamente as chaves.

```

.
.
. public class CGSCertificadoDigital {
    private KeyPair chaves;

    public CGSCertificadoDigital() {
        super();
        chaves = new KeyPair(
            (byte)KeyPair.ALG_RSA_CRT, (short)KeyBuilder.LENGTH_RSA_512);
        chaves.genKeyPair();
    }
    .
    .
}

```

Quadro 17 – Construtor da classe *CGSCertificadoDigital*

Para que a chave pública do usuário seja enviada ao *host*, é extraído desta chave o módulo e expoente. Com estes dois componentes o *host* será capaz de montar novamente a chave pública do usuário.

Conforme Quadro 18, inicialmente são criados dois vetores temporários, *mod* e *exp*, possuindo o tamanho da chave pública. Em seguida, dois *shorts*, *modTamanho* e *expTamanho* recebem respectivamente o tamanho do módulo e expoente, como resultado às chamadas dos métodos *getModulus()* e *getExponent()* da chave pública. Nesse processo, *mod* e *exp* recebem

efetivamente o módulo e expoente desta chave.

Para acomodar os dados que serão enviados ao *host*, é criado um vetor *chavePublica* com tamanho igual à soma do tamanho do módulo e tamanho do expoente. A este valor são somados ainda quatro *bytes* referentes ao tamanho respectivamente, do módulo e expoente (dois *bytes* cada) conforme especificação das respostas APDU do *applet CGSCertificadoDigitalApplet* (seção 3.2.4.4).

Por fim, o método retorna o vetor *chavePublica*, onde são concatenados respectivamente, o tamanho do módulo (2 *bytes*), tamanho do expoente (2 *bytes*), módulo e expoente.

```
.
.
. public class CGSCertificadoDigital {
.
.
. public byte[] getChavePublica() {
.     //cria arrays temporários com tamanho da chave (em bytes)
.     byte[] mod = new byte[(short)KeyBuilder.LENGTH_RSA_512 / (byte)0x08];
.     byte[] exp = new byte[(short)KeyBuilder.LENGTH_RSA_512 / (byte)0x08];
.
.     //extrai módulo
.     short modTamanho =
.         ((RSAPublicKey)chaves.getPublic()).getModulus(mod, (short)0);
.     //extrai expoente
.     short expTamanho =
.         ((RSAPublicKey)chaves.getPublic()).getExponent(exp, (short)0);
.
.     //cria array de bytes para acomodar módulo e expoente
.     byte[] chavePublica = new byte[modTamanho + expTamanho + (byte)0x04];
.
.     Util.setShort(chavePublica, (short)0, modTamanho); //(posições 0 e 1)
.     Util.setShort(chavePublica, (short)2, expTamanho); //(posições 2 e 3)
.
.     Util.arrayCopy(mod, (short)0, chavePublica, (short)0x0004, modTamanho);
.     Util.arrayCopy(exp, (short)0, chavePublica,
.         (short)((short)0x0004 + modTamanho), expTamanho);
.
.     exp = null;
.     mod = null;
.     return chavePublica;
. } //fim getChavePublica
.
. } //fim classe
```

Quadro 18 – Método *getChavePublica()* da classe *CGSCertificadoDigital*

3.3.7 Estudos de caso com JCOP Shell

Para demonstrar o funcionamento dos *applets* desenvolvidos sendo executados no *smart card* real juntamente com o CAD (vistos na Figura 16) será utilizado o componente JCOP Shell presente no *plugin JCOP Tools*.



Figura 16 – Kit *smart card* + CAD utilizado para testes com JCOP Shell

Inicialmente é apresentado um outro componente do *plugin* chamado JCOP Explorer (Figura 17) onde são associados aos *applets* e *packages* seus respectivos AIDs, sendo que os AIDs aqui definidos possuem caráter didático seguindo apenas a regra de possuírem entre 5 e 16 bytes. Esta visão é importante para entendimento posterior do funcionamento do JCOP Shell.

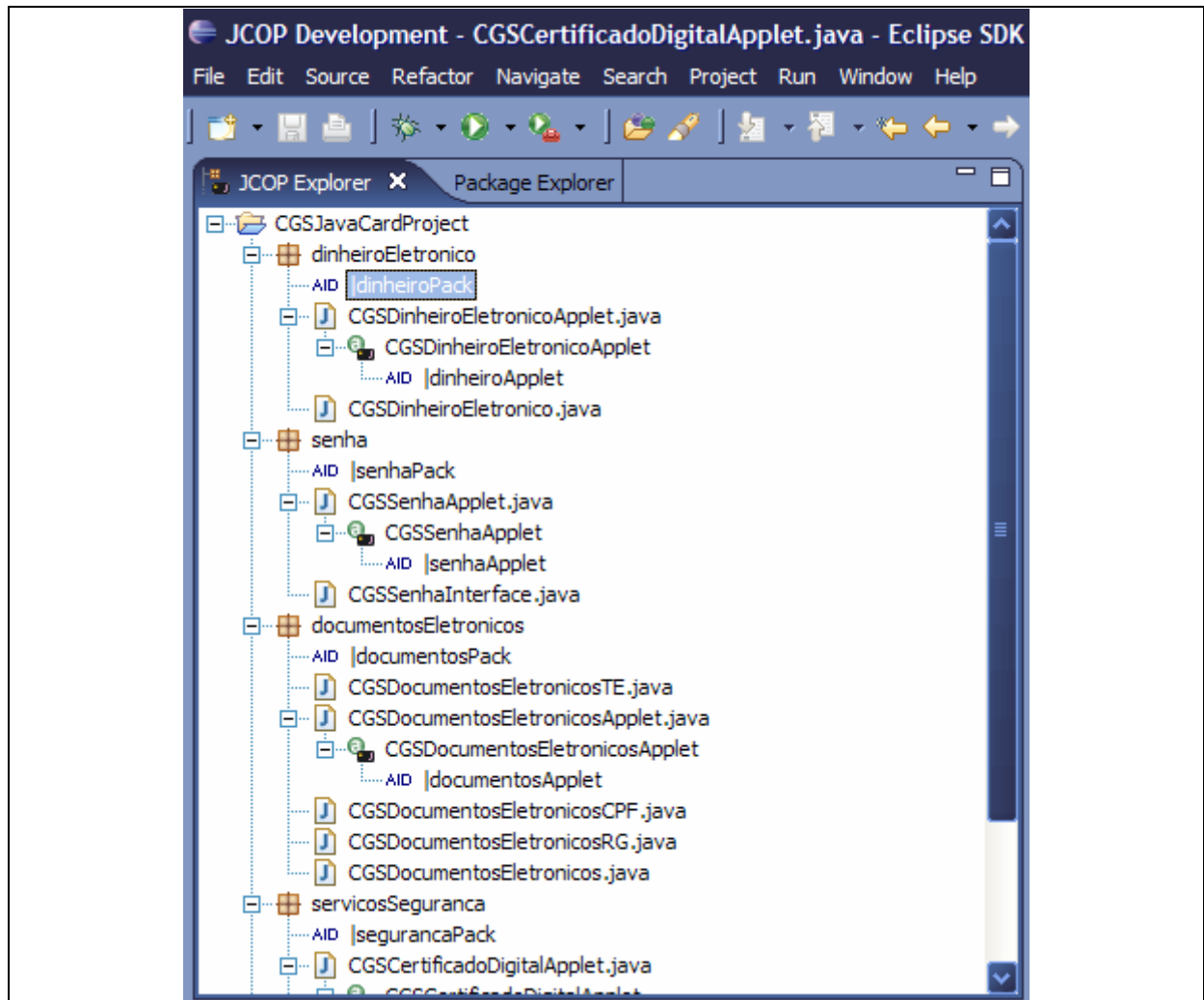


Figura 17 – JCop Explorer

Como exemplo, tem-se no *package dinheiroEletronico* o AID definido como */dinheiroPack*, assim como o *applet CGSDinheiroEletronicoApplet* possui o AID */dinheiroApplet*. O uso do caractere *pipe* (`|`) efetua automaticamente a transformação dos caracteres para hexadecimal quando os *packages* e *applets* forem transferidos para o *smart card*.

Antes de iniciar os testes, os *packages* contendo cada *applet* devem ser transferidos para o *smart card*, para então estarem disponíveis para instalação e registro.

No *JCop Shell*, pode-se verificar após execução do comando *card-info*, a listagem dos *packages* e respectivos *applets* que estão carregados (*LOADED*) no *smart card*, mas ainda não foram instalados e registrados, conforme Figura 18.


```

cm> card-info

Card Manager AID   : A000000003000000
Card Manager state : OP_READY

Load File  :      LOADED (-----) A0000000035350   (Security Domain)
Module    :      A000000003535041
Load File  :      LOADED (-----) "senhaPack"
Module    :      "senhaApplet"
Load File  :      LOADED (-----) "dinheiroPack"
Module    :      "dinheiroApplet"
Load File  :      LOADED (-----) "documentosPack"
Module    :      "documentosApplet"
Load File  :      LOADED (-----) "segurancaPack"
Module    :      "segurancaApplet"

cm> card-info

```

Figura 18 – JCOP Shell: comando *card-info*

Encontrando-se os *packages* e *applets* transferidos para o *smart card*, o processo de instalação de cada *applet* pode ser efetuado, conforme exemplo na Figura 19, que mostra a instalação do *package senha*. A instalação do *applet* (comando *install*) é realizada facilmente através de botão disponível na interface do *plugin*. Nota-se ainda que após execução do comando *card-info*, o *applet senhaApplet* está selecionável (*SELECTABLE*).

```

cm> install -i 73656e68614170706c6574 -q C9#() 73656e68615061636b 73656e68614170706c6574
=> 80 E6 0C 00 28 09 73 65 6E 68 61 50 61 63 6B 0B      ....(.senhaPack.
   73 65 6E 68 61 41 70 70 6C 65 74 0B 73 65 6E 68      senhaApplet.senh
   61 41 70 70 6C 65 74 01 00 02 C9 00 00 00           aApplet.....
(280 msec)
<= 90 00                                               ..
Status: No Error
cm> card-info

Card Manager AID   : A000000003000000
Card Manager state : OP_READY

Application: SELECTABLE (-----) "senhaApplet"
Load File   : LOADED (-----) A0000000035350 (Security Domain)
Module     : A000000003535041
Load File   : LOADED (-----) "senhaPack"
Module     : "senhaApplet"
Load File   : LOADED (-----) "dinheiroPack"
Module     : "dinheiroApplet"
Load File   : LOADED (-----) "documentosPack"
Module     : "documentosApplet"
Load File   : LOADED (-----) "segurancaPack"
Module     : "segurancaApplet"

cm>

```

Figura 19 – JCop Shell: instalação do package senha

A partir do momento que *applets* estão instalados e registrados no *smart card*, é possível a seleção dos mesmos e envio de comandos APDU.

3.3.7.1 Applet CGSSenhaApplet

Para demonstrar as funcionalidades do *applet CGSSenhaApplet*, primeiramente, o *applet* precisa ser selecionado pela JCRE. Para tal, é utilizado o comando `/select /senhaApplet` no JCop Shell. Se nenhum erro ocorrer, o *applet* estará apto a receber comandos APDU que serão verificados e processados no método *process()* do *applet*. Para enviar comandos APDU é utilizado o comando `/send`.

Conforme verificado na Figura 20, após o comando *select*, envia-se um comando APDU para o *applet* indicando que deverá ser gravada a senha (INS 0x01 com senha 0x31

0x32 0x33 representando 123) do usuário no *smart card*. Nota-se que o comando retorna o *status word* 90 00 indicando que não houveram erros.

```

Java Card Bytecode Properties CAP File Properties JCOPI Shell Progress Console
cm> /select |senhaApplet
=> 00 A4 04 00 0B 73 65 6E 68 61 41 70 70 6C 65 74 .....senhaApplet
00
(60 msec)
<= 90 00 ..
Status: No Error
cm> /send 00010000003313233
=> 00 01 00 00 03 31 32 33 .....123
(251 msec)
<= 90 00 ..
Status: No Error
cm>

```

Figura 20 – Applet *CGSSenhaApplet*: seleção e gravação da senha

A seguir, é enviado comando ao *applet* solicitando validação de uma senha que está incorreta (INS 0x03 com senha 0x31 0x32 0x31 que representa 121) com relação à senha cadastrada anteriormente. Este comando será repetido algumas vezes para demonstrar que a senha será bloqueada.

No primeiro comando na Figura 21 é efetuada pela terceira vez a tentativa de validação da senha 121 que está incorreta. Nota-se que o *applet* retornou o *status word* com código 97 96. Se verificado no apêndice D, conclui-se que a senha informada é inválida.

Em seguida, o mesmo comando é novamente submetido ao *applet*, porém, o *status word* retornado possui código 97 97, que indica que a senha foi bloqueada.

```

Java Card Bytecode | Properties | CAP File Properties | JCOP Shell X | Progress | Console
cm> /send 0003000003313231
=> 00 03 00 00 03 31 32 31          .....121
(80 msec)
<= 97 96                             ..
Status: 0x9796
cm> /send 0003000003313231
=> 00 03 00 00 03 31 32 31          .....121
(90 msec)
<= 97 97                             ..
Status: 0x9797
cm>

```

Figura 21 – *Applet CGSSenhaApplet*: validação da senha incorreta e posterior bloqueio

Na Figura 22, o primeiro comando tenta validar uma senha (informada corretamente), porém, pelo fato de estar bloqueada, é retornado o *status word* 97 97. Neste caso, é necessário enviar comando para desbloqueio da senha (INS 0x02). Em seguida a senha é, finalmente, validada.

```

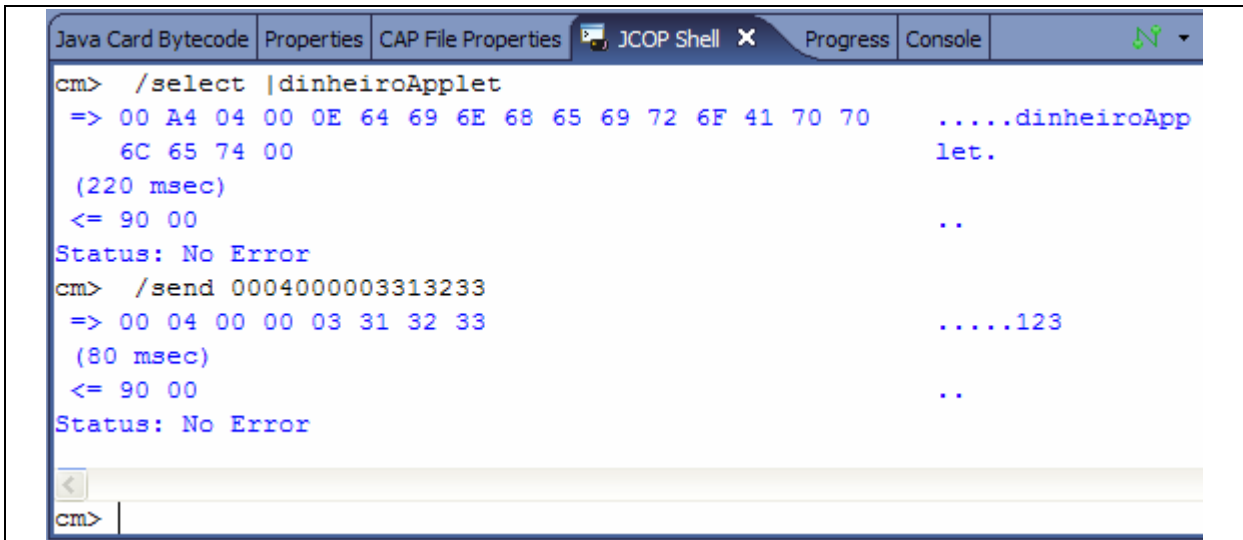
Java Card Bytecode | Properties | CAP File Properties | JCOP Shell X | Progress | Console
cm> /send 0003000003313233
=> 00 03 00 00 03 31 32 33          .....123
(70 msec)
<= 97 97                             ..
Status: 0x9797
cm> /send 0002000000
=> 00 02 00 00 00
(60 msec)
<= 90 00                             ..
Status: No Error
cm> /send 0003000003313233
=> 00 03 00 00 03 31 32 33          .....123
(100 msec)
<= 90 00                             ..
Status: No Error
cm>

```

Figura 22 – *Applet CGSSenhaApplet*: desbloqueio e validação da senha

3.3.7.2 Applet *CGSDinheiroEletronicoApplet*

Depois de selecionar o *applet* responsável pelo gerenciamento do dinheiro eletrônico, para efetuar quaisquer operações é necessária autenticação com a senha anteriormente cadastrada no *applet* *CGSSenhaApplet*. A seleção do *applet* e o comando para validação da senha (INS 0x04) é mostrada na Figura 23.



```

Java Card Bytecode | Properties | CAP File Properties | JCop Shell | Progress | Console
cm> /select |dinheiroApplet
=> 00 A4 04 00 0E 64 69 6E 68 65 69 72 6F 41 70 70 .....dinheiroApp
    6C 65 74 00 .....let.
(220 msec)
<= 90 00 ..
Status: No Error
cm> /send 0004000003313233
=> 00 04 00 00 03 31 32 33 .....123
(80 msec)
<= 90 00 ..
Status: No Error
cm>

```

Figura 23 – *Applet* *CGSDinheiroEletronicoApplet*: seleção do *applet* e validação da senha

Na Figura 24, é feita a tentativa de saque (INS 0x02) no valor de 3,00 unidades de moeda que retorna o código 99 96 (conforme apêndice D) indicando que o saldo é insuficiente. Após executar a instrução para verificar o saldo (INS 0x03), o *applet* retorna os *bytes* 0x00 0x00 0x00, o que mostra que o saldo com relação ao dinheiro eletrônico é zero. Conforme especificação de respostas APDU para o *applet* (seção 3.2.4.1), com relação ao saldo, os dois primeiros *bytes* representam a parte inteira do valor, enquanto o último *byte* representa a parte decimal do valor.

```

Java Card Bytecode Properties CAP File Properties JCOP Shell X Progress Console
cm> /send 0002000003000300
=> 00 02 00 00 03 00 03 00 .....
(70 msec)
<= 99 96 ..
Status: 0x9996
cm> /send 0003000000
=> 00 03 00 00 00 .....
(70 msec)
<= 00 00 00 90 00 .....
Status: No Error
cm>

```

Figura 24 – Applet *CGSDinheiroApplet*: saque e saldo

Na Figura 25, tem-se uma seqüência enumerada de comandos submetidos ao *applet*. No comando 1 é efetuado um depósito no valor de 99,99 (0x00 0x63 0x63) unidades de moeda. A seguir, no comando 2, é verificado o saldo que mostra o valor do depósito realizado. Nos comandos 3 e 4, é mostrado mais um depósito no valor de apenas 0,01 de unidade de moeda e, em seguida, o saldo, que mostra 100,00 unidades de moeda (0x00 0x64 0x00).

No comando 5 é feito um saque no valor de 49,99 (0x00 0x31 0x63) e no comando 6 é mostrado o saldo restante que agora está em 50,01 (0x00 0x32 0x63).

```

Java Card Bytecode Properties CAP File Properties JCOP Shell X Progress Console
cm> /send 0001000003006363 ①
=> 00 01 00 00 03 00 63 63 .....cc
(120 msec)
<= 90 00 ..
Status: No Error
cm> /send 0003000000 ②
=> 00 03 00 00 00 .....
(80 msec)
<= 00 63 63 90 00 .cc..
Status: No Error
cm> /send 0001000003000001 ③
=> 00 01 00 00 03 00 00 01 .....
(110 msec)
<= 90 00 ..
Status: No Error
cm> /send 0003000000 ④
=> 00 03 00 00 00 .....
(80 msec)
<= 00 64 00 90 00 .d...
Status: No Error
cm> /send 0002000003003163 ⑤
=> 00 02 00 00 03 00 31 63 .....1c
(120 msec)
<= 90 00 ..
Status: No Error
cm> /send 0003000000 ⑥
=> 00 03 00 00 00 .....
(60 msec)
<= 00 32 01 90 00 .2...
Status: No Error
cm>

```

Figura 25 – Applet *CGSDinheiroApplet*: depósito, saque e saldo

3.3.7.3 Applet *CGSDocumentosEletronicosApplet*

Assim como no *applet* responsável pelo dinheiro eletrônico, o *applet* de documentos eletrônicos também necessita ser selecionado e a senha para autenticação deve ser validada. Como essas funções foram explanadas anteriormente não serão apresentadas nesta seção.

Para demonstrar as funcionalidades do *applet*, serão armazenadas informações referentes apenas ao documento CPF, visto que os demais documentos possuem funcionalidades semelhantes.

Neste *applet*, inicialmente, devem ser armazenadas as informações comuns a todos os documentos, que compreendem o nome completo e data de nascimento do usuário, conforme mostrado no primeiro comando APDU na Figura 26 que, na seqüência, mostra o comando que efetua a gravação das informações referentes ao documento CPF.

```

Java Card Bytecode Properties CAP File Properties JCOP Shell X Progress Console
cm> /send 0006000021436C656265722047696F76616E6E6920537561766900313630333139383200
=> 00 06 00 00 21 43 6C 65 62 65 72 20 47 69 6F 76      ....!Cleber Giov
    61 6E 6E 69 20 53 75 61 76 69 00 31 36 30 33 31      anni Suavi.16031
    39 38 32 00                                           982.
(250 msec)
<= 90 00                                               ..
Status: No Error
cm> /send 00070000143031323334353637383900303231313230303500
=> 00 07 00 00 14 30 31 32 33 34 35 36 37 38 39 00      .....0123456789.
    30 32 31 31 32 30 30 35 00                          02112005.
(201 msec)
<= 90 00                                               ..
Status: No Error
cm>

```

Figura 26 – *Applet CGSDocumentosEletronicosApplet*: gravação das informações referentes ao CPF

Conforme especificação das respostas APDU (seção 3.2.4.2), ao efetuar a leitura de um documento, o *applet* retornará os dados comuns do usuário do *smart card* concatenados com as informações referentes ao documento solicitado.

Na Figura 27 é apresentado o comando que busca as informações referentes ao documento CPF que foi cadastrado anteriormente. Em seguida é mostrado o comando que busca somente as informações comuns entre todos os documentos.


```

Java Card Bytecode Properties CAP File Properties JCOPI Shell X Progress Console
cm> /send 0002000000
=> 00 02 00 00 00 .....
(541 msec)
<= 43 6C 65 62 65 72 20 47 69 6F 76 61 6E 6E 69 20 Cleber Giovanni
53 75 61 76 69 00 31 36 30 33 31 39 38 32 00 30 Suavi.16031982.0
31 32 33 34 35 36 37 38 39 00 30 32 31 31 32 30 123456789.021120
30 35 00 90 00 05...
Status: No Error
cm> /send 0001000000
=> 00 01 00 00 00 .....
(320 msec)
<= 43 6C 65 62 65 72 20 47 69 6F 76 61 6E 6E 69 20 Cleber Giovanni
53 75 61 76 69 00 31 36 30 33 31 39 38 32 00 90 Suavi.16031982..
00 .
Status: No Error
cm>

```

Figura 27 – Applet *CGSDocumentosEletronicosApplet*: leitura do documento CPF e dos dados comuns entre os documentos

3.3.7.4 Applet *CGSCertificadoDigitalApplet*

Ao ser instalado no *smart card*, o applet *CGSCertificadoDigitalApplet* instancia o par de chaves assimétricas RSA. Na Figura 28 é apresentado o resultado da instrução que busca a chave pública do usuário.

Conforme resposta APDU especificada para o applet (seção 3.2.4.4), os dois primeiros *bytes* representam o tamanho do módulo e os dois *bytes* seguintes representam o expoente da chave pública. Na resposta à execução do comando, nota-se que o módulo possui 64 *bytes* (0x00 0x40) e o expoente possui 3 *bytes* (0x00 0x03).

```

Java Card Bytecode Properties CAP File Properties JCOP Shell X Progress Console
cm> /send 0003000000
=> 00 03 00 00 00          .....
(440 msec)
<= 00 40 00 03 D6 FE D4 9C 45 C5 B7 16 98 E1 62 C4  .@.....E.....b.
    7E 26 E2 A0 C9 04 C2 30 21 BB D4 67 A2 E9 31 E5  ~&.....0!..g..1.
    18 10 01 D6 85 76 FD DF 08 D7 08 98 17 3B 1D 60  ....v.....;.`
    0B 5C A6 62 B2 A4 BC CF 70 DF F6 C8 6D 91 0F 15  .\b....p...m...
    2A 0E 00 B5 01 00 01 90 00          *......
Status: No Error
cm>

```

Figura 28 – Applet *CGSCertificadoDigitalApplet*: envio ao *host* da chave pública do usuário

A chave pública exportada anteriormente poderia ser submetida a uma autoridade certificadora de modo a gerar um certificado digital para o usuário do *smart card*. Esse certificado poderia então, ser importado para o *smart card*. Inicialmente deve ser informado o tamanho, em bytes, do certificado digital, conforme Figura 29. Na seqüência, devem ser enviados ao *applet* fragmentos com até 127 bytes de tamanho até que todo o certificado digital seja transferido por completo para o *smart card*. No exemplo, é demonstrado o envio dos dois primeiros fragmentos do certificado digital.

```

Java Card Bytecode Properties CAP File Properties JCOP Shell x Progress Console
cm> /send 0002000002021B
=> 00 02 00 00 02 02 1B .....
(130 msec)
<= 90 00 ..
Status: No Error
cm> /send 0002007F7F30820217308201C10204435B30D0300D06092A864886F70D0101
=> 00 02 00 7F 7F 30 82 02 17 30 82 01 C1 02 04 43 .....0...0.....C
5B 30 D0 30 0D 06 09 2A 86 48 86 F7 0D 01 01 04 [0.0...*.H.....
05 00 30 81 94 31 0B 30 09 06 03 55 04 06 13 02 ..0..1.0...U....
42 52 31 0B 30 09 06 03 55 04 08 13 02 53 43 31 BR1.0...U....SC1
11 30 0F 06 03 55 04 07 13 08 42 6C 75 6D 65 6E .0...U....Blumen
61 75 31 25 30 23 06 03 55 04 0A 13 1C 43 47 53 au1%0#..U....CGS
20 41 75 74 6F 72 69 64 61 64 65 20 43 65 72 74 Autoridade Cert
69 66 69 63 61 64 6F 72 61 31 25 30 23 06 03 55 ificadora1%0#..U
04 0B 13 1C .....
(301 msec)
<= 90 00 ..
Status: No Error
cm> /send 0002017F7F434753204175746F72696461646520436572746966696361646F
=> 00 02 01 7F 7F 43 47 53 20 41 75 74 6F 72 69 64 .....CGS Autorid
61 64 65 20 43 65 72 74 69 66 69 63 61 64 6F 72 ade Certificador
61 31 17 30 15 06 03 55 04 03 13 0E 43 6C 65 62 a1.0...U....Cleb
65 72 20 47 20 53 75 61 76 69 30 1E 17 0D 30 35 er G Suavi0...05
31 30 32 33 30 36 34 32 32 34 5A 17 0D 30 36 30 1023064224Z..060
31 32 31 30 36 34 32 32 34 5A 30 81 94 31 0B 30 121064224Z0..1.0
09 06 03 55 04 06 13 02 42 52 31 0B 30 09 06 03 ...U....BR1.0...
55 04 08 13 02 53 43 31 11 30 0F 06 03 55 04 07 U....SC1.0...U..
13 08 42 6C ..B1
(300 msec)
<= 90 00 ..
Status: No Error
cm>

```

Figura 29 – Applet *CGSCertificadoDigitalApplet*: comandos para envio do certificado ao *smart card*

Encontrando-se o certificado digital armazenado no *smart card* é possível que o *applet* retorne o certificado digital ao *host*. Desta forma o *host* poderia verificar se o certificado recebido é de uma autoridade certificadora confiável e, assim, estabelecer ou não a troca de informações com o *smart card*.

Da mesma forma que o certificado digital deve ser armazenado em fragmentos, o envio ao *host* também deve ocorrer desta forma, conforme Figura 30, onde o retorno do *applet* informando o tamanho do certificado digital e o envio dos dois primeiros fragmentos é ilustrado.

```

Java Card Bytecode Properties CAP File Properties JCOP Shell x Progress Console
cm> /send 0005000000
=> 00 05 00 00 00 .....
(120 msec)
<= 02 1B 90 00 .....
Status: No Error
cm> /send 0005007F00
=> 00 05 00 7F 00 .....
(510 msec)
<= 30 82 02 17 30 82 01 C1 02 04 43 5B 30 D0 30 0D 0...0.....C[0.0.
06 09 2A 86 48 86 F7 0D 01 01 04 05 00 30 81 94 ..*.H.....0..
31 0B 30 09 06 03 55 04 06 13 02 42 52 31 0B 30 1.0...U....BR1.0
09 06 03 55 04 08 13 02 53 43 31 11 30 0F 06 03 ...U....SC1.0...
55 04 07 13 08 42 6C 75 6D 65 6E 61 75 31 25 30 U....Blumenau1%0
23 06 03 55 04 0A 13 1C 43 47 53 20 41 75 74 6F #..U....CGS Auto
72 69 64 61 64 65 20 43 65 72 74 69 66 69 63 61 ridade Certifica
64 6F 72 61 31 25 30 23 06 03 55 04 0B 13 1C 90 dora1%0#..U.....
00 .
Status: No Error
cm> /send 0005017F00
=> 00 05 01 7F 00 .....
(511 msec)
<= 43 47 53 20 41 75 74 6F 72 69 64 61 64 65 20 43 CGS Autoridade C
65 72 74 69 66 69 63 61 64 6F 72 61 31 17 30 15 ertificadora1.0.
06 03 55 04 03 13 0E 43 6C 65 62 65 72 20 47 20 ..U....Cleber G
53 75 61 76 69 30 1E 17 0D 30 35 31 30 32 33 30 Suavi0...0510230
36 34 32 32 34 5A 17 0D 30 36 30 31 32 31 30 36 64224Z..06012106
34 32 32 34 5A 30 81 94 31 0B 30 09 06 03 55 04 4224Z0..1.0...U.
06 13 02 42 52 31 0B 30 09 06 03 55 04 08 13 02 ...BR1.0...U....
53 43 31 11 30 0F 06 03 55 04 07 13 08 42 6C 90 SC1.0...U....Bl.
00 .
Status: No Error
cm>

```

Figura 30 – *Applet CGSCertificadoDigitalApplet*: respostas do *applet* à solicitação de envio do certificado digital armazenado

3.4 RESULTADOS E DISCUSSÃO

Os trabalhos de Buse (1998) e Paludo (2003) apenas descrevem cenários onde *smart cards* poderiam ser utilizados para gerenciamento e armazenamento de informações, respectivamente, nas áreas de cooperativas médicas e bancária.

Hong e Chun (2001) apresentam o *design* de um cenário utilizando *smart cards* para

aplicações de *e-commerce* utilizando o conceito de dinheiro eletrônico da mesma forma abordada pelo presente material.

A diferença notável do trabalho presente para os trabalhos correlatos apresentados está no fato deste trabalho abordar em detalhes o desenvolvimento dos *applets* para *smart cards* utilizando a tecnologia Java *card*, deixando em segundo plano um cenário abrangente para demonstrar aplicações específicas.

Com relação ao *applet* para gerenciamento de dinheiro eletrônico, com algumas modificações no código e, comandos e respostas APDU, pode-se facilmente utilizar o *applet* para gerenciar, por exemplo, passagens de transporte coletivo ou pontos de um programa de fidelidade de clientes.

A API Java *card* não possui classes para manipulação de certificados digitais e o *applet CGSCertificadoDigitalApplet* (responsável pelo gerenciamento do par de chaves e certificado digital) apenas armazena o certificado do usuário. Com base nestas duas afirmações, com relação ao certificado digital, pode-se visualizar o *smart card* apenas como sendo um *token* que mantém o certificado.

3.4.1 Dificuldades encontradas

O desenvolvimento e testes iniciais dos *applets* foram iniciados no Java card SDK. Porém, como o *kit* não é capaz de simular compartilhamento, cada um dos *applets* responsáveis pelo gerenciamento de dinheiro e documentos eletrônicos mantinha uma senha própria para acesso às informações armazenadas.

Depois de adquiridos o *smart card*, o CAD e o *plugin JCOP Tools*, os testes passaram a serem executados em ambiente real. Assim, foi necessário alterar os *applets* citados acima, implementando o método que busca a referência à interface compartilhada (responsável pela

senha) junto a JCRE. Desta forma, é possível que um único *applet* (*CGSSenhaApplet*) gerencie uma única senha para acesso a todas as informações de todos os *applets* registrados no *smart card*.

Quando um documento eletrônico é armazenado no *smart card* são feitas verificações com relação ao tamanho de determinados campos. Caso o campo não corresponda ao tamanho esperado pela classe que o mantém, o *applet* lança uma exceção com o código do erro.

Para a gravação de um documento, uma transação era aberta no método *process()*, então, o processamento sobre o vetor de *bytes* recebido no comando APDU era efetuado, os dados eram armazenados e, então, a transação era finalizada. Durante esse processamento, os valores dos campos do documento a ser armazenado eram extraídos do vetor e, para tal, algumas variáveis auxiliares eram criadas localmente nos métodos invocados pelo método *process()*.

Porém, se durante a gravação de um campo, o mesmo não correspondesse ao tamanho esperado, seria lançada uma exceção. Neste momento, o *smart card* não retornava o código do erro e não respondia mais a nenhum comando APDU, sendo necessária finalização e inicialização de uma nova sessão CAD.

Para resolver o problema foi necessário iniciar a transação somente no instante em que os campos são gravados no *smart card*. Todo o processamento para extração dos dados do comando APDU deve ser feito fora da transação.

Com isso, conclui-se que todas as operações e variáveis locais criadas durante uma transação serão mantidas pela JCRE internamente, para o caso da transação ser abortada.

4 CONCLUSÕES

O objetivo principal do trabalho que trata do gerenciamento de dinheiro e documentos eletrônicos no *smart card* utilizando a tecnologia Java *card* foi alcançado.

Com relação aos objetivos específicos, todos foram alcançados na íntegra, exceto aquele que diz respeito à autenticação do *host* pelo *smart card* através da análise do certificado digital do *host*. Visto que a API Java *card* não possui classes para manipulação de certificados digitais, tal análise não foi possível.

O IDE Eclipse juntamente com o *plugin J COP Tools* aliados ao *smart card* e CAD com o auxílio do Java *card* SDK, se mostraram como sendo um *kit* completo, robusto, estável e de custo acessível, para o desenvolvimento de aplicações Java *card*.

Por tratar-se de um programa Java sendo executado no *microchip* de um *smart card*, pensa-se num código de alta complexidade, principalmente por tratar-se de programação em baixo nível onde áreas de persistência, entradas e saídas do *microchip* deveriam ser acessadas para efetuar as operações. Porém, como visto, o uso da tecnologia Java *card* abstrai muito desta complexidade.

Para a comunidade de desenvolvedores, este trabalho se apresenta como exemplo para aqueles que tem interesse em iniciar no desenvolvimento utilizando a tecnologia Java *card*.

4.1 EXTENSÕES

Sugere-se como extensão deste trabalho, a implementação do software do *host* que faria o envio de comandos APDU ao *smart card*, simulados neste trabalho com o uso do J COP *Shell*. Para tal é possível fazer uso de uma API chamada *Open Card Framework* (OCF).

Além disto, pode-se efetuar o desenvolvimento de mecanismos de segurança utilizando

técnicas de criptografia de modo que o *smart card* seja capaz de autenticar o *host* utilizando os mecanismos de armazenamento de chaves e certificado digital já presente neste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

ALONSO, Edson E.; MEDEIROS, Igor. Explorando pequenos grandes mundos com java card. **Mundo Java**, Curitiba, n. 12, p. 52-61, 2005.

BUSE, Alibert. **Tecnologia do smart card aplicada em cooperativas médicas**. 1998. 66 f. Monografia (Especialização em Tecnologias em Desenvolvimento de Sistemas) – Universidade Regional de Blumenau, Blumenau.

CERTISIGN. **Identidade digital**: garantindo sua identidade na Internet. [S.l.] 2005. Disponível em: <http://www.certisign.com.br/produtos/id/identidade_digital.jsp>. Acesso em: 31 out. 2005.

CHEN, Ziqun. **Java card technology for smart cards: architecture and programmer's guide**. Massachusetts: Addison Wesley, 2000.

HONG, Insuk; CHUN, Ingook. The implementation of electronic money for e-commerce using Java card. In: INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 2001, Pusan, Coréia. **Proceedings...** Pusan, Coréia: Information and Technology Department, Pusan National University, 2001. p. 1369-1372. Disponível em: <<http://ieeexplore.ieee.org/iel5/7417/20158/00931681.pdf>>. Acesso em: 23 mar. 2005.

LISBÔA, Carlos A. L. **Smart cards**. Porto Alegre, 2003. Disponível em: <<http://www.inf.ufrgs.br/~flavio/ensino/cmp502/SmartCards.ppt>>. Acesso em: 08 mar. 2005.

MEDEIROS, Igor. **Baixe, instale e conheça o kit de desenvolvimento Java card**. [S.l.], 2004a. Disponível em: <www.igormedeiros.com.br>. Acesso em: 26 fev. 2005.

_____. **Escrevendo applets Java card**. [S.l.], 2004b. Disponível em: <www.igormedeiros.com.br>. Acesso em: 26 fev. 2005.

ORTIZ, C. Enrique. **An introduction to Java card technology: part 1**. [S.l.], 2003a. Disponível em: <<http://developers.sun.com/techttopics/mobility/javacard/articles/javacard1/>>. Acesso em: 13 mar. 2005.

_____. **An introduction to Java card technology: part 2, the Java card applet**. [S.l.], 2003b. Disponível em: <<http://developers.sun.com/techttopics/mobility/javacard/articles/javacard2/>>. Acesso em: 13 mar. 2005.

PALUDO, Lauriana. **Um estudo sobre as tecnologias Java de desenvolvimento de aplicações móveis**. 2003. 117 f. Monografia (Especialização em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <<http://www.inf.ufsc.br/~leandro/ensino/esp/monografiaLaurianaPaludo.pdf>>. Acesso em: 08 mar. 2005

PERTO SOFTWARE. **Perto smart**: guia rápido. [Versão 1.0?]. Gravataí, [2005?].

PHILIPS SEMICONDUCTORS. **Smart mx p5ct072**. Versão 1.3. Holanda, 2004. Disponível em : <<http://www.semiconductors.philips.com>>. Acesso em: 25 out. 2005.

SUN MICROSYSTEMS. **Development kit user's guide**. Java card plataform. Version 2.2.1. Santa Clara, Califórnia. 2003a. Documento eletrônico disponibilizado com o Java card software development kit.

_____. **Virtual machine especification**. Java card plataform. Version 2.2.1. Santa Clara, Califórnia. 2003b. Documento eletrônico disponibilizado com o Java card software development kit.

APÊNDICE A – Script JCWDE (*senhaApplet.app*)

Um script JCWDE contém referências aos *applets* que estarão sendo simulados com o Java *card* SDK. Por padrão, a classe *InstallerApplet*, presente no *kit* de desenvolvimento, deve ser a primeira classe referenciada no *script* pois terá a função de instanciar os demais *applets* e, também, emular o registro do JCRE.

Como parâmetro da ferramenta JCWDE, deve ser informado o nome do arquivo contendo o *script*.

O arquivo é formado pela classe do *applet* e respectivo *package* e, também, o identificador do *applet* (AID), que pode ser informado em hexadecimal tendo os *bytes* separados por dois pontos (:) ou então, uma cadeia de caracteres entre aspas (“”). Comentários neste script podem ser efetuados com o uso de duas barras consecutivas (//).

Segue exemplo de script no Quadro 19.

```
//arquivo senhaApplet.app

//applet                               Applet ID (AID)
com.sun.javacard.installer.InstallerApplet 0xa0:0x00:0x00:0x00:0x62:0x03:0x01:0x08:0x01
senha.CGSSenhaApplet                     0x53:0x45:0x4E:0x48:0x41

//SENHA = 0x53 0x45 0x4E 0x48 0x41
```

Quadro 19 – Exemplo de script JCWDE

APÊNDICE B – *Script APDUTool (senhaApplet.src)*

A seguir, no Quadro 20, é demonstrado exemplo de *script* que será lido pela ferramenta APDUTool e submetido ao *applet CGSSenhaApplet*. Neste *script*, cada comando deve encerrar com ponto e vírgula (;) sendo que comentários podem ser feitos utilizando duas barras consecutivas (//).

A ferramenta APDUTool requer alguns argumentos ao ser executada sendo usados, neste exemplo, os seguintes: *senhaApplet.src > saidasenhaApplet.txt*. Esses parâmetros indicam à ferramenta que deve submeter ao *applet CGSSenhaApplet* o arquivo *senhaApplet.src* sendo que o resultado da execução será salvo em um *log* com nome *saidasenhaApplet.txt*.

No *script*, os comandos *powerup* e *powerdown* indicam, respectivamente, o início e fim de uma sessão CAD durante a simulação, entre os quais serão submetidos ao *applet* os comandos APDU.

Após o comando *powerup*, obrigatoriamente, deve ser instanciado o *applet installer* (referente à classe *com.sun.javacard.installer.InstallerApplet*) e, somente a seguir, o *applet CGSSenhaApplet* pode ser instanciado e selecionado para então receber comandos APDU.

```

//arquivo senhaApplet.src

//SENHA = 0x53 0x45 0x4E 0x48 0x41

powerup;

//instancia o applet installer da SUN
0x00 0xA4 0x04 0x00 0x09 0xA0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;

//instancia a classe CGSSenhaApplet
0x80 0xB8 0x00 0x00 0x06 0x05 0x53 0x45 0x4E 0x48 0x41 0x7F;

//efetua seleção do applet CGSSenhaApplet
//[comando de selecao][LC] [          AID          ]
0x00 0xA4 0x04 0x00 0x05 0x53 0x45 0x4E 0x48 0x41 0x7F;

//instrução 1: grava senha informando 123
//[CLA][INS][P1] [P2] [LC] [  senha  ]
0x00 0x01 0x00 0x00 0x03 0x01 0x02 0x03 0x7F;

//instrução 3:
//informa senha incorreta forçando bloqueio da mesma após quarta tentativa
//[CLA][INS][P1] [P2] [LC] [  senha  ]
0x00 0x03 0x00 0x00 0x03 0x03 0x02 0x01 0x7F; //tentativa 1
0x00 0x03 0x00 0x00 0x03 0x03 0x02 0x01 0x7F; //tentativa 2
0x00 0x03 0x00 0x00 0x03 0x03 0x02 0x01 0x7F; //tentativa 3
0x00 0x03 0x00 0x00 0x03 0x03 0x02 0x01 0x7F; //tentativa 4

//instrução 2: desbloqueia senha bloqueada
//[CLA][INS][P1] [P2] [LC]
0x00 0x02 0x00 0x00 0x00 0x7F;

//instrução 3: validar senha informando 123
//[CLA][INS][P1] [P2] [LC] [  senha  ]
0x00 0x03 0x00 0x00 0x03 0x01 0x02 0x03 0x7F;

powerdown;

```

Quadro 20 – Exemplo de *script* APDUTool

APÊNDICE C – Resultado do *script* submetido ao APDUTool (*saidasenhaApplet.txt*)

A seguir, no Quadro 21, é apresentado o resultado do arquivo *senhaApplet.src* submetido ao *applet CGSSenhaApplet* através do APDUTool. Comentários adicionados estão precedidos por duas barras consecutivas (//):

```
//arquivo saidasenhaApplet.txt

//instancia applet installer
CLA:00, INS:a4, P1:04, P2:00,
Lc:09, a0, 00, 00, 00, 62, 03, 01, 08, 01, Le: 00, SW1: 90, SW2: 00

//instancia CGSSenhaApplet
CLA:80, INS:b8, P1:00, P2:00,
Lc:06, 05, 53, 45, 4e, 48, 41, Le:05, 53, 45, 4e, 48, 41, SW1: 90, SW2: 00

//seleciona CGSSenhaApplet
CLA:00, INS:a4, P1:04, P2:00, Lc:05, 53, 45, 4e, 48, 41, Le:00, SW1:90, SW2: 00

//informa a senha - retorno 90 00 - processamento ok
CLA:00, INS:01, P1:00, P2:00, Lc:03, 01, 02, 03, Le:00, SW1:90, SW2:00

//tenta validar senha incorreta (três vezes) - retorno 97 96 - CGS_SW_SENHA_INV
CLA:00, INS:03, P1:00, P2:00, Lc:03, 03, 02, 01, Le:00, SW1:97, SW2:96
CLA:00, INS:03, P1:00, P2:00, Lc:03, 03, 02, 01, Le:00, SW1:97, SW2:96
CLA:00, INS:03, P1:00, P2:00, Lc:03, 03, 02, 01, Le:00, SW1:97, SW2:96

//tenta validar senha incorreta - retorno 97 97 - CGS_SW_BLOQUEADO
CLA:00, INS:03, P1:00, P2:00, Lc:03, 03, 02, 01, Le:00, SW1:97, SW2:97

//desbloqueia a senha bloqueada anteriormente - retorno 90 00 - processamento ok
CLA:00, INS:02, P1:00, P2:00, Lc:00, Le:00, SW1:90, SW2:00

//valida a senha informando senha correta - retorno 90 90 - processamento ok
CLA:00, INS:03, P1:00, P2:00, Lc:03, 01, 02, 03, Le:00, SW1:90, SW2:00
```

Quadro 21 – Resultado do *script* submetido à ferramenta APDUTool

APÊNDICE D – Status Words específicos dos applets desenvolvidos

D.1. Package documentosEletronicos

Descrição do erro	Variável estática	Código
Gravar data de nascimento	CGS_SW_SET_DADOSCOMUNS_DATANASC	0x9899
Tamanho inválido data de nasc.	CGS_SW_SET_DADOSCOMUNS_DATANASC_TAM	0x9898
Leitura data de nascimento	CGS_SW_GET_DADOSCOMUNS_DATANASC	0x9897
Gravar nome completo	CGS_SW_SET_DADOSCOMUNS_NOMECOMP	0x9896
Leitura nome completo	CGS_SW_GET_DADOSCOMUNS_NOMECOMP	0x9895
Função extrair dados	CGS_SW_EXTRAIADOS	0x9894
Erro ao preparar transação	CGS_SW_SET_DADOSCOMUNS	0x9893
Erro ao retornar dados comuns	CGS_SW_GET_DADOSCOMUNS	0x9892
Erro durante transação	CGS_SW_SET_DADOSCOMUNS_TRANSACAO	0x9891

Gravar número de inscrição	CGS_SW_SET_NUMINSCRICAO	0x9849
Gravar divisão	CGS_SW_SET_DIVISAO	0x9848
Gravar zona eleitoral	CGS_SW_SET_ZONA	0x9847
Gravar seção eleitoral	CGS_SW_SET_SECAO	0x9846
Gravar município	CGS_SW_SET_MUNICIPIO	0x9845
Gravar unidade federativa	CGS_SW_SET_UF	0x9844
Gravar emissão do documento	CGS_SW_SET_EMISSAO	0x9843
Leitura número de inscrição	CGS_SW_GET_NUMINSCRICAO	0x9842
Leitura divisão eleitoral	CGS_SW_GET_DIVISAO	0x9841
Leitura zona eleitoral	CGS_SW_GET_ZONA	0x9840
Leitura seção eleitoral	CGS_SW_GET_SECAO	0x9839
Leitura município	CGS_SW_GET_MUNICIPIO	0x9838
Leitura unidade federativa	CGS_SW_GET_UF	0x9837
Leitura emissão do documento	CGS_SW_GET_EMISSAO	0x9836
Tamanho inválido divisão eleitoral	CGS_SW_SET_DIVISAO_TAM	0x9835
Tamanho inválido zona eleitoral	CGS_SW_SET_ZONA_TAM	0x9834
Tamanho inválido emissão	CGS_SW_SET_EMISSAO_TAM	0x9833
Tamanho inválido nº inscrição	CGS_SW_SET_NUMINSCRICAO_TAM	0x9832
Tamanho inválido seção eleitoral	CGS_SW_SET_SECAO_TAM	0x9831
Tamanho inválido unid. federativa	CGS_SW_SET_UF_TAM	0x9830
Erro ao preparar transação	CGS_SW_SET_TITULOELEITOR	0x9829
Leitura Título de Eleitor	CGS_SW_GET_TITULOELEITOR	0x9828
Erro durante transação	CGS_SW_SET_TITULOELEITOR_TRANSACAO	0x9827

Gravar filiação nome Mãe	CGS_SW_SET_FILIACAOMAE	0x9869
Gravar filiação nome Pai	CGS_SW_SET_FILIACAOPAI	0x9868
Gravar naturalidade	CGS_SW_SET_NATURALIDADE	0x9867
Gravar documento de origem	CGS_SW_SET_DOCORIGEM	0x9866
Gravar data de emissão	CGS_SW_SET_DATAEMISSAO_RG	0x9865
Leitura filiação Mãe	CGS_SW_GET_FILIACAOMAE	0x9864
Leitura filiação Pai	CGS_SW_GET_FILIACAOPAI	0x9863
Leitura naturalidade	CGS_SW_GET_NATURALIDADE	0x9862
Leitura documento de origem	CGS_SW_GET_DOCORIGEM	0x9861
Leitura data de emissão	CGS_SW_GET_DATAEMISSAO_RG	0x9860
Erro ao preparar transação	CGS_SW_SET_RG	0x9859
Erro ao retornar dados RG	CGS_SW_GET_RG	0x9858
Erro ao preparar transação RG	CGS_SW_SET_RG_TRANSACAO	0x9857

Gravar número do CPF	CGS_SW_SET_NUMERO	0x9879
Gravar data de emissão CPF	CGS_SW_SET_DATAEMISSAO_CPF	0x9878
Leitura número do CPF	CGS_SW_GET_NUMERO	0x9877
Leitura data de emissão	CGS_SW_GET_DATAEMISSAO_CPF	0x9876
Erro ao preparar transação	CGS_SW_SET_CPF	0x9875
Leitura dados CPF	CGS_SW_GET_CPF	0x9874
Erro durante transação	CGS_SW_SET_CPF_TRANSACAO	0x9873

D.2. Package dinheiroEletronico

Descrição do erro	Variável	Código
Saque	CGS_SW_ERRO_SAQUE	0x9999
Depósito	CGS_SW_ERRO_DEPOSITO	0x9998
Saldo	CGS_SW_ERRO_SALDO	0x9997
Crédito Insuficiente	CGS_SW_CREDITO_INSUF	0x9996
Crédito Máximo Atingido	CGS_SW_CREDITO_MAXIM	0x9995
Decimal Overflow (0..99)	CGS_SW_DECIMAL_OVERF	0x9994

D.3. *Package servicosSeguranca*

Descrição do erro	Variável	Código
Gravar K_U CA durante transação	CGS_SW_SET_CHAVE_PUBLICA_CA_TRANSACAO	0x9699
Gravar K_U CA	CGS_SW_SET_CHAVE_PUBLICA_CA	0x9698
Nº máximo de K_U CA permitidas	CGS_SW_MAX_CHAVES	0x9697
Gravar certificado digital	CGS_SW_SET_CERTIFICADO_DIG	0x9696
Gravar certificado digital durante transação	CGS_SW_SET_CERTIFICADO_DIG_TAMANHO	0x9695
Leitura K_U usuário	CGS_SW_GET_CHAVEPUBLICA_USUARIO	0x9694
Montar K_U usuário para ser enviado ao <i>host</i>	CGS_SW_MONTAR_CHAVEPUBLICA_USUARIO	0x9693
Leitura do certificado digital	CGS_SW_GET_CERTIFICADO_DIG	0x9692
Certificado digital inexistente	CGS_SW_CERT_DIGITAL_INEXISTENTE	0x9691
<i>Host</i> não autêntico	CGS_SW_AUTENTICACAO_HOST	0x9690
Erro ao localizar posição livre no vetor que mantém K_U CAs	CGS_SW_POSICAO_LIVRE	0x9689
Erro ao instanciar nova K_U CA	CGS_SW_CRIAR_CHAVE_PUBLICA_CA	0x9688
Erro ao retornar uma K_U CA	CGS_SW_RETORNAR_CHAVE_PUBLICA_CA	0x9687
K_U CA inexistente	CGS_SW_CHAVE_PUBLICA_CA_EXISTENTE	0x9686
Gravar certificado digital durante transação	CGS_SW_GET_CERTIFICADO_DIG_TAMANHO	0x9685

D.4. *Package senha*

Descrição do erro	estática	Código
Gravar senha do usuário	CGS_SW_GRAVARSENHA	0x9799
Efetuar desbloqueio	CGS_SW_DESBLOQUEIO	0x9798
Erro emitido quando senha estiver bloqueada	CGS_SW_BLOQUEADO	0x9797
Senha informada for inválida	CGS_SW_SENHA_INV	0x9796

D.5. *Status words de uso geral*

Descrição do erro	Variável estática	Código
Erro emitido caso AID não existe na busca de interface <i>shareable</i>	CGS_SW_AID_NAOEXISTE	0x9599
Erro emitido caso Interface <i>CGSSenhaInterface</i> não existe	CGS_SW_INTF_NAOEXISTE	0x9598

APÊNDICE E – Descrição dos casos de uso - *Host*

E.1. *Cadastrar senha pessoal*

Cenário Principal:

- a) *Host* envia comando APDU ao *smart card* solicitando o cadastro da senha pessoal do usuário, contida no comando APDU;
- b) O *smart card* extrai a senha do comando APDU recebido e efetua a gravação retornando *status word* de comando executado com sucesso.

E.2. *Efetua depósito*

Pré-condições:

- a) A senha deve estar validada.

Cenário principal:

- a) O *host* solicita ao *smart card* que efetue o depósito do valor informado no comando APDU;
- b) O *smart card*, por sua vez, soma a quantia informada no comando APDU recebido, referente ao valor do depósito, com a quantidade já existente;
- c) Por fim, o *smart card* retorna *status word* de comando executado com sucesso.

Cenário de exceção:

- a) Saldo máximo alcançado: no item b, caso seja alcançado o saldo máximo permitido será emitido o *status word* CGS_SW_CREDITO_MAXIM.

E.3. Realizar saque

Pré-condições:

- a) A senha deve estar validada.

Cenário principal:

- a) O *host* solicita ao *smart card* que efetue o saque do valor informado no comando APDU;
- b) O *smart card*, por sua vez, subtrai a quantia informada no comando APDU recebido (referente ao valor do saque) da quantidade já existente;
- c) Por fim, o *smart card* retorna *status word* de comando executado com sucesso.

Cenário de exceção:

- a) Saldo insuficiente: no item b, caso não haja um saldo maior ou igual ao valor sacado será emitido o *status word* CGS_SW_CREDITO_INSUF.

E.4. Visualizar Saldo

Pré-condições:

- a) A senha deve estar validada.

Cenário principal:

- a) O *host* solicita ao *smart card* que retorne o valor do saldo existente;
- b) O *smart card*, por sua vez, envia ao *host* resposta APDU contendo o valor do saldo.

E.5. Visualizar documentos eletrônicos

Pré-condições:

- a) A senha deve estar validada.

Cenário principal:

- a) O *host* solicita ao *smart card* que envie os dados de determinado documento eletrônico existente (somente dados do usuário, dados do CPF, título de eleitor ou RG);
- b) O *smart card*, por sua vez, envia resposta APDU ao *host* contendo os dados do documento solicitado.

E.6. Visualizar certificado digital

Cenário principal:

- a) O *host* solicita ao *smart card* o tamanho, em *bytes*, do certificado digital do usuário;
- b) O *smart card* envia resposta APDU ao *host* contendo o tamanho do certificado digital;
- c) O *host* solicita ao *smart card* um fragmento do certificado digital;
- d) O *smart card* envia resposta APDU ao *host* contendo o fragmento solicitado pelo *host*;
- e) Caso o *host* não tenha recebido todos os fragmentos do certificado digital, de modo que o total de *bytes* desses fragmentos seja igual ao tamanho recebido no item b, retorna ao item c.

E.7. Validar senha pessoal

Cenário principal:

- a) O *host* envia comando APDU para o *smart card* com a senha a ser validada, contida no comando APDU;
- b) O *smart card*, por sua vez, valida a senha recebida no comando APDU com a senha previamente cadastrada no caso de uso "Cadastrar senha pessoal", no Apêndice E.1;
- c) Por fim, o *smart card* retorna *status word* de comando executado com sucesso.

Cenários alternativos:

- a) Funções sobre dinheiro eletrônico: após validar a senha (item b), o *host* pode optar por efetuar um depósito ou saque, ou então visualizar o saldo existente, caso o *applet CGSDinheiroEletronicoApplet* esteja selecionado;
- b) Funções sobre documentos eletrônicos: após validar a senha (item b), o *host* pode optar por visualizar os documentos eletrônicos existentes ou efetuar a emissão dos documentos eletrônicos, caso o *applet CGSDocumentosEletronicosApplet* esteja selecionado.

Cenários de exceção:

- a) Senha bloqueada: no item b, caso a senha esteja bloqueada será emitido o *status word* CGS_SW_BLOQUEADO;
- b) Senha inválida: no item b, caso a senha recebida no comando APDU seja diferente da senha previamente cadastrada será emitido o *status word* CGS_SW_SENHA_INV.

APÊNDICE F – Descrição dos casos de uso – *Host* órgão emissor

F.1. *Armazenar certificado digital*

Cenário Principal:

- a) O *host* solicita ao *smart card* a gravação do tamanho, em *bytes*, do certificado digital do usuário;
- b) O *smart card* então, grava o tamanho do certificado digital e retorna *status word* de comando executado com sucesso;
- c) O *host* solicita ao *smart card* a gravação de um fragmento do certificado digital;
- d) O *smart card* grava este fragmento e retorna *status word* de comando executado com sucesso;
- e) Caso o *host* não tenha efetuado a gravação de todos os fragmentos do certificado digital (total de *bytes* dos fragmentos igual ao tamanho informado no item a), retorna ao item c.

F.2. *Armazenar chave pública de CAs confiáveis*

Cenário Principal:

- a) O *host* envia comando APDU ao *smart card* solicitando o cadastro de uma chave pública de uma entidade certificadora;
- b) O *smart card* extrai a chave pública do comando APDU recebido e efetua a gravação da chave;
- c) Por fim, o *smart card* retorna *status word* de comando executado com sucesso.

F.3. Gerar par de chaves para usuário do smart card

Cenário Principal:

- a) O *host* solicita ao *smart card* a instalação do *applet CGSSenhaApplet*;
- b) O *smart card* (JCRE) efetua a instalação do *applet*, que por sua vez, instancia, armazena e inicializa um novo par de chaves.

F.4. Emitir documento CPF

Pré-condições:

- a) A senha deve estar validada.

Cenário principal:

- a) O *host* envia comando APDU ao *smart card* solicitando o cadastro do CPF do usuário;
- b) O *smart card* efetua a gravação do CPF a partir dos dados contidos no comando APDU recebido;
- c) A seguir, *smart card* retorna *status word* de comando executado com sucesso.

Cenários de exceção:

- a) Data de emissão inválida: no item b, caso a data de emissão do documento não possua oito caracteres, será emitido o *status word* CGS_SW_SET_DATAEMISSAO_CPF.

F.5. Emitir documento RG

Pré-condições:

- a) A senha deve estar validada

Cenário principal:

- a) O *host* envia comando APDU ao *smart card* solicitando o cadastro do RG do usuário;
- b) O *smart card* efetua a gravação do RG a partir dos dados contidos no comando APDU recebido;
- c) Por fim, o *smart card* retorna *status word* de comando executado com sucesso.

F.6. Emitir documento Título de Eleitor

Pré-condições:

- a) A senha deve estar validada.

Cenário principal:

- a) O *host* envia comando APDU ao *smart card* solicitando o cadastro do Título de Eleitor do usuário;
- b) O *smart card* efetua a gravação do Título de Eleitor a partir dos dados contidos no comando APDU recebido;
- c) O *smart card* retorna *status word* de comando executado com sucesso.

Cenários de exceção:

- a) Data de emissão inválida: no item b, caso a data de emissão informada não possuir

- oito caracteres, será emitido *status word* CGS_SW_SET_EMISSAO_TAM;
- b) Divisão Inválida: no item b, caso a divisão informada não possuir dois caracteres, será emitido *status word* CGS_SW_SET_DIVISAO_TAM;
 - c) Número de inscrição inválido: no item b, caso o número de inscrição informado não ter nove caracteres, será emitido *status word* CGS_SW_SET_NUMINSCRICAO_TAM;
 - d) Seção eleitoral inválida: no item b, caso a seção eleitoral informada não possuir quatro caracteres, será emitido *status word* CGS_SW_SET_SECAO_TAM;
 - e) Unidade federativa inválida: no item b, caso a unidade federativa informada não ter dois caracteres, será emitido *status word* CGS_SW_SET_UF_TAM;
 - f) Zona eleitoral inválida: no item b, caso a zona eleitoral não possuir três caracteres, será emitido *status word* CGS_SW_SET_ZONA_TAM.

F.7. Validar senha pessoal

Os cenários referentes à validação da senha pessoal podem ser verificados no Apêndice E.7.