

Leonardo Malagoli da Silva

**RIC: UM ESTUDO PARA DESENVOLVIMENTO DE
APLICAÇÕES EMBARCADAS**

Trabalho de conclusão de curso
apresentado como parte dos requisitos
para obtenção do grau de Bacharel em
Sistemas de Informação.

Orientador: Rick Lopes de Souza
Coorientador: Prof. Dr. Ricardo Felipe
Custódio

Florianópolis
2013

Ficha de identificação da obra elaborada pelo autor
através do Programa de Geração Automática da Biblioteca Universitária
da UFSC.

A ficha de identificação é elaborada pelo próprio
autor

Maiores informações em:
<http://portalbu.ufsc.br/ficha>

Leonardo Malagoli da Silva

**RIC: UM ESTUDO PARA DESENVOLVIMENTO DE
APLICAÇÕES EMBARCADAS**

Este Trabalho de conclusão de curso foi julgado adequado para obtenção do Título de “Bacharel em Sistemas de Informação”, e aprovado em sua forma final pelo Curso de Bacharelado em Sistemas de Informação.

Florianópolis, 19 de Junho de 2013.

Prof. Leandro J. Komosinski, Dr.
Coordenador do Curso

Banca Examinadora:

Rick Lopes de Souza
Orientador
Universidade Federal de Santa Catarina

Ricardo Felipe Custódio, Dr.
Coorientador
Universidade Federal de Santa Catarina

Ruy Ramos, Dr.
Instituto Nacional de Tecnologia da Informação

Jean Everson Martina, Dr.
Universidade Federal de Santa Catarina

A minha mãe, minha tia e em especial
ao meu pai, que são a razão de eu ter
chegado até aqui.

AGRADECIMENTOS

A minha família e amigos, que me apoiaram durante todos os momentos difíceis, ao professor Custódio que me deu a oportunidade de aprender e desenvolver este trabalho e ao Rick Lopes de Souza que foi um grande companheiro durante toda a execução deste trabalho, o meu muito obrigado

**“As oportunidades multiplicam-se à medida
que são agarradas”**

Sun Tzu

RESUMO

Atualmente nota-se uma necessidade cada vez maior de interação entre máquinas e humanos, principalmente no quesito identificação. O registro de identidade civil - RIC, proposto pelo governo brasileiro, tem como objetivo facilitar essa interação. Ele torna o processo de identificação ainda mais seguro. Para isso ele contará com um cadastro único, onde todos os registros serão mantidos e contará também com identificação biométrica para impedir que um mesmo titular possua duas identidades. Além disso o RIC será emitido com certificado digital, fazendo com que após a implantação total do projeto, todo brasileiro tenha também uma identidade digital. Este trabalho visa o estudo das tecnologias empregadas no RIC, buscando conhecer e exemplificar o processo de desenvolvimento de aplicações com suporte ao mesmo, seja de forma embarcada, ou mesmo aplicações *host*. Farão parte deste estudo, as formas de distribuição, acesso e decodificação das informações armazenadas no RIC. Também serão estudadas tecnologias aplicáveis ao processo de desenvolvimento de aplicações para execução embarcada.

Palavras-chave: RIC, Certificado Digital, ICP-Brasil, Identificação, Autenticação, Smart card, Java card.

ABSTRACT

Nowadays there is necessity to increase the interaction between machines and humans, mainly in the identification process. The civic identification registry (RIC), proposed by the Brazilian government, has as main objective to ease these interactions. It turns the identification process reliable. To make it possible, it will have integrated records, where it will be stored in a distributed way. It will also have biometric identification, forbidding the same holder of having two identifications. The RIC will be issued with a digital certificates, and after the project implantation, every brasilian will have an electronic identity. This work aims to study RIC's technology, evidencing and exemplifying the development process of embedded or stand alone applications. The study will tackle the data distribution, access and codification stored in RIC. It will also be studied the technologies that could be applied in the application development process to embedded execution.

LISTA DE FIGURAS

Figure 1: Hierarquia PKCS#15 Fonte: [2] pg 10.....	36
Figure 2: Hierarquia de arquivos de um cartão PKCS#15- Fonte: [2] pg 11.....	36
Figure 3: Hierarquia do DF PKCS#15 – Fonte: [2]	37
Figure 4: Exemplo de código OCR-B – Editado de [21].....	45
Figure 5: Indicação visual que um DVLM é um DVLM-e – Fonte : [7].....	46
Figure 6: Anverso do documento de Registro de Identidade Civil - Fonte : [6].....	47
Figure 7: Reverso do documento de Registro de Identidade Civil – Fonte : [6].....	47
Figure 8: Comunicação entre Host e o Cartão durante o registro de Ponto.....	55

LISTA DE TABELAS

Tabela 1: File Identifiers – editado de [2] pgs 18 e 19.....	38
Tabela 2: Fragmento de código com exemplo de uso da classe APDU. 57	
Tabela 3: Fragmento de código com exemplo de uso da classe Util.....	58
Tabela 4: Fragmento de código com exemplo de uso da classe ISO7816	58
Tabela 5: Fragmento de código com exemplo de uso da classe JCSysm	59
Tabela 6: Exemplo de registro armazenado no cartão.....	60
Tabela 7: Exemplo de método de montagem.....	62
Tabela 8: Exemplo de método para envio de comandos ao cartão.....	63
Tabela 9: Fragmento de código para tratamento de comandos recebidos	64
Tabela 10: Fragmento de código para envio de respostas a partir do applet.....	64
Tabela 11: Captura de exceções em Java Card.....	65
Tabela 12: Manipulação de exceções e retorno de erros.....	66
Tabela 13: Verifica se PIN foi validado.....	67
Tabela 14: Inicialização e geração da chave simétrica.....	69
Tabela 15: Inicialização da cifra no modo cifragem simétrica.....	69
Tabela 16: Inicialização da cifra no modo decifragem simétrica.....	69
Tabela 17: Fragmento de código para cifrar um array de bytes com o algoritmo DES.....	70
Tabela 18: Inicialização e geração das chaves públicas e privadas.....	72
Tabela 19: Inicialização da cifra em modo cifragem assimétrica.....	73
Tabela 20: Inicialização da cifra em modo decifragem assimétrica.....	73

Tabela 21: Inicialização do objeto Cipher com algoritmo RSA.....	73
Tabela 22: Fragmento de código para cifrar um array de bytes com algoritmo RSA.....	74
Tabela 23: Inicialização do objeto para cálculo de hash.....	75
Tabela 24: Fragmento de código para cálculo de hash.....	75
Tabela 25: Inicialização de um objeto para assinar dados.....	76
Tabela 26: Fragmento de código para assinatura de dados.....	76

LISTA DE ABREVIATURAS E SIGLAS

RIC - Registro de Identidade Civil
ICAO – *International Civil Aviation Organization*
PKCS#15 – *Public-Key Cryptography Standards : Cryptographic Token Information Format Standard*
LabSEC - Laboratório de Segurança em Computação
DOV – Dispositivo Opticamente Variável
ITI- Instituto Nacional de Tecnologia da Informação
ISO – *International Organization for Standardization*
ICP-Brasil – Infraestrutura de chaves públicas brasileira
IBM – *International Business Machines*
ROM – *Read-Only Memory*
RAM – *Random Access Memory*
EEPROM – *Electrically-Erasable Programmable Read-Only Memory*
JCVM - *Java Card Virtual Machine*
JCRE – *Java Card Runtime*
API – *Application Programming Interface*
APDU – *Application Protocol Data Unit*
AID – *Applet Identifier*
CAD – *Card Acceptance Device*
PC/SC – *Personal Computer/Smart Card*
PKCS - *Public-Key Cryptography Standards*
CI - Circuito integrado
PIN – *Personal Identification Number*
EF- *Elementary File*
PrKDFs - *Private Key Directory Files*
PuKDFs - *Public Key Directory Files*
SKDFs - *Secret Key Directory Files*
CDFs - *Certificate Directory Files*
DODFs - *Data Object Directory Files*
AODFs - *Authentication Object Directory Files*
RID – *Registered application provider identifier*
FIPS - *Federal Information Processing Standards Publication*
NIST - *National Institute of Standards and Technology*
DVLM – Documento de Viagem de leitura Mecânica
ZLM - Zona de leitura mecânica
ZIV - Zona de inspeção Visual
DNle - Documento Nacional de *Identidad electrónico*
EID – *Eletronic Identity*

PUK – *PIN Unlock Key*

REP – Registro Eletrônico de Ponto

NSR – Número sequencial de Registro

JSE - *Java Standard Edition*

SUMÁRIO

1 INTRODUÇÃO.....	23
1.1 OBJETIVO GERAL.....	24
1.2 OBJETIVOS ESPECÍFICOS.....	24
1.3 JUSTIFICATIVA.....	24
1.4 METODOLOGIA.....	25
1.5 PREMISSAS DE DESENVOLVIMENTO.....	26
2 FUNDAMENTAÇÃO TEÓRICA.....	27
2.1 RIC.....	27
2.1.1 Leis, Decretos e Resoluções.....	28
2.1.1.1 Lei Nº 9454 de 7 de Abril de 1997.....	28
2.1.1.2 Lei Nº 12058, de 13 de Outubro de 2009.....	28
2.1.1.3 Decreto Nº 7166, de 5 de Maio de 2010.....	29
2.1.1.4 Resolução MJ Nº 2, de 10 de Setembro de 2010.....	29
2.1.1.5 Projeto de Lei 3860/12.....	30
2.2 SMART CARD	30
2.2.1 Tecnologias.....	31
2.2.1.1 Java Card.....	31
2.2.2 Normas, Especificações e padrões.....	34
2.2.2.1 GlobalPlatform.....	34
2.2.2.2 Especificação PC/SC.....	34
2.2.2.3 PKCS#15.....	35
2.2.2.4 ISO/IEC 7816.....	38
2.2.2.5 Federal Information Processing Standards Publication - FIPS 140.....	39
2.3 CRIPTOGRAFIA.....	40

2.3.1 Criptografia simétrica.....	40
2.3.2 Criptografia assimétrica.....	40
2.3.3 Funções Hash ou Funções de resumo.....	41
2.3.4 Assinatura digital.....	42
2.4 AUTENTICAÇÃO E IDENTIFICAÇÃO BIOMÉTRICA.....	42
2.4.1 Reconhecimento Facial.....	43
2.4.2 Impressão digital.....	43
2.4.3 Leitura da Íris.....	44
2.5 CÓDIGOS OCR-B	44
2.5.1 Dados da ZLM.....	45
2.6 DOCUMENTO DE VIAGEM COM CAPACIDADE DE IDENTIFICAÇÃO BIOMÉTRICA.....	45
2.6.1 Dados e sua estrutura.....	48
2.7 DOCUMENTOS DE IDENTIDADE ELETRÔNICOS NO MUNDO.....	49
2.7.1 Cartão cidadão – Portugal.....	49
2.7.2 Documento Nacional de Identidad electrónico – Espanha.....	50
2.7.3 EID – Bélgica.....	50
2.7.4 Estonian identity card – Estônia.....	50
3 DESENVOLVIMENTO DE APLICAÇÕES EMBARCADAS.....	52
3.1 CHIP COM CONTATO.....	52
3.1.1 Capacidade de armazenamento.....	52
3.1.2 Homologação na ICP-BRASIL.....	53
3.2 CHIP SEM CONTATO.....	53
3.3 EXPERIMENTOS.....	53
3.3.1 Objetivos específicos.....	55

3.3.2	Escopo.....	56
3.3.3	Etapas do desenvolvimento de um applet Java Card....	56
3.3.4	Manipulação de dados em java card(cópia de arrays, tipos, operações, API e etc.....)	57
3.3.5	Armazenar dados no cartão.....	59
3.3.6	Comunicação entre applet java card e aplicação host;. 60	
3.3.6.1	Lado Host ou Desktop.....	61
3.3.6.2	Lado Applet.....	63
3.3.7	Tratamento de exceções.....	65
3.3.8	Controle de acesso e privilégios através de PIN e PUK.	66
3.3.9	Criptografia simétrica de dados em java card;.....	68
3.3.10	Criptografia assimétrica de dados em java card;.....	71
3.3.11	Outras operações criptográficas.....	75
4	TRABALHOS FUTUROS.....	77
5	CONCLUSÃO.....	79
6	REFERÊNCIA.....	81
7	ANEXO A.....	85
8	ANEXO B - CÓDIGO FONTE- CLASSE CORE.....	87
9	ANEXO C- CÓDIGO FONTE – CLASSE DADOS PESSOAIS	101
10	ANEXO D – CÓDIGO FONTE – CLASSE FUNCOESCRIPTOGRAFICASAUXILIARES.....	104
11	ANEXO E – CÓDIGO FONTE – CLASSE GUARDIAODACHAVE.....	106
12	ANEXO F – CÓDIGO FONTE - PERSISTENCIA.....	110
13	ANEXO G- CÓDIGO FONTE – CLASSE CIFRA.....	112
14	ANEXO H - ARTIGO.....	129

1 INTRODUÇÃO

Na atual geração nota-se uma necessidade cada vez maior de interação entre máquinas e humanos, principalmente nos quesitos identificação e autenticação. Neste contexto foi lançado pelo governo brasileiro o Registro de Identidade Civil – RIC, o qual além de acompanhar as tendências mundias de documentos do gênero, supri esta demanda através de certificados digitais emitidos junto ao novo documento. Para implantar o RIC foi criado o Sistema Nacional de Registro de Identificação Civil que tem como finalidade além da implantação do RIC, a implantação do cadastro nacional de registro de identidade civil. O RIC é um número único baseado nas impressões digitais de cada indivíduo, que o identificará em todas as suas relações com a sociedade. O RIC contará com dois chips, além de um código OCR-B. O chip sem contato e o código OCR-B, permitirão que o RIC funcione como um documento de viagem, seguindo o padrão definido pela *International Civil Aviation Organization - ICAO*, já o chip com contato, terá suporte a multi-aplicação. O RIC será emitido com certificado digital, tornando o processo de identificação pessoal mais rápido e seguro, além de permitir também a assinatura de documentos, seja presencialmente ou em comunicações através da internet. Os avanços tecnológicos implantados no RIC, colocam o Brasil na vanguarda da identificação civil.¹

O RIC acompanha a evolução nas formas de relacionamento entre as pessoas na sociedade, preocupando-se com outras formas de relacionamento diferentes da presencial. Um exemplo disto é o certificado digital emitido no cartão, o qual abre a possibilidade da digitalização de vários processos do governo, ganhando agilidade e confiabilidade.

Além das novas formas de identificação suportadas pelo RIC, outro ponto de grande expressividade do projeto é o cadastro único para o país inteiro. Atualmente cada estado possui um sistema isolado dos demais, tornando possível a emissão de mais de uma identidade a uma mesma pessoa. O registro único, aliado com o reconhecimento através da impressão digital, torna praticamente impossível um mesmo cidadão emitir duas identidades.

O objetivo desde projeto é estudar a tecnologia utilizada no RIC, com intuito de conhecer características e limitações de *hardware* e *software*. Ao final do projeto espera-se ter uma clara visão do processo de desenvolvimento de aplicações embarcadas no RIC (características,

capacidades, limitações e obrigações), assim como também, a forma de acesso as informações disponibilizadas no RIC. Estas informações serão importantes na construção de aplicações com suporte ao RIC no futuro.

1.1 OBJETIVO GERAL

Conhecer as tecnologias empregadas no RIC, com intuito de implementar e descrever o processo de desenvolvimento de aplicações para execução embarcada, ou com suporte ao RIC.

1.2 OBJETIVOS ESPECÍFICOS

- Descrever características de segurança do RIC;
- Identificar questões legais sobre a execução de aplicações embarcadas, e uso das informações disponíveis no RIC;
- Identificar limitações de *hardware* e *software* do RIC;
- Identificar, testar e avaliar tecnologias de *Frameworks*, *Middleware* e bibliotecas existentes para utilização no/com RIC;
- Implementar e descrever o processo de desenvolvimento de aplicações embarcadas no RIC;
- Identificar possíveis pontos que podem ser melhorados no projeto do RIC.
- Realizar eventuais críticas ao projeto;

1.3 JUSTIFICATIVA

O novo documento de identificação civil brasileiro, denominado RIC, trás consigo grandes avanços tecnológicos quando comparado aos atuais documentos nacionais. Estes avanços permitem entre outras coisas, que o preenchimento de um formulário de cadastro seja feito de forma quase instantânea, identificar-se e ou autenticar-se de várias formas, inclusive através da rede, possibilita também integrar vários números de documentos em um mesmo suporte documental, por

exemplo, os números de RG e CPF impressos no suporte documental do RIC.

Com a emissão do RIC, surgirá uma demanda crescente por sistemas capazes de lidar com esta tecnologia. Esta demanda poderá vir não apenas da administração pública onde vários processos poderão ser digitalizados, mais também da iniciativa privada, visto que com o RIC processos diários como identificar-se, ganham agilidade e confiabilidade. No entanto, para que os avanços trazidos pelo RIC tornem-se viáveis na prática, é preciso saber como interagir com os mesmos, por exemplo: saber como acessar as informações armazenadas no RIC, quais regras devem ser satisfeitas, quais normas ou especificações ajudam neste processo e qual a melhor interface de acesso para determinada situação.

Estas são perguntas básicas para o sucesso do RIC e neste trabalho, buscam-se respostas para as mesmas.

1.4 METODOLOGIA

A execução deste trabalho está dividida em três etapas principais. Na primeira, será realizada uma pesquisa bibliográfica sobre as legislações, normalizações e tecnologias empregadas ou relacionais ao RIC. Esta pesquisa visa apresentar uma visão introdutória do RIC e servir como base para trabalhos futuros realizados sobre o tema. A pesquisa será realizada em especificações, normas, recomendações, artigos e manuais aplicáveis ao RIC.

Na etapa dois, serão realizados experimentos práticos utilizando cartões de testes, ajudando a demonstrar e validar as teorias levantadas na primeira fase. Nesta etapa, serão testadas tecnologias de *Frameworks*, *Middleware*, bibliotecas e outras que possam ser incorporadas ao RIC, seja para execução embarcada, ou mesmo para facilitar a manipulação do mesmo.

Por fim, será desenvolvida uma aplicação modelo e um relatório das atividades de pesquisa, testes, desenvolvimento e demais realizadas ao longo desde trabalho.

1.5 PREMISSAS DE DESENVOLVIMENTO

Para a realização deste trabalho, assume-se como premissa do novo documento, que o mesmo estará de acordo com o padrão PKCS#15 ². Assumi-se isto, visto que grande maioria dos cartões existentes no mercado possuem tal característica, e a mesma é considerada uma boa prática entre os desenvolvedores.

Outra premissa fundamental para execução deste trabalho, é que o chip de contato do RIC, o qual possui suporte para multi-aplicação, possuirá também suporte a tecnologia *Java Card*. O suporte a esta tecnologia é a única forma de garantir-se que um *applet* desenvolvido para o RIC, funcionará em todos os RICs de todos os estados do país. Isto se deve ao fato de que cada estado é livre para realizar suas próprias concorrências para escolha de empresas fornecedoras de cartões. Com isto, em diferentes estados, diferentes fabricantes de cartões podem ser adotados, com diferentes cartões, haverão conseqüentemente diferentes sistemas operacionais.

Estas premissas foram assumidas após análise dos cartões de testes existentes no Laboratório de Segurança em Computação - LabSEC, onde praticamente todos possuíam tais características.

2 FUNDAMENTAÇÃO TEÓRICA

Conforme metodologia proposta, esta seção do trabalho apresenta um estudo sobre as principais tecnologias e legislações aplicáveis ao RIC, ela poderá servir de base para outros trabalhos sobre o tema, além de fundamentar a etapa de experimentos.

2.1 RIC

O Registro de Identidade Civil - RIC é um novo documento de identificação civil brasileiro, criado pela lei Nº 9.454 de 7 de Abril de 1997 ³, a qual foi ajustada pela Lei nº 12058, de 13 de Outubro de 2009 ⁴. Em 5 de maio de 2010, através do decreto nº 7.166 ⁵ foi criado o Sistema Nacional de Registro de Identidade Civil com a finalidade de implantar o número único do Registro de Identidade Civil – RIC e o Cadastro Nacional de Registro de Identificação Civil. Apesar da sua criação em 1997, somente em 2010, através da resolução nº 2 de 10 de Setembro de 2010 ⁶, o Ministério da Justiça publicou as especificações técnicas do novo documento.

Como especificado na resolução nº2 de 10 de Setembro de 2010 do Ministério da Justiça ⁶, o novo documento terá dois chips, um sem contato (o qual fará do RIC um documento de viagem de leitura mecânica, padrão ICAO ⁷) e o segundo, com contato, para questões de *match-on-card* e suporte a multi-aplicações. Outro avanço trazido pelo RIC, está justamente no chip de contato, o qual será emitido com um certificado digital do titular do cartão, que permite ao mesmo se autenticar também no ambiente digital. Ainda no chip de contato do RIC, estarão as imagens de 4 digitais do titular, permitindo também a autenticação biométrica do mesmo.⁶

Entre os avanços tecnológicos implantados no RIC, além dos chips embarcados, podemos destacar⁶:

- Imagens combinadas(MLI) gravadas a laser da foto, número RIC, assinatura do titular e unidade da federação do órgão emissor;
- Código OCR-B (MRZ), Os principais dados do titular, são armazenados na forma de código OCR-B no verso do cartão, permitindo assim, agilidade no processo de identificação em portos, aeroportos e fronteiras;

- Material do cartão, é especialmente preparado para o processo de gravação a laser, o que proporciona, maior qualidade gráfica e dificulta sua falsificação;

- DOV(dispositivo óptico variável), transparente e metalizado, produz efeito de transição de formas e cores, desenvolvido exclusivamente para o RIC;

- Foto fantasma, replicação da foto do titular, também gravada a laser. Representa mais um item de segurança do documento;

- Número de série do cartão, todo suporte documental é numerado e vinculado ao cadastro do titular;

2.1.1 Leis, Decretos e Resoluções

Esta seção apresenta de forma resumida todas as leis, decretos e normativos relacionados ao RIC. Sua principal função, é dar embasamento para questões legais envolvendo o novo documento.

2.1.1.1 Lei Nº 9454 de 7 de Abril de 1997

A Lei nº 9.454 de 7 de Abril de 1997 ³, institui o número único de Registro de Identidade Civil, número pelo qual todo cidadão Brasileiro, nato ou naturalizado, passa a ser identificado nas suas relações com a sociedade. Institui também o Cadastro Nacional de Registro de Identidade Civil, destinado a conter o número único de Registro de Identidade Civil, acompanhado dos dados de identificação de cada cidadão³.

2.1.1.2 Lei Nº 12058, de 13 de Outubro de 2009.

A lei nº 12058, de 13 de Outubro de 2009 ⁴, altera a Lei 9.454 de 7 de Abril de 1997³, autorizando a União a firmar convênios com Estados e o Distrito Federal para implantação do número único de registro de identidade Civil, definindo responsabilidades aos Estados, Distrito Federal e ao órgão central.⁴

2.1.1.3 Decreto Nº 7166, de 5 de Maio de 2010.

O decreto nº 7.166, de 5 de Maio de 2010 ⁵ cria o Sistema Nacional de Registro de Identidade Civil e o comitê gestor, define sua composição e suas responsabilidades. Neste decreto, é atribuído ao comitê gestor, entre outras responsabilidades, a de “definir as especificações do Cadastro Nacional de Registro de Identidade Civil e do documento de identificação a ser emitido com o RIC” ⁵. O sistema Nacional de Registro de Identidade Civil, tem a finalidade de implementar o número único do Registro de Identidade Civil – RIC e o Cadastro Nacional de Registro de Identificação Civil.⁵

Participarão do comitê: o Ministério da Justiça (Órgão Central), Ministério da Defesa, Ministério da Fazenda, Ministério do Planejamento, Orçamento e Gestão, Ministério do Trabalho e Emprego, Ministério da Previdência Social, Ministério do Desenvolvimento Social e Combate à Fome, Ministério da Saúde, Ministério das Cidades, Ministério do Desenvolvimento Agrário, Secretaria de Direitos Humanos da Presidência da República, Casa Civil da Presidência da República e Instituto Nacional de Tecnologia da Informação – ITI. ⁵

2.1.1.4 Resolução MJ Nº 2, de 10 de Setembro de 2010.

A resolução nº2 do ministério da justiça, de 10 de Setembro de 2010 ⁶ dispõe sobre as especificações técnicas básicas do suporte documental do número único de Registro de Identidade Civil. Entre os requisitos do documento ⁶, vale destacar:

- O cartão utilizado como suporte documental para o novo documento, deverá atender as normas internacionais para documentos similares, em especial as ISO 1073-2 e 1831(reconhecimento óptico de caracteres), ISO 7810 (características físicas do cartão), e Documento 9303 da ICAO (documentos de viagem de leitura mecânica).

- Cada um dos chips deverá ter capacidade mínima de armazenamento de 64KB;

- O sistema cartão/chip deve possuir homologação da ICP-Brasil para questões de certificado digital;

2.1.1.5 Projeto de Lei 3860/12

O projeto de lei nº3860/12 ⁸ altera a lei 9454/97 ³ que institui o número único de Registro de Identidade Civil. O projeto entre outras providências, estabelece a utilização de número único sequenciado para o registro de identidade civil. Este projeto, segundo Gilmar Machado, visa fomentar a entrada em funcionamento do cadastro, oferecendo diretrizes para sua organização.⁹

2.2 SMART CARD

Smart Card ou cartão de circuito integrado é definido na especificação PKCS#15² página 3 como: “are intrinsically secure computing platforms ideally suited to providing enhanced security and privacy functionality to applications”² em livre tradução fica: “plataformas de computação intrinsecamente seguras ideais para proporcionar maior segurança e funcionalidade de privacidade para aplicações”

Smart card é um cartão, geralmente do tamanho de um cartão de crédito, com funções de processamento e/ou armazenamento. A arquitetura de um *smart card* é muito semelhante a arquitetura de um computador IBM, composta por um microprocessador, memória ROM, memória RAM e memória EEPROM. Da mesma forma como um computador pessoal, um *smart card* também possui um sistema operacional, além é claro de dispositivos de entrada e saída de dados.¹⁰

Em geral, a maior parte dos *smart cards* disponíveis no mercado atendem a ISO 7816, a qual especifica questões como tamanho, características elétricas, arquitetura de *hardware*, estrutura lógica, identificação de aplicações e etc....

O *hardware* do cartão é bastante limitado, contorna-se isso através da divisão do processamento entre o cartão e o *host*(computador ou terminal no qual o leitor de cartão está conectado).¹⁰

Existem diversas tecnologias para o desenvolvimento de aplicações com suporte ao *Smart card*.

2.2.1 Tecnologias

Esta seção do trabalho, apresenta as tecnologias ligadas a *smart cards* e dispositivos de segurança. Seu objetivo é fornecer conceitos básicos para o desenvolvimento do trabalho.

2.2.1.1 Java Card

Java Card é um *Smart Card* que possui uma *Java Card Virtual Machine* - JCVM e ou uma *Java Card Runtime* - JCRE, em outras palavras, é um cartão capaz de executar códigos contendo um subconjunto da linguagem *Java*. A plataforma *Java Card* segue as especificações da Sun. São três especificações que regem uma plataforma *Java Card*. A especificação da JCVM, da JCRE e a especificação da API *Java Card*.

O uso crescente de aplicações *Java Card* é justificado em parte pela segurança oferecida por esta plataforma. A plataforma *Java Card* oferece recursos bastante interessantes visando proteger aplicações *Java Card* contra possíveis erros de desenvolvimento, ou mesmo tentativas de fraude. O principal elemento para a segurança na plataforma *Java Card*, é o *Firewall Java Card*, o qual é apresentado com mais detalhes a seguir neste trabalho.¹¹

Outra grande vantagem da plataforma *Java Card*, está na facilidade de desenvolvimento de aplicações. É relativamente fácil para um programador *Java*, desenvolver aplicações em *Java Card*. Outro ponto facilitador da plataforma, está na abstração do *hardware* no qual a aplicação está instalada. Com *Java Card*, detalhes de *hardware* são abstraídos e o desenvolvedor não precisa mais ter um conhecimento tão profundo do *hardware*, como acontece com alguns concorrentes.¹¹

Contexto

Cada *applet* executando na máquina virtual *Java Card*, está contido em um contexto. Em *Java Card*, o contexto é determinado pelo pacote onde o *applet* foi definido. A máquina virtual *Java Card* utiliza o contexto para aplicar políticas de segurança entre *applets*. O *Java Card Runtime Environment* tem seu próprio contexto.¹¹

Tempo de vida de um *Applet*

O ciclo de vida de um *applet* inicia quando o mesmo é corretamente carregado no cartão e de alguma forma tem sua execução preparada. Em geral a preparação do *applet* é realizada no método denominado *install*, este método é chamado pelo JCRE quando o *applet* é carregado no cartão. O método *install* deve fazer todas as preparações (alocações de memória, criação de objetos e etc) necessárias para a execução do *applet*, e após as preparações, deve-se chamar o método *static* da classe *Applet register*. Após esta fase, o *applet* existe no cartão.¹²

A partir do momento que um *applet* é selecionado, todos os APDU recebidos pelo JCRE são encaminhados para o *applet* em execução através do seu método *process*, o qual recebe como parâmetro uma instância da classe APDU com a APDU recebida pela JCRE.¹²

Quando o JCRE recebe um APDU *select* com um AID diferente do AID do *applet* em execução, o JCRE chama o método *deselect* do *applet* atualmente em execução, e depois chama o método *select* do *applet* referente ao APDU recebido. Isto permite ao *applet* em execução, executar quaisquer limpezas necessárias.¹²

Objetos Transitórios

A plataforma *Java Card* não suporta a palavra-chave *transient*, no entanto, caso o *applet* necessite de um dado temporário, ou seja, que não são persistidos em sessões CAD, *Java Card* oferece métodos para criar matrizes temporárias com componentes primitivos ou referência à objetos.¹²

Por questões de segurança, campos de objetos temporários não devem ser armazenados em tecnologias de memória persistente.¹²

Existem dois tipos de objetos *transient*, os *CLEAR_ON_RESET* e os *CLEAR_ON_DESELECT*. *CLEAR_ON_RESET* são objetos *transient* que mantêm os dados armazenados entre seleções de *applet*, porém, no *reset* do cartão, estes dados são limpos. *CLEAR_ON_DESELECT* também são objetos *transient*, porém não mantêm os dados armazenados entre seleções de *applet*. Da mesma forma como *CLEAR_ON_RESET*, no *reset* do cartão, os dados dos objetos *CLEAR_ON_DESELECT* também são limpos.¹²

Isolamento de *Applet* e Objeto compartilhado

O JCRE deve garantir o isolamento de contexto e isolamento de *applet*, ou seja, um *applet* não pode acessar dados de outro, a menos que este segundo *applet*, forneça uma interface de acesso.¹²

Applet Firewall

O *applet Firewall* é a proteção em tempo de execução. O *Firewall* divide a plataforma em espaços protegidos denominados contexto. Cada pacote *java card* tem seu próprio contexto, e o *Firewall* é o limite entre um e outro contexto. Duas instâncias da mesma classe, compartilham um mesmo contexto, pois em *Java Card*, a granularidade dos binários está no pacote, então todos os *applets* definidos dentro do mesmo pacote, compartilham contexto.¹²

Objetos só podem ser acessados por outros objetos do mesmo contexto, quaisquer tentativa de acesso por parte de objetos de contexto diferente, será negada pelo *Firewall*, e será lançada uma exceção *SecurityException*.¹²

Em *java card*, as instâncias de pacotes estão inseridas em contextos, e não a classe em si, ou seja, ao acessar métodos *static* ou atributos *static*, não há verificação ou troca de contexto.¹²

O *applet Firewall* também mantém um contexto próprio, o mesmo detém privilégios especiais para que ele possa executar operações que são negadas aos outros *applets*.

As proteções da linguagem *Java* também se aplicam na plataforma *Java Card*.

Transações e Atomicidade

Transação é um conjunto de atualizações de dados persistentes, sendo que para manter a integridade dos dados, ao iniciar uma atualização, ou todos os dados são atualizados, ou nenhum é. O JCRE oferece suporte para transações atômicas, de modo que os dados são restaurados aos valores pré-transação caso a mesma não seja concluída com sucesso.¹²

Em *Java Card*, o suporte e transações atômicas se dá através de chamadas aos métodos *JCSYSTEM.beginTransaction*, *JCSYSTEM.commitTransaction* e *JCSYSTEM.abortTransaction*. Uma transação é iniciada através da chamada ao método *JCSYSTEM.beginTransaction*, e concluída com uma chamada do método

JCSystem.commitTransaction. Todos os dados alterados entre as chamadas à estes dois métodos ou serão completamente concluídos ou abortados. Durante a execução de uma transação, a mesma pode ser interrompida através da chamada ao método *JCSystem.abortTransaction*.¹¹

2.2.2 Normas, Especificações e padrões

Esta seção do trabalho, estuda as normas, especificações e padrões ligados a *smart cards*. Entre elas: a *GlobalPlatform*, especificação PC/SC e padrão PKCS#15.

2.2.2.1 *GlobalPlatform*

GlobalPlatform é uma organização formada por empresas líderes dos mercados de pagamento e comunicação e setores governamentais. Seu principal objetivo é reduzir as barreiras que impedem que a tecnologia de *smart card* alcancem seu verdadeiro potencial.¹³

GlobalPlatform define uma poderosa e flexível especificação para emissores de cartões poderem criar aplicativos que possam ser portados entre diferentes hardwares. Esta especificação é bem aceita no mercado, e grande parte dos cartões existentes, atendem a tal especificação. Esta especificação é baseada na norma ISO7816.¹³

2.2.2.2 *Especificação PC/SC*

A especificação PC/SC é definida pela PC/SC *Workgroup*, cuja principal missão é garantir que os *smart cards*, leitores de cartões e computadores pessoais de diferentes fabricantes possam trabalhar em conjunto e facilitar o desenvolvimento de aplicações de *smart cards* para PC e outras plataformas de computação.¹⁴

Para alcançar sua missão, o PC/SC *Workgroup* identificou dois objetivos essenciais: Padronizar interfaces para dispositivos de interface e Especificar interfaces de PC comuns e mecanismos de controle.¹⁵

O PC/SC *Workgroup* é grupo formado pelas maiores empresas

ligadas a tecnologia de *smart card*, sendo a Gemalto e a Oracle os membros centrais.¹⁶

A especificação PC/SC possui 10 partes, e atualmente está na sua versão 2.01.12 de Novembro de 2012.¹⁴

2.2.2.3 PKCS#15

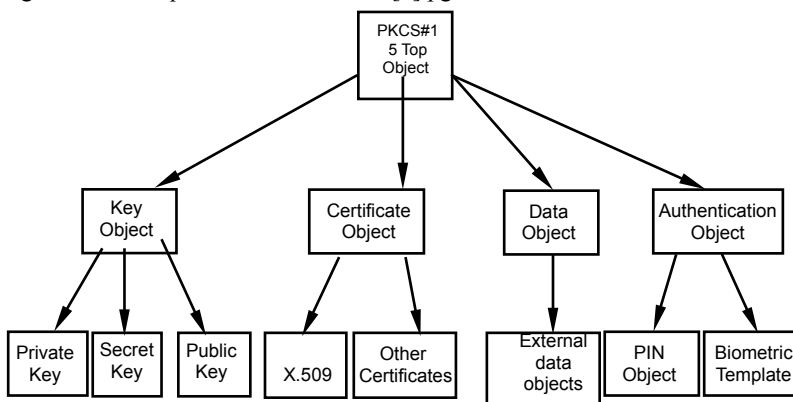
É um padrão baseado na ISO7816 desenvolvido pela RSA *Laboratories*, cujos objetivos são ²:

- Permitir a interoperabilidade entre componentes executando em diferentes plataformas;
- Permitir que os aplicativos aproveitem os produtos e componentes de diferentes fabricantes;
- Permitir avanços tecnológicos sem a necessidade de reescrever aplicações; e
- Manter a consistência com as normas existentes.

O padrão PKCS propõem um modelo de acesso a informação, no qual as informações armazenadas seguindo esta especificação, são lidas por um terminal e tratadas por um Intérprete PKCS15. O padrão PKCS15 propõem uma estrutura lógica de dados, assim, um intérprete PKCS15 será capaz de Ler e tratar qualquer cartão de CI que atenda a tal padrão².

Este documento define 4 classes gerais de objetos: chaves, certificados, objetos de autenticação e objetos de dados. Todas as classes gerais podem ter subclasses. Abaixo é apresentada uma figura com a estrutura de classes do padrão PKCS15².

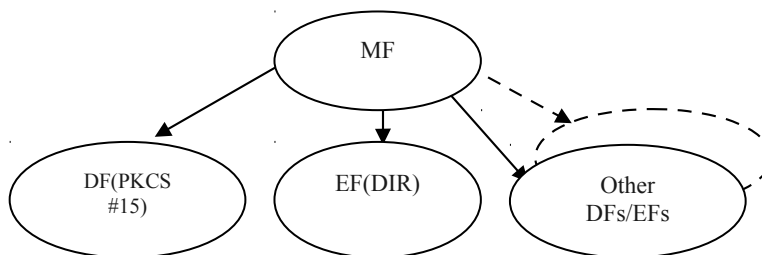
Figure 1: Hierarquia PKCS#15 Fonte: [2] pg 10



O padrão PKCS15 especifica, que os cartões de CI compatíveis, devem ter suporte à ISO/IEC 7816-4, ISO/IEC 7816-5 e ISO/IEC 7816-6. Recursos adicionais, tais como funções de gerenciamento de PIN avançados e operações de segurança mais avançadas podem requerer apoio às ISO/IEC 7816-8 e/ou ISO/IEC 7816-9.²

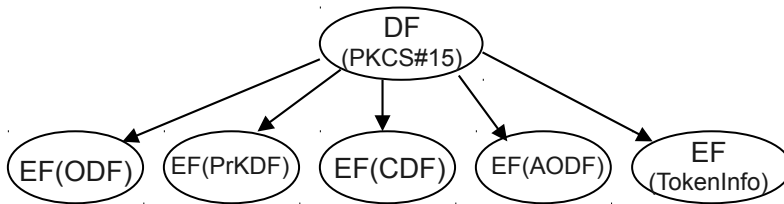
Um cartão apoiado neste padrão terá a seguinte estrutura:

Figure 2: Hierarquia de arquivos de um cartão PKCS#15- Fonte: [2] pg 11



O conteúdo do DF(PKCS#15) depende um pouco do tipo do cartão de CI e seu uso pretendido. Abaixo é apresentada a estrutura que acredita-se ser a mais comum.²

Figure 3: Hierarquia do DF PKCS#15 – Fonte: [2]



EF(ODF), este arquivo obrigatório contém ponteiros para os outros EF da estrutura PKCS15.²

Private Key Directory Files (PrKDFs), contém uma lista das chaves privadas de conhecimento da aplicação PKCS15. Ele contém informações como atributos gerais das chaves, tais como rótulo, uso pretendido, identificadores e etc. Ele pode conter ainda uma referência cruzada a objetos de autenticação utilizados para proteger o acesso a chave. Este EF é opcional, porém pelo menos um PrKDF deve existir.²

Public Key Directory Files (PuKDFs), semelhante ao PrKDF, porém contém a lista das chaves públicas conhecidas pela aplicação PKCS15. Quando a chave privada correspondente a determinada chave pública também estiver no cartão, ambas devem compartilhar o mesmo identificador. Caso exista um certificado ao qual a chave pública esteja associada, este também deverá compartilhar um mesmo identificador, ou seja, neste caso irão compartilhar o mesmo identificador, a chave privada, a chave pública e o certificado.²

Secret Key Directory Files (SKDFs), este EF também é opcional, porém ao menos um deve existir. Ele pode ser considerado como uma lista com as chaves secretas conhecidas pela aplicação PKCS15. Como o **PrKDF**, este também pode manter referências a objetos de autenticação usados para proteger os objetos chave.²

Certificate Directory Files (CDFs), contém uma lista dos certificados conhecidos pela aplicação PKCS15. Estes certificados podem conter referências à outros certificados.²

Data Object Directory Files (DODFs), contém outros objetos que não sejam chaves ou certificados, que são conhecidos pela aplicação PKCS15. Neste EF estão atributos gerais dos objetos, além do identificador da aplicação a qual o objeto pertence, se ele é um objeto público, ou privado, etc.²

Authentication Object Directory Files (AODFs), este DE

armazena objetos de autenticação, por exemplo: PIN, senhas, dados biométricos etc. Ele armazena atributos gerais tais como, no caso do PIN, informações sobre os caracteres aceitos, tamanho etc. Ele também pode armazenar referências a outros objetos de autenticação. Objetos de autenticação são utilizados para controlar o acesso a outros objetos, por exemplo, chaves. A informação de qual objeto de autenticação protege determinada chave, por exemplo, fica armazenada na PrKDF da chave.²

EF(TokenInfo), arquivo obrigatório que contém informações gerais do cartão, informações como capacidade, número de série do cartão, tipos de arquivos suportados, algoritmos implementados no cartão.²

EF(UnusedSpace), arquivo utilizado para manter o controle sobre o espaço livre no cartão. Este EF também é opcional, e quando presente deve conter pelo menos uma referência à um espaço vazio.²

Identificador de arquivos

O padrão PKCS15, define identificadores para os arquivos PKCS15. O padrão também define o RID como A0 00 00 00 63. A tabela 1 apresenta os identificadores de arquivos definidos pelo padrão.²

Tabela 1: File Identifiers – editado de [2] pgs 18 e 19

File	DF	File Identifier
MF	X	3F00 ₁₆
DIR		2F00 ₁₆
PKCS15	X	<i>Decided by application issuer (AID is RID “PKCS-15”)</i>
ODF		5031 ₁₆
TokenInfo		5032 ₁₆
UnusedSpace		5033 ₁₆)
(Reserved)		5034 ₁₆ - 5100 ₁₆ (<i>Reserved for future use</i>)

2.2.2.4 ISO/IEC 7816

A ISO/IEC 7816 é uma série de especificações para cartões de circuito integrado. Ela visa a possibilidade de intercâmbio entre tecnologias de diferentes fabricantes. A ISO/IEC 7816 é formada por 15 partes.¹⁷

O RIC, deve estar de acordo com 9 delas⁶. São elas:

- ISO/IEC 7816-1, especifica características físicas para cartões de contato;
- ISO/IEC 7816-2, especifica dimensões e localização dos contatos do cartão;
- ISO/IEC 7816-3, especifica protocolos de interface elétrica e para cartões assíncronos;
- ISO/IEC 7816-4, especifica o conteúdo das mensagens, comandos e respostas transmitidos pelo dispositivo de interface com o cartão. Métodos de acesso a arquivos e dados no cartão;
- ISO/IEC 7816-5, especifica o registro de provedores de aplicativos;
- ISO/IEC 7816-6, especifica elementos de dados intersetoriais para intercâmbio;
- ISO/IEC 7816-7, especifica comandos de linguagem estruturada de consulta;
- ISO/IEC 7816-8, especifica comandos para operações de segurança;
- ISO/IEC 7816-11, especifica métodos para verificação pessoal através da biometria;

2.2.2.5 Federal Information Processing Standards Publication - FIPS 140

É uma norma emitida pelo *National Institute of Standards and Technology* - NIST dos Estados Unidos, onde são especificados requisitos para a segurança de módulos criptografados. Ele especifica 4 níveis crescentes de segurança. Fazem parte dos requisitos de segurança especificados, portas do módulo criptográfico e interfaces, funções e serviços, modelo de estado finito, segurança física, ambiente operacional e gestão de chaves criptográficas.¹⁸

Cada um dos 4 níveis definidos, substituí em sua totalidade o nível anterior.

Para o RIC, foi especificado pelo normativo do ministério da justiça⁶, que o RIC deve atender a norma FIPS 140-2, a qual além de requisitos básicos de segurança, especifica também recursos de evidência de violação ao modulo criptográfico.¹⁸

2.3 CRIPTOGRAFIA

Esta seção do trabalho apresenta os fundamentos teóricos básicos sobre criptografia. Nela são definidos conceitos como criptografia simétrica, assimétrica e assinatura digital.

2.3.1 Criptografia simétrica

Criptografia simétrica, é o sistema criptográfico onde uma única chave é utilizada para cifrar e decifrar uma mensagem. Nestes sistemas, a segurança é dada pelo sigilo da chave, e pela incomputabilidade da mesma a partir de uma mensagem cifrada.

A criptografia simétrica, visa oferecer privacidade ou confidencialidade aos dados, e não fornece garantias de autenticação, integridade, e não repúdio, características estas desejáveis a qualquer sistema.¹⁹

A principal dificuldade em implantar um sistema criptográfico de chave simétrica, está na distribuição da chave, uma vez que a interceptação da chave durante a sua distribuição pode comprometer a segurança de todo o sistema.¹⁹

2.3.2 Criptografia assimétrica

Criptografia assimétrica é o sistema criptográfico onde cada entidade possui um par de chaves e e d , sendo uma chave pública e e sua correspondente privada d . A segurança deste sistema está baseado no sigilo da chave privada de cada entidade, visto que ao obter a chave privada de uma entidade, é possível enviar mensagens em seu nome, além de ler todas as mensagens enviadas a tal entidade.¹⁹

Os nomes criptografia assimétrica e criptografia de chave pública são dados pelas características do sistema. Onde são utilizadas duas chaves diferentes nos processos de cifragem e decifragem da mensagem, por isso o nome criptografia assimétrica. Já o nome criptografia de chave pública é dado pela característica de uma das chaves do par, ser disponibilizada publicamente.

Quando um atacante consegue decifrar uma mensagem c , sem

conhecimento da respectiva chave privada, é dito que o sistema foi quebrado. Quando uma entidade obtém a chave privada de outra entidade é dito que o sistema foi completamente quebrado.

A principal vantagem de um sistema criptográfico de chave pública está na facilidade de transmitir as chaves públicas autênticas em relação a distribuição de chaves privadas do sistema simétrico. Já sua principal desvantagem está no desempenho, onde sistemas criptográficos de chaves públicas são bem inferiores aos sistemas criptográficos de chave simétrica. Na prática, a criptografia assimétrica é utilizada para distribuir a chave simétrica utilizada durante o resto da comunicação entre duas entidades, ou seja, grosseiramente falando, uma entidade A que deseja estabelecer uma comunicação com uma entidade B , cria uma chave simétrica s , cifra a mesma com a transformação $Ee(m)$, onde e é a chave pública de B e m é a chave simétrica s , e à envia a B . Ao receber, B decifra a mensagem, e passa a se comunicar com A através da chave simétrica s , compartilhada e de conhecimento somente das duas entidades.¹⁹

Um sistema criptográfico de chave pública por si só, não provê autenticidade de dados, nem tão pouco integridade dos mesmos, porém podem ser utilizados recursos adicionais para prover tais garantias.

Um sistema criptográfico de chave pública exige que exista uma forma segura de distribuição das chaves autênticas. Existem várias técnicas de distribuição de chaves públicas, dentre as quais: o uso de um servidor confiável, arquivo confiável e canal seguro.¹⁹

2.3.3 Funções *Hash* ou Funções de resumo.

Funções *hash*, são funções que em geral, recebem como entrada uma sequência de n -bits, e retornam como saída outra sequência de m -bits, geralmente sendo m menor que n .¹⁹

As funções de *hash* desempenham um papel fundamental na criptografia moderna, onde são utilizadas para ajudar a prover garantias de integridade de dados e em assinaturas digitais.

Uma função *hash* criptográfica se difere em muitos pontos das funções de *hash* tradicionais, comumente utilizadas em sistemas computacionais para os mais variados fins. Para que uma função de *hash* seja adequada à criptografia, é necessário que para cada entrada diferente ela gere um *hash* diferente, ou seja, deve ser intratável achar duas entradas de tamanho arbitrário que gerem a mesma saída.

2.3.4 Assinatura digital

Uma assinatura digital é um número dependente de um segredo de conhecimento somente do assinante. Assinaturas digitais devem ser verificáveis de tal forma que em caso de conflito, uma terceira parte confiável possa validar uma assinatura digital sem o conhecimento da chave privada do assinante.¹⁹

Assinatura digital tem muitas aplicações na área da segurança da informação, através dela é possível prover: autenticação, integridade de dados e não repúdio. Atualmente o principal uso de assinatura digital está em sistemas criptográficos de chave pública.

O processo de assinar e verificar assinaturas é baseado no sistema criptográfico assimétrico, onde uma mensagem é cifrada ou assinada com uma chave privada e o destino utiliza a sua correspondente pública para decifrar e validar a assinatura. Como a chave privada é de conhecimento somente do seu proprietário, um documento cifrado com a mesma não poderá ser repudiado. Os certificados digitais completam este cenário, provendo garantias de propriedade das chaves.

2.4 AUTENTICAÇÃO E IDENTIFICAÇÃO BIOMÉTRICA

Autenticação refere-se a confirmar a identidade alegada de indivíduo. Já Identificação refere-se a determinar a Identidade de um indivíduo. Tradicionalmente os métodos de autenticação envolvem o compartilhamento de um segredo entre o usuário e o objeto de segurança. Este método apresenta várias vulnerabilidades. Primeiro, o segredo pode ser descobertos por terceiros. Segundo, o segredo pode ser esquecido. Isto faz com que muitas pessoas optem por utilizar segredos “fracos”, ou seja, que podem ser facilmente descobertos por um atacante.²⁰

Neste contexto, surge a biometria, como uma forma de autenticar/identificar uma pessoa de forma segura. A autenticação/identificação através da biometria é baseada em uma característica única de cada pessoa. Entre as características utilizadas estão: face, geometria da mão, impressão digital, íris, retina, voz e comportamentos.

Segundo Paulo Sérgio Magalhães²⁰, cada uma das técnicas de autenticação/identificação biométrica pode ser avaliada levando em

consideração uma série de fatores, entre eles: grau de fiabilidade, nível de conforto, nível de aceitação e custo de implementação.

Um titular do RIC poderá ser identificado/autenticado mediante a biometria da digital ou ainda através da face, sendo a primeira forma a mais comum no mundo.⁷

2.4.1 Reconhecimento Facial

O processo de reconhecimento facial é baseado na localização de pontos fixos (olhos, boca, nariz e etc...). Para esta técnica, utiliza-se uma imagem do indivíduo, a qual é submetida ao processo de detecção de um rosto e depois comparada a modelos de uma base de dados.²⁰

A ICAO (Organização de aviação Civil Internacional) a considera a tecnologia biométrica de funcionamento mundial, e deve ser implementada por um DVLM-e (*Documento de viagem oficial de leitura mecânica e com um Circuito Integrado sem contato incorporado e capacidade para fins de identificação Biométrica*). A ISO/IEC 19794 apresenta especificações para este tipo de Autenticação/Identificação biométrica.⁷

Uma grande vantagem desta técnica está no fato, de uma foto(imagem) não revelar nenhuma informação que tradicionalmente o indivíduo não revele. Esta é dentre as técnicas de autenticação/identificação a mais aceita em diferentes culturas.⁷

O Reconhecimento facial é uma das formas de identificação/autenticação biométrica existentes no RIC, ela é realizada através de uma imagem do rosto do titular armazenada no chip sem contato do RIC.⁶

2.4.2 Impressão digital

É a técnica mais utilizada atualmente, apresentando um bom nível de aceitação e um baixo nível de fiabilidade (grau de erros do sistema). A desvantagem desta técnica está no fato da maioria dos equipamentos leitores de digitais não distinguirem entre um dedo vivo, morto ou separado do corpo. Apesar de existirem leitores capazes de realizar tal distinção, estas tecnologias ainda são muito caras e não apresentam um nível de maturidade aceitável.²⁰

Existem três classes de tecnologias para impressão digital, sistemas baseados na imagem da impressão, sistemas baseados em detalhes minuciosos e sistemas baseados na configuração da impressão.⁷

A impressão digital, é uma técnica opcional aceita pela ICAO em DVLM-e. O estado que deseja optar pelo seu uso, deve disponibilizar uma imagem da impressão digital, para que o estado verificador possa aplicar uma das 3 classes de tecnologias.⁷

A identificação/autenticação através da impressão digital também estará presente no RIC, através das imagens dos 5 dedos do titular.⁶

2.4.3 Leitura da Íris

Método com eficácia acima da média, baseado na análise do anel colorido que envolve a pupila do olho humano.²⁰

A Leitura de íris é aceita pela ICAO como uma técnica biométrica opcional. A autenticação/identificação biométrica através da leitura da íris se complica pela escassez de fornecedores. O estado que optar por tal método de autenticação/identificação deve disponibilizar uma imagem da íris.⁷

2.5 CÓDIGOS OCR-B

O RIC será emitido em conformidade com as normas internacionais estabelecidas pela Organização de Aviação Civil Internacional (ICAO)⁶. Isto fará com que o RIC seja aceito como documento de identidade em qualquer porto, aeroporto ou fronteira que adote o padrão. Para estar em conformidade com as normas da ICAO, o RIC irá contar com um código OCR-B, este código forma a Zona de leitura mecânica - ZLM. A ZLM tem a finalidade de agilizar o processo de identificação e ou autenticação em portos, aeroportos e fronteiras em todo o mundo.⁷

A ZLM apresenta um conjunto de dados essenciais formatados seguindo as diretrizes definidas pelo documento 9303⁷ (Documentos de viagem de leitura mecânica) da ICAO, onde são definidos os dados contidos, a sequência, o tamanho, regras de tratamento e regras de verificação, além de características do código em si, características como fonte, largura, altura e etc... Os dados da ZLM tem um formato

requerer que mais informações sejam armazenadas para uso na identificação e ou autenticação do usuário, por exemplo, pode requerer que dados biométricos sejam armazenados. Para atender a este requisito, a ICAO através do documento 9303⁷ parte 3 volume 2, define as especificações adicionais para documentos de viagem de leitura mecânica com capacidade biométrica, o qual poderá ser usado por qualquer estado que possua os equipamentos adequados na identificação/autenticação do titular. Este documento, define o reconhecimento facial como tecnologia de identificação/autenticação obrigatório para um documento de viagem de leitura mecânica com capacidade de identificação biométrica - DVLM-e. Além do reconhecimento facial, os estados emissores podem ainda adotar de forma opcional por outras formas de identificação/autenticação do titular, são elas: reconhecimento da digital e da íris, em todos os casos o reconhecimento é feito através de imagens de alta resolução armazenadas em um chip sem contato no documento.⁷

Para auxiliar os estados receptores na diferenciação dos DVLM e DVML-e, os DVLM-e serão emitidos com um símbolo (Figura 5) indicativo de que possuem capacidade de identificação biométrica.

Figure 5: Indicação visual que um DVLM é um DVLM-e – Fonte : [7]



Além do símbolo, os DVLM-e devem contar com um circuito integrado sem contato, com capacidade de armazenamento de dados de no mínimo 32 KB, estruturados conforme documento 9303.⁷

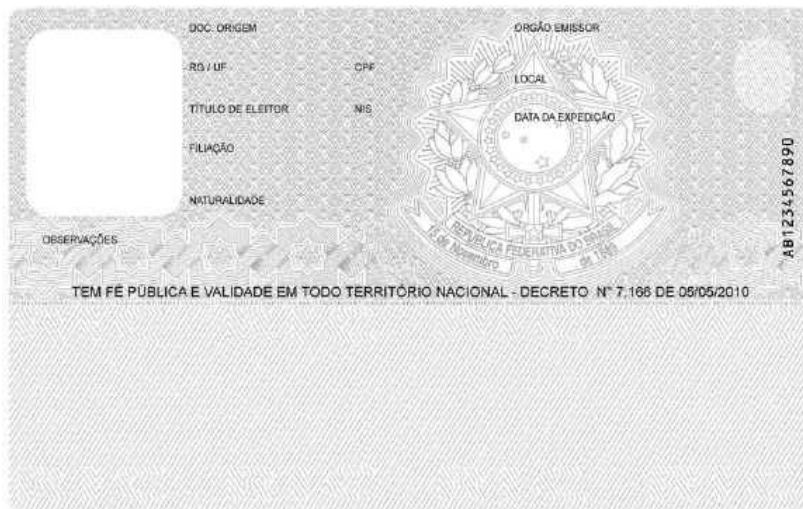
Segundo a Resolução MJ N° 2, de 10 de Setembro de 2010⁶, o

RIC irá contar com um circuito integrado sem contato com capacidade de pelo menos 64KB de armazenamento, seguindo a estrutura de dados definidas pela ICAO. Neste circuito integrado, estarão armazenadas imagens do rosto e da digital do titular, permitindo a identificação do usuário biometricamente de ambas as formas. Neste ponto foi encontrada uma pequena inconsistência na especificação do novo documento, pois segundo o documento 9303⁷ da ICAO, como mencionado anteriormente, um DVLM-e deve possuir o símbolo indicativo, porém o modelo gráfico do RIC, publicado na resolução e replicado aqui, não possui este símbolo.

Figure 6: Anverso do documento de Registro de Identidade Civil - Fonte : [6]



Figure 7: Reverso do documento de Registro de Identidade Civil – Fonte : [6]



2.6.1 Dados e sua estrutura

A ICAO através do documento 9303⁷, define uma estrutura lógica para os dados armazenados no chip sem contato dos DVLM-e, esta estrutura tem os objetivos de assegurar a proteção dos dados armazenados e permitir que documentos de diferente países sejam tratados de uma forma única, agilizando o processo de identificação/autenticação. Para isso, são definidos dados obrigatórios, dados opcionais, formato, além de regras de acesso. Os dados obrigatórios são, os dados presentes na ZLM, uma foto do rosto do titular, além dos dados de controle de autenticação, que são eles o EF.COM e EF.SOD. No documento 9303⁷ da ICAO, são apresentados em detalhes todos os dados obrigatórios e opcionais, além do formato dos mesmos.

2.7 DOCUMENTOS DE IDENTIDADE ELETRÔNICOS NO MUNDO

Esta seção visa estudar outros documentos com características semelhantes ao RIC, e eventualmente realizar comparações entre os mesmos.

2.7.1 Cartão cidadão – Portugal

O cartão cidadão é um documento de identidade português com características bem semelhantes ao RIC. Assim como o RIC, o cartão cidadão permite ao “titular identificar-se presencialmente de forma segura”, “identificar-se perante serviços informatizados” e “autenticar documentos eletrônicos”²¹. O cartão cidadão também tem o objetivo de integrar vários documentos em um único suporte documental, além de ser o principal motor da concretização dos princípios do governo eletrônico em Portugal.²²

Fisicamente o cartão cidadão também é bem semelhante ao RIC, apresentando um formato de *smart card*. O cartão cidadão apresenta na frente a fotografia e os elementos de identificação civil. No verso, apresenta os números de identificação dos diferentes documentos integrados pelo cartão cidadão, além de uma ZLM e o chip.

O cartão cidadão conta ainda com um chip de contato, o qual contém um certificado digital para autenticação e assinatura eletrônica. Pode ainda conter as mesmas informações do cartão físico. Este documento permite o relacionamento do titular com a sociedade através da internet, telefone (utilizando para isso a identificação e autenticação através das chaves de acesso (“*one-time-password*”) obtidas através do cartão de cidadão e o respectivo leitor e presencialmente.²³

Valem destacar três pontos de divergência entre o cartão cidadão e o RIC. O cartão cidadão não é um documento de viagem com capacidade biométrica, todo RIC é emitido com um certificado digital do titular e o cartão cidadão conta com uma ampla plataforma governamental baseada no mesmo.

2.7.2 Documento Nacional de *Identidad electrónico* – Espanha

O Documento Nacional de *Identidad electrónico* – DNIE é um documento de identidade em formato de *smart card*, o qual possui, assim como o RIC, um chip de contato, onde é armazenado um certificado digital. O “DNIE é a adaptação da identificação tradicional com a nova realidade de uma sociedade interconectada por redes de comunicação”.²⁴ O DNIE é expedido pela *Policia* Nacional Espanhola é já alcançou o número de 34.000.000 de documentos expedidos.²⁵

2.7.3 EID – Bélgica

O eID é o cartão de identidade eletrônico para os belgas com mais de 12 anos. Com ele é possível viajar na Bélgica e outros países da União Europeia. O EID contém um microchip o qual contém além das informações visíveis no cartão, certificados digitais do titular. Tanto para autenticação quanto para assinatura de documentos.²⁶

Hoje são disponibilizadas diversas aplicações para o eID, com elas é possível por exemplo, utilizá-lo como um cartão de biblioteca, como uma chave ou mesmo como um bilhete de trem.²⁶

Além do eID, a Bélgica disponibiliza para as crianças os *kids-ID*. O *kids-ID* é um documento de identidade para menores de 12 anos e tem validade de 3 anos. É possível ativar um aplicativo chamada “Olá Pais”, através do qual, caso uma criança se perca, alguém que à localizar poderá ligar para um número oficial e o aplicativo automaticamente ira chamar todos os números especificados pelos pais anteriormente, caso nenhum dos números atenda, o número de emergência da *Child Focus* é acionado.²⁷

2.7.4 *Estonian identity card* – Estônia

O *Estonian identity card* é um documento de identificação nacional da Estônia . Este documento também possui um chip integrado com dois certificados digitais, sendo um de autenticação e o segundo para assinatura digital. Assim como o RIC, este cartão foi desenvolvido de acordo com a norma ICAO 9303, porém o documento da Estônia não

possui capacidades biométricas. ²⁸

3 DESENVOLVIMENTO DE APLICAÇÕES EMBARCADAS

O desenvolvimento de aplicações embarcadas no RIC é um grande desafio, principalmente pela pequena capacidade de armazenamento de ambos os chips e pelas regras impostas visando a segurança do documento pela ICP-Brasil para o chip com contato e pela ICAO para o chip sem contato. Outro ponto dificultador é a escassez de informação sobre estes recursos do RIC. A única fonte de informação sobre os mesmos é a resolução do MJ nº2, de 10 de Setembro de 2010⁶, apresentada na seção 2.1.1.4, que dispõe sobre as especificações básicas do novo documento. Nesta resolução são definidas características básicas e tem como foco propriedades físicas e elétricas. Sistema operacional, interfaces de entrada e saída de dados e rotinas internas do sistema operacional não são especificados.

3.1 CHIP COM CONTATO

O desenvolvimento de aplicações para execução embarcada no chip de contato do RIC, enfrenta dois grandes desafios: capacidade de armazenamento e a homologação da ICP-BRASIL.

3.1.1 Capacidade de armazenamento

O chip com contato do RIC deverá conter no mínimo 4 imagens de impressões digitais e dados de certificação digital⁶. O doc.9393 da ICAO define que para intercambio internacional, os dados biométricos devem estar armazenados na forma de imagens, define também que o tamanho ideal de uma imagem da impressão digital do titular deve ter 10 Kb. Assim sendo, é reservado para as 4 impressões digitais um espaço total de 40Kb. Outro dado importante, um cartão criptográfico deve ter uma capacidade de armazenamento mínima de 16 KB, segundo o manual de condutas técnicas (MCT)1 da ICP-Brasil²⁹, ou seja, o tamanho dos dados obrigatórios armazenados no chip de contato do RIC, pode chegar a 56KB, restando apenas 8Kb para novas aplicações. Isto limita bastante a possibilidade de aplicações que poderiam ser embarcadas no novo documento.

3.1.2 Homologação na ICP-BRASIL

A falta de uma política para o desenvolvimento de aplicações para execução embarcada em um cartão criptográfico, é outro desafio no desenvolvimento das mesmas. Atualmente não existem normas, padrões ou mesmo um processo de homologação para tais aplicações. A única diretriz que existe hoje é a MCT1 da ICP-Brasil²⁹, a qual apesar de permitir a presença de serviços diversos em um cartão criptográfico, os mesmos já devem estar instalados durante a homologação do cartão junto a ICP, desta forma o desenvolvimento de aplicações embarcadas, fica restrita aos fabricantes dos cartões. Para que seja possível embarcar novas aplicações ao RIC, será necessário estabelecer um processo para homologação de *applets* para execução embarcada em cartões criptográficos.

3.2 CHIP SEM CONTATO

Para o chip sem contato, não é possível desenvolver aplicações para execução embarcada, uma vez que na fase de personalização, o chip sem contato deve ser bloqueado contra alterações, conforme define o doc.9303⁷ da ICAO.

3.3 EXPERIMENTOS

Durante a fase de experimentos foram desenvolvidas duas aplicações. A primeira foi um *applet java card* para execução embarcada no RIC. Já a segunda é uma aplicação *desktop* a qual se comunica com o cartão. As duas aplicações em conjunto simulam um sistema gerenciador de ponto eletrônico utilizando um *smart card* para armazenar os bilhetes do titular.

O experimento realizado tem alguns objetivos específicos, buscando atingir os objetivos principais de validar as informações levantadas durante a fase de fundamentação teórica, testar tecnologias e implementar exemplos de códigos capazes de interagir com o RIC, seja de forma embarcada, ou mesmo no acesso as informações presentes no RIC.

O sistema gerenciador de ponto eletrônico utiliza um cartão do tipo *smart card* para armazenar os registros. Estes registros podem ser exportados do cartão tanto em texto plano, quanto em texto cifrado. A segunda forma, é utilizada para contornar a limitação de armazenamento do *smart card*. Nesta forma, o usuário pode exportar seus registros para o seu computador pessoal e mantê-los cifrados com uma chave simétrica que permanece dentro do cartão. É disponibilizado também, uma forma de decifrar os registros que estão fora do cartão, onde os mesmos são enviados ao cartão decifrados e retornados à aplicação *Desktop*.

O sistema desenvolvido utiliza técnicas de criptografia para autenticar o *applet* e gerar um registro autentico que não possa ter sua autoria negada por nenhuma das partes. Para isso, um certificado digital é armazenado no cartão, o mesmo é utilizado para autenticar o titular do cartão e assinar o registro. A aplicação desktop também possui um certificado digital e também assina o registro. Desta forma o registro gerado é assinado por ambas as partes. O protocolo utilizado é apresentado na figura 8.

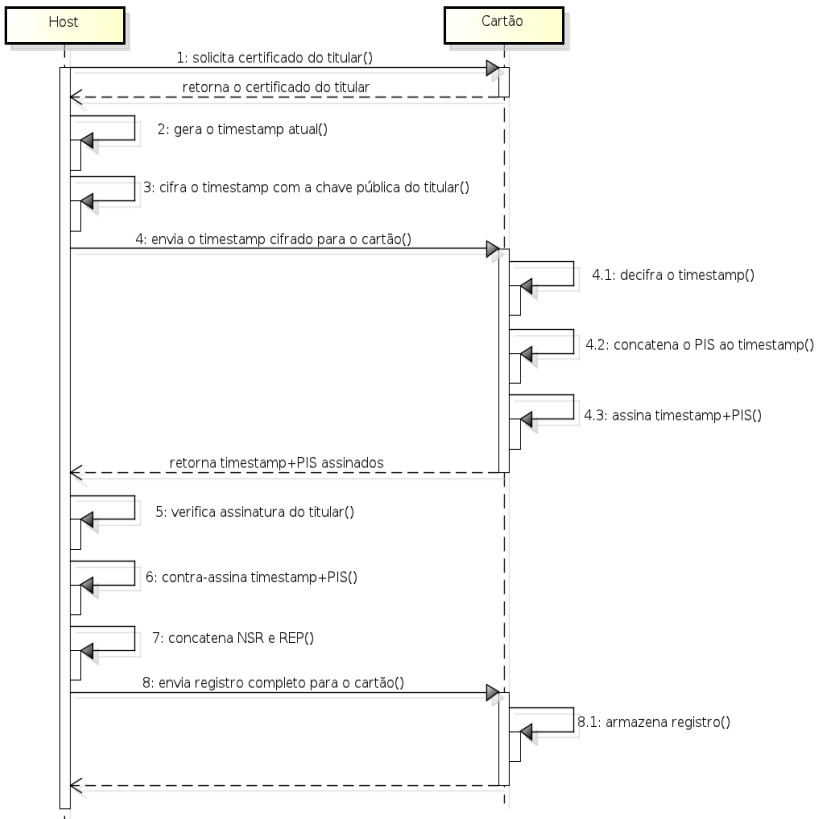
As operações de importação de registros e edição dos dados pessoais, são realizadas mediante apresentação do PIN válido. Uma vez validado o PIN, o mesmo continua válido durante toda a seção CAD. Existem também métodos para edição e inicialização do PIN, sendo o último utilizado somente quando o cartão é iniciado e ainda não existe um PIN no mesmo.

As funções do *applet* são acessadas através de códigos de instruções, foram desenvolvidos 15 códigos de instrução para o *applet* em estudo.

O uso de um sistema como apresentado traria as seguintes vantagens ao titular do cartão:

- Não seria necessário armazenar dezenas de registros de pontos;
- Facilidade para conferir os registros de bilhetes do trabalhador com os dados passados pela empresa ;
- Os dados poderiam ser armazenados de forma segura por tempo indeterminado, ao contrário dos bilhetes atuais que possuem um tempo de retenção da informação limitado.

Figure 8: Comunicação entre Host e o Cartão durante o registro de Ponto



Para a realização dos experimentos, foi utilizado um *smart cards* com suporte a tecnologia *java card*.

3.3.1 Objetivos específicos

- Visualizar as etapas de desenvolvimento de um *applet java card*;
- Manipular dados em *java card*(cópia de *arrays*, tipos, operações, API e etc...);
- Armazenar dados no cartão;

- Comunicação entre *applet java card* e aplicação *host*;
- Verificar o tratamento de exceções;
- Controlar o acesso e privilégios através de PIN e PUK em *java card*;
- Estudar a criptografia simétrica de dados em *java card*;
- Estudar a criptografia assimétrica de dados em *java card*;
- Estudar outras operações criptográficas;

3.3.2 Escopo

O *software* em desenvolvimento como dito anteriormente, é um experimento, logo não tem a intenção de ser uma proposta para a adição desta funcionalidade ao RIC. Por ser um experimento onde o principal objetivo é exemplificar o processo de desenvolvimento de aplicações para execução embarcada no RIC, o software desenvolvimento não terá obrigatoriedade de comprimento das normas e leis que regulamentam o controle de Ponto do trabalhador no Brasil.

3.3.3 Etapas do desenvolvimento de um *applet Java Card*

O desenvolvimento de um *applet java card*, assim como o desenvolvimento de qualquer aplicação *Java*, começa pela codificação de um ou mais arquivos de classes. Após a fase de codificação, os arquivos gerados são compilados com um compilador *Java*, por exemplo o *javac*, e convertidos em arquivos *.cap* e *.exp*, sendo o arquivo *.cap* o binário o qual será interpretado pela máquina virtual *java card*.

Todo *applet java card*, possui uma classe que estende a classe *Applet* do pacote *javacard.framework*. Esta classe será a porta de comunicação entre o *JCRE* e o nosso *applet*. Uma classe que estende a classe *Applet* deve implementar os métodos *install*, *process*, *select* e *deselect*. Para o desenvolvimento do *applet* Ponto Eletrônico, a classe *Core* foi definida como uma extensão da classe *Applet*, nela foram implementados os métodos *install* e *process*. Os métodos *select* e *deselect* não foram sobrescritos visto que nenhuma preparação adicional se vez necessária para tornar o *applet* uma vez instalado apto a receber os comandos seguintes, tão pouco foi necessário limpar ou desalocar

recursos antes de finalizar sua execução.

Além dos métodos obrigatórios, também foram implementados métodos para tratar cada um dos possíveis *bytes* de instruções. Após a seleção do *applet*, todos os APDUs seguintes são direcionados ao *applet*, onde são recebidos no método *process*, método este responsável por receber o APDU e dar o encaminhamento correto.

3.3.4 Manipulação de dados em *java card*(cópia de *arrays*, tipos, operações, API e etc...).

Java card possui uma grande API, a qual fornece vários métodos e classes que facilitam o desenvolvimento de aplicações embargadas. As principais são:

- *APDU* – classe que cria uma abstração dos pacotes físicos transmitidos entre o cartão e o leitor, seguindo o formato APDU definido na norma ISO7816-4. Esta classe suporta somente mensagens que estejam em conformidade com a estrutura de comandos e respostas definidos na ISO7816-4. Uma limitação desta classe, é não aceitar campos de comprimento estendido. A classe APDU é projetada para ser independente de protocolo de transporte. Ela possui vários métodos, entre eles, métodos que permitem identificar o protocolo utilizado na comunicação, o tamanho configurado dos bloco de saída e entrada, ler e enviar *bytes*.

Tabela 2: Fragmento de código com exemplo de uso da classe APDU

```
public void process(APDU apdu) throws IOException {  
  
    byte[] cmd_apdu = apdu.getBuffer();  
    byte cla = cmd_apdu[ISO7816.OFFSET_CLA];  
    byte p1 = cmd_apdu[ISO7816.OFFSET_P1];  
    byte p2 = cmd_apdu[ISO7816.OFFSET_P2];  
    ...  
    apdu.setOutgoing();  
    apdu.setOutgoingLength((short)lc);  
    apdu.sendBytesLong(data, (short)0, lc);  
}
```

- *Util* – A classe *Util* contém funções utilitárias comuns, alguns métodos são implementados como funções nativas por questões de desempenho. Todos os seus métodos são estáticos. A classe *Util* possui métodos para cópia e comparação de *arrays*, concatenação e separação de *bytes* entre outros.

Tabela 3: Fragmento de código com exemplo de uso da classe Util

```
Util.arrayCopy(cmd_apdu,(short) (ISO7816.OFFSET_CDATA+
Persistencia.TAMANHOPIS+1), data, (short) 0, (short) (lc-
Persistencia.TAMANHOPIS));
```

- ISO7816 – A classe ISO7816 define constantes relacionadas a ISO7816-3 e ISO7816-4. Todas as suas constantes são estáticas.

As constantes com prefixo “SW_” são constantes definidas conforme ISO7816-4 para bytes de *status* de respostas. Os atributos com o sufixo “_00” necessitam de um byte de complemento. Já os atributos com prefixo “OFFSET_” definem constantes para manipulação do *buffer* do *APDU*, estas constantes são definidas seguindo a ISO7816-4.

Tabela 4: Fragmento de código com exemplo de uso da classe ISO7816

```
...
byte[] cmd_apdu = apdu.getBuffer();
byte cla = cmd_apdu[ISO7816.OFFSET_CLA];
byte p1= cmd_apdu[ISO7816.OFFSET_P1];
byte p2= cmd_apdu[ISO7816.OFFSET_P2];
...
```

- *Applet* – A classe *Applet* é talvez a classe mais importante da API, visto que todo o *applet* para executar no cartão deve ter ao menos uma classe que estenda a classe abstrata *Applet*.

- *JCSystem* – A classe *JCSystem* possui vários métodos para controle de execução do *applet*, gerenciamento de recursos, compartilhamento de objetos e operações atômicas, além de controlar a persistência a transitoriedade dos objetos. Todos os seus métodos são estáticos.

Para controle de transações, são disponibilizados os métodos *beginTransaction*, *abortTransaction*, *commitTransaction*, *getMaxCommitCapacity*, *getUnusedCommitCapacity* e *getTransactionDepth*. Para controle de execução, são disponibilizados os métodos *getAID*, *getAppletShareableInterfaceObject*, *getPreviousContextAID*, *getVersion* e *lookipAID*. Para controle de objetos persistentes e transitórios, são disponibilizados os métodos *isTransient*, *makeTransientBooleanArray*, *makeTransientByteArray*, *makeTransientObjectArray* e *makeTransientShortArray*.

Tabela 5: Fragmento de código com exemplo de uso da classe JCSysstem

```
byte[]chave=JCSysstem.makeTransientByteArray((short)8,  
JCSysstem.CLEAR_ON_RESET);
```

3.3.5 Armazenar dados no cartão

O armazenamento de dados no cartão, ou seja, dos registros de batidas de ponto, será realizada através de um objeto persistente. Objetos persistentes são armazenadas na área de EEPROM, por isso a performance na leitura e escrita de informações neste objeto é muito inferior. A escrita em um objeto persistente é cerca de 1000 vezes mais lenta que a escrita em um objeto não persistente. Optou-se por utilizar este tipo de objeto, pois pretende-se desenvolver um *applet* capaz de armazenar até 1 mês de registros de um usuário, cerca de 100 registros, considerando 4 registros por dia. Cada registro contém um *timestamp*, o número do REP, NSR do registro, PIS do usuário, assinatura da aplicação *host*. O tamanho total do registro ficou em torno de 190 *bytes*, ou seja, o espaço total necessário para armazenar estes registros é cerca de 19 *Kbytes*, muito mais que os 2 *Kbytes* de memória RAM de um cartão.

Neste *applet* foi definida uma classe exclusiva para tratar do armazenamento dos dados, esta classe foi denominada *Persistencia* e possui um *array* de *bytes* onde são armazenados os registros. Nesta classe foram disponibilizados métodos para manipulação da memória, são eles: *registraPonto*, *limpaDados* e *exportarDados*, conforme segue:

- *registraPonto*, método que recebe como parâmetro um *array* de *bytes* contendo um registro de ponto, já com a assinatura e adiciona

este registro a memória. Adicionar o registro a memória significa inseri-lo no *array* de *bytes* de memória, que nada mais é que, fazer com que os dados fiquem armazenados na EEPROM, visto que o objeto *Persistencia* é um objeto persistente como citado anteriormente.

- *exportaDados*, método que recebe dois parâmetros do tipo *byte*, e retorna todos os registros desde a posição representada pelo primeiro parâmetro até o total de registros representados pelo segundo parâmetro.

Tabela 6: Exemplo de registro armazenado no cartão

```
303133373531313335373232323031332D30352D3236203230
3A35353A32302E3236333030303030303030303030303032353230
1823A42E80C73417E6D103C17C33F38480EC9A0E126F6249A9ED
987677D4F291129D65032CC737B05B930AFD8CFCFC8D29A9014
458896AD8D8E43CD9A8AC3DE10DF1EB6FDC021DFAA40ACE0
52578D5046576D64469E2FDCB7AE01839340BB81587CD8B0E689
FA667771D30F26C6020D810CCCAAC7B6F1DB1B167A3709738F9
94
PIS  TIMESTAMP  REP  NSR  Assinatura
```

3.3.6 Comunicação entre *applet java card* e aplicação *host*;

A comunicação entre o *applet* e a aplicação *host* é feita através de pequenos pacotes de dados denominados de APDU, o tratamento dado para enviar e receber estes pacotes do lado *host* e do lado do cartão são diferentes, visto que do lado *host* pode-se utilizar qualquer linguagem desktop ou web, enquanto do lado do cartão utiliza-se uma linguagem de programação embarcada.

Neste experimento, optou-se por utilizar a tecnologia *Java* dos dois lados da comunicação. No lado *host* ou *desktop* ou ainda aplicação externa ao cartão, foi utilizado o *Java Standard Edition* - JSE. Já no lado *Applet* ou cartão, foi utilizado a tecnologia *java card*.

3.3.6.1 Lado *Host* ou *Desktop*

Do lado *host*, foram utilizadas as classes *TerminalFactory*, *CardTerminal*, *Card*, *CardChannel*, *CommandAPDU* e *ResponseAPDU*. Todas do pacote *javax.smartcardio*.

A estrutura simplificada de um método para envio de APDUs ao *applet* está explicado no diagrama do anexo A.

No *Applet* Ponto Eletrônico, a responsabilidade de enviar os comandos APDU ao *applet* e depois tratar as respostas, foi dada à classe *Host*. Para atender à esta responsabilidade, foram implementados 14 métodos, sendo 2 métodos para o tratamento de dados recebidos, 2 métodos auxiliares, 9 métodos para montar os comandos e tratar os dados que serão enviados ao cartão e 1 método para envio efetivo do APDU e recebimento da resposta. Os bytes CLA, INS, P1, P2 e Data de cada comando, são montados(definidos ou calculados) dentro dos métodos de montagem de comandos. Estes métodos, apenas definem os dados que serão enviados, e posteriormente chamam o método *enviaAPDU* para envio efetivo do comando. Caso o comando a ser enviado necessite de autenticação através do PIN, o método de montagem chama antes do comando de envio de APDU, o método para validação do PIN. A tabela 7 exemplifica um método de montagem e a tabela 8 exemplifica um método de envio de apdu.

Tabela 7: Exemplo de método de montagem

```
public String[][] importaDadosCartao(){
    ...
    byte p1 = 0x00;//define p1
    byte p2 = MAXIMOREGISTROSAPDU; //define p2
    int
    qts_comandos=TAMANHOLISTA/MAXIMOREGISTROSAPDU;//calcula o
    número de comandos que serão enviados ao cartão
    for(int i=0;i<qts_comandos;i++){
        bytesResposta=enviaAPDU((byte)0x80, (byte)0x05, p1,
        p2,null);//busca os registros a partir do registro na posição p1, até um total de
        p2
        temp=trataRespostaImportarDados(bytesResposta);//separa os bytes
        recebidos em Strings e devolve um array bidimensional de Strings
        retorno[(1*i*MAXIMOREGISTROSAPDU)]=temp[0];
        retorno[((1*i*MAXIMOREGISTROSAPDU)+1)]=temp[1];
        retorno[((1*i*MAXIMOREGISTROSAPDU)+2)]=temp[2];
        p1=(byte) ((byte)p1+MAXIMOREGISTROSAPDU);//atualiza o valor
        de p1
    }
    return retorno;
}
```

Tabela 8: Exemplo de método para envio de comandos ao cartão

```

...
try {
    if(!terminal.isCardPresent()){
        JOptionPane.showConfirmDialog(null, "Favor, insira
o cartão");
        terminal.waitForCardPresent(0);
    }
    //conecta ao cartão utilizando o protocolo t=1
    cartao = terminal.connect("t=1");
    //abre um canal de comunicação com o cartão
    CardChannel canalComunicacao = cartao.getBasicChannel();
    //define o data do comando select
    byte [] select =
{0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01,0x00,0x00};
    //transmite o comando select
    ResponseAPDU resposta = canalComunicacao.transmit(new
CommandAPDU(0x00,0xA4,0x04,(byte)0,select));
    if(resposta.getSW()==36864){//verifica se o comando retorno
os bytes de status igual 0x9000
        if(data==null){
            resposta = canalComunicacao.transmit(new
CommandAPDU(cla,ins,p1,p2));
        }
        else{
            resposta = canalComunicacao.transmit(new
CommandAPDU(cla,ins,p1,p2,data));
        }
    }
    ...
    cartao.disconnect(true);
    ...

```

3.3.6.2 Lado Applet

Do lado *Applet*, foi utilizada a classe APDU e a interface ISO7816, ambas da API *java card*, mais especificamente do pacote

javacard.framework.

O recebimento de um APDU pelo *applet* é bastante simplificado em *java card*, o JCRE recebe o pacote e o repassa ao *applet* através de uma chamada ao método *process* do mesmo, passando como parâmetro um objeto APDU com os *bytes* recebidos. Após o recebimento do APDU, os dados recebidos são tratados com auxílio da interface ISO7816, conforme exemplo:

Tabela 9: Fragmento de código para tratamento de comandos recebidos

```
public void process(APDU apdu) throws IOException {
    byte[] cmd_apdu = apdu.getBuffer();
    byte cla = cmd_apdu[ISO7816.OFFSET_CLA];
    byte p1 = cmd_apdu[ISO7816.OFFSET_P1];
    byte p2 = cmd_apdu[ISO7816.OFFSET_P2];
    byte ins = cmd_apdu[ISO7816.OFFSET_INS];
    switch(ins){
        case INS_REGISTRA_PONTO:
            registraPonto(cmd_apdu,apdu);
            break;
        case INS_EXPORTA_REGISTROS:
            exportarDados(apdu,p1,p2,false);
            break;
        case INS_ALTERA_DADOS:
            alterarDados(apdu);
            ...
    }
```

O envio de dados pelo *applet* também é bem simples. Ele é feito através de chamada aos métodos *setOutgoing*, *setOutgoingLength* e *sendBytesLong*, todos da classe APDU.

Tabela 10: Fragmento de código para envio de respostas a partir do applet

```
...
    apdu.setOutgoing(); //define direção da transferência de dados e
    obtêm o tamanho da resposta prevista
    apdu.setOutgoingLength((short)lc); //define o tamanho atual da
    resposta
    apdu.sendBytesLong(data, (short)0, lc); //envia os bytes de um array,
    a partir da posição passada no parâmetro 2, até um total igual ao parâmetro 3
    ...
```

3.3.7 Tratamento de exceções

O suporte ao tratamento de exceções em *java card* é muito similar ao tratamento dado na linguagem *Java* (JSE), assim como em JSE, existe a classe *Throwable* que estende diretamente a classe *Object* e serve como uma raiz na hierarquia de classes de exceções.

Assim como em *Java*, em *java card* o desenvolvedor pode criar as suas próprias classes de exceções, para isso, a classe criada pelo desenvolvedor deve estender a classe *CardException* caso seja uma exceção verificada, ou seja, que é verificada durante a compilação, ou ainda estender a classe *RuntimeException*, caso seja uma exceção de tempo de execução. É importante observar que assim como em *Java*, ao estender as classes *CardException* e *RuntimeException* a classe estenderá também as classes *Exception* e *Throwable*, herdando todos seus métodos *public*, *protected* ou com nível de acesso *default*.

A tabela 11 apresenta um fragmento de código para captura de exceções. Como pode ser visto, é utilizado o mesmo mecanismo da linguagem *java* tradicional.

Tabela 11: Captura de exceções em Java Card

```
...
try{
    cifra.doFinal(apdubuf, ISO7816.OFFSET_CDATA, dataLen,
retorno, (short) 0);
} catch(Exception ex){
    ISOException.throwIt((short)(0x6F20+
((CardRuntimeException) ex).getReason()));
}
...
```

Em *java card*, o retorno de exceções para a aplicação *host* geralmente é feita através de uma chamada ao método *static throwIT* da classe *ISOException*, passando um código de erro. Muitas vezes o tratamento para *debug* tem que ser feito de forma manual, ou seja, retornando um código diferente para cada possível exceção, conforme exemplificado na tabela 12.

Tabela 12: Manipulação de exceções e retorno de erros

```
try{  
    ...  
} catch(ArithmeticException ex){  
    ISOException.throwIt((short)(0x6F00+0X01) );  
} catch(ArrayStoreException ex){  
    ISOException.throwIt((short)(0x6F00+0X02) );  
} catch(ClassCastException ex){  
    ISOException.throwIt((short)(0x6F00+0X03) );  
} catch(IndexOutOfBoundsException ex){  
    ISOException.throwIt((short)(0x6F00+0X04) );  
} catch(NegativeArraySizeException ex){  
    ISOException.throwIt((short)(0x6F00+0X05) );  
} catch(NullPointerException ex){  
    ISOException.throwIt((short)(0x6F00+0X06) );  
} catch(TransactionException ex){  
    ISOException.throwIt((short)(0x6F00+0X15) );  
} catch(CardRuntimeException ex){  
    ISOException.throwIt((short)(0x6F00+0X08) );  
} catch(RuntimeException ex){  
    ISOException.throwIt((short)(0x6F00+0X09) );  
} catch(Exception ex){  
    ISOException.throwIt((short)(0x6F00+0X0A) );  
}
```

3.3.8 Controle de acesso e privilégios através de PIN e PUK

O controle de acesso através dos códigos PIN e PUK são realizados em *java card* através da classe *OwnerPIN*, a qual implementa a interface PIN. A classe *OwnerPIN* implementa todas as funções básicas relacionadas a um PIN, como inicializar/alterar o seu valor, realizar o controle de tentativas de validação, controle sobre o bloqueio do PIN ao ultrapassar o limite de tentativas, controla também a validade do PIN apresentado para uma determinada sessão CAD, mantendo-se ativo durante a mesma, e oferecendo métodos para consultar esta validade.

A implementação da classe *OwnerPIN* protege o PIN contra ataques baseados na previsão de fluxo do programa, se a transação está em andamento, o estado interno, não deve ser condicionalmente

atualizado durante a apresentação do PIN. A classe *OwnerPIN* garante que todos as matrizes criadas durante os processos de inicialização/atualização e validação são transitórios do tipo *CLEAR_ON_RESET*, ou seja, serão excluídos ou “limpos” durante o *reset* do cartão.

Para a implementação de um PIN global do cartão, são combinadas a classes *OwnerPIN* e a interface *Shareable*. A instância *OwnerPIN* é manipulada somente pelo contexto proprietário, porém para os demais contextos, é disponibilizada uma interface publica através de uma interface *Shareable*.

No *applet* Ponto Eletrônico, a implementação do controle através do PIN, foi realizado na classe *Core*, esta classe, é a classe principal do *applet*, sendo esta a estender a classe *Applet* da API *java card*. Para dar à classe *Core* a capacidade de controle de acesso através de código PIN, foram feitas as seguintes mudanças na classe:

- criado um atributo do tipo *OwnerPIN*, o qual possui controlador de acesso do tipo *protected*, ou seja, pode ser acessado por qualquer classe dentro do mesmo pacote, além de classes de outros pacotes que estendam a classe *Core*;
- métodos de iniciação, atualização e validação do PIN, sendo estes acessados através dos códigos de instruções 0C, 0B e 0A respectivamente. Para atualizar o PIN, é necessário primeiro apresentar o PIN atual válido, através do código de instrução 0A;
- os métodos que tratam da exportação de dados, seja de forma cifrada ou não ou ainda os métodos que tratam da atualização de dados pessoais, receberam as linhas apresentadas na tabela 13, as quais verificam se o PIN já foi validado na sessão CAD atual.

Tabela 13: Verifica se PIN foi validado

```
if(!pin.isValidated()){  
    ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATI  
SFIED);  
}
```

- A primeira linha faz uma chamada ao método *isValidated()* da classe *OwnerPIN*, o qual retorna *true* se o PIN já foi validado ou *false*, caso ainda não tenha sido validado, neste caso são retornados os *bytes* de status 0x6985, o qual conforme ISO7816-4, é um código de status

padronizado para condição de uso não satisfeita.

3.3.9 Criptografia simétrica de dados em *java card*;

Em *java card*, são disponibilizadas uma variedade de classes e interfaces para implementação de operações relacionadas à segurança através dos pacotes *javacard.security* e *javacard.cripto*.

No pacote *javacard.security* são disponibilizadas interfaces que criam abstração das chaves utilizadas em diferentes algoritmos de criptografia, *DESKey*, *DSAKey*, *DSAPrivateKey*, *DSAPublicKey*, *Key*, *PrivateKey*, *PublicKey*, *RSAPrivateKey*, *RSAPrivateKey*, *RSAPublicKey* e *SecretKey*. São disponibilizadas também, classes para geração e armazenamento de chaves ou pares de chaves (*KeyBuilder* e *KeyPair*), classe base para algoritmos de *hash* (*MessageDigest*), classe para geração de número aleatórios (*RandomData*) e classe base para algoritmos de assinatura (*Signature*). Além da exceção *CryptoException*.

No pacote *javacard.cripto* é implementada a interface *KeyEncryption* e a classe *Cipher*, a qual é uma classe abstrata base para algoritmos de criptografia.

No *applet* Ponto Eletrônico, foram implementadas duas operações de criptografia simétrica. São elas: criptografar os registros armazenados no cartão e exportar à aplicação *host* e decifrar registros cifrados recebidos da aplicação *host*. A ideia básica é tentar suprir a pequena capacidade de armazenamento do cartão, com a criptografia, ou seja, como o cartão tem uma área para armazenamento bastante limitada, fazendo com que número de registros armazenados também seja limitado, disponibiliza-se uma forma de exportar os dados que estão no cartão para outros dispositivos, mantendo-lhes cifrados com uma chave simétrica, que se mantém protegida dentro do cartão. Desta forma se uma pessoa mal-intencionada obtiver os registros, ainda não poderá visualizá-los. Enquanto que ao proprietário do cartão, é oferecida uma forma de decifrar estes dados e visualizá-los em texto plano.

A implementação da chave simétrica, foi feita utilizando a interface *DESKey*, ou seja, foi criado um atributo *DESKey*, para armazenar a chave do usuário. Para gerar a chave, foi utilizado o método estático *buildKey* da classe *KeyBuilder*. O código completo de geração da chave é apresentado na tabela 14.

Tabela 14: Inicialização e geração da chave simétrica

```
...
    byte[]chave=JCSYSTEM.makeTransientByteArray((short)8,
JCSYSTEM.CLEAR_ON_RESET);
    this.chaveDES=(DESKey)
KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
KeyBuilder.LENGTH_DES,false);
    this.chaveDES.setKey(chave, (short) 0);
    try {
        cifra=Cipher.getInstance(Cipher.ALG_DES_CBC_ISO9797_M2,
false);
    } catch (CryptoException ex) {
        ISOException.throwIt((short)(0x6F00+ex.getReason() ));
    } catch (Exception ex) {
        ISOException.throwIt((short)(0x6F20+((CardRuntimeException)
ex).getReason() ));
    }
    ...
```

Para cifrar e decifrar os dados, foi utilizado o método *doFinal* da classe *Cipher*. A diferença entre os métodos de cifrar e decifrar, está na inicialização do algoritmo, a qual é realizada através do método *init* da classe *Cipher*. A inicialização do algoritmo para cifrar dados é apresentada na tabela 15. Já a tabela 16 apresenta a inicialização do algoritmo para decifrar os dados.

Tabela 15: Inicialização da cifra no modo cifragem simétrica

```
    cifra.init(this.chaveDES, Cipher.MODE_ENCRYPT);
```

Tabela 16: Inicialização da cifra no modo decifragem simétrica

```
    cifra.init(this.chaveDES, Cipher.MODE_MODE_DECRYPT);
```

Após inicializado o algoritmo, o método *doFinal* é chamado, para cifrar/decifrar os dados.

Alguns detalhes são importantes, por exemplo, caso o algoritmo esteja operando no modo *ALG_DES_CBC_ISO9797_M2*, o tamanho do bloco cifrado é 8 *bytes* maior que os dados em texto plano. Outro detalhe importante, é que ainda operando em *ALG_DES_CBC_ISO9797_M2*, os dados cifrados tem que ser de um tamanho múltiplo de 8, ou seja, o *padding* dos bytes não é realizado de forma automática. Como visto, estas particularidades são o que torna o processo de desenvolvimento de *applet* com suporte a operações criptográficas complexo, visto que estes “detalhes” não ficam claros na API *java card*. A tabela 17 apresenta um fragmento de código responsável por cifrar um *array* de *bytes*.

Tabela 17: Fragmento de código para cifrar um array de bytes com o algoritmo DES

```
protected byte[] encripta(byte [] dados){
    dados=this.pad(dados);
    cifra.init(this.chaveDES, Cipher.MODE_ENCRYPT);
    short dataLen = (short)dados.length;
    byte[] retorno;

    if(cifra.getAlgorithm()==Cipher.ALG_DES_CBC_ISO9797_M2){
        retorno = new byte[((short)(dataLen+8))];
    }
    else{
        retorno = new byte[dataLen];
    }
    try{
        cifra.doFinal(dados, (short)0, dataLen, retorno,
(short) 0);
    }catch(Exception ex){
        ISOException.throwIt((short)(0x6F20+
((CardRuntimeException) ex).getReason()));
    }
    return retorno;
}
```

3.3.10 Criptografia assimétrica de dados em *java card*;

Para exemplificar a implementação de operações criptográficas para execução embarcada no RIC, foram implementados 2 métodos, um responsável por cifrar um conjunto de *bytes* recebidos, e o segundo responsável por decifrar um conjunto de *bytes* recebidos, em ambos existe o tratamento para blocos menores que mínimo aceitável, porém em nenhum deles existe o tratamento para blocos maiores que o máximo aceito. Optou-se por não implementar este tratamento, visto que o desempenho desde procedimento realizado do lado *host*, é muito superior ao mesmo procedimento realizado do lado do cartão. Somente operações que demandem algum tipo de recurso especial, por exemplo segurança, devem ser implementadas no cartão.

Para a implementação das chaves públicas e privadas, foram utilizadas as interfaces *RSAPublicKey* e *RSAPrivateKey*, além das classes *KeyPair* e *Cipher*. Estas interfaces foram utilizadas para armazenar respectivamente as chaves pública e privada . Já as classes *KeyPair* e *Cipher* foram utilizadas respectivamente para gerar as chaves e para realizar as operações de cifragem e decifragem dos dados. Para gerar as chaves, foi utilizado o método *gemKeyPair()*, o qual gera um par de chaves assimétrica, o algoritmo utilizado e o tamanho da chave são definidos na inicialização do objeto *KeyPair* no qual foi chamado o método *gemKeyPair()*. Após gerar o par de chaves, foram utilizados os métodos *getPublic* e *getPrivate* para buscar respectivamente as chaves pública e privada do par de chaves gerados. O código de geração das chaves é apresentado na tabela 18.

Tabela 18: Inicialização e geração das chaves públicas e privadas

```

//Inicializa Chaves RSA
KeyPair parDechaves;
try{
    parDechaves = new KeyPair( KeyPair.ALG_RSA,
KeyBuilder.LENGTH_RSA_1280 );
    //2 Passo - Efetivamente gera o par de chaves
    parDechaves.genKeyPair();
    this.chavePublica = (RSAPublicKey) parDechaves.getPublic();
    this.chavePrivada = (RSAPrivateKey) parDechaves.getPrivate();
}
catch (CryptoException e) {
    switch (e.getReason()) {
        case CryptoException.ILLEGAL_USE :

            ISOException.throwIt(CryptoException.ILLEGAL_USE);break;
            case CryptoException.ILLEGAL_VALUE :

            ISOException.throwIt(CryptoException.ILLEGAL_VALUE); break;
            case CryptoException.INVALID_INIT :

            ISOException.throwIt(CryptoException.INVALID_INIT); break;
            case CryptoException.NO_SUCH_ALGORITHM :

            ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM);
            break;
            case CryptoException.UNINITIALIZED_KEY :
            ISOException.throwIt(CryptoException.UNINITIALIZED_
KEY);
            break;
            default:
            ISOException.throwIt(CryptoException.NO_SUCH_ALGO
RITHM);
            break;
        }
    }
    catch (Exception e) {
        ISOException.throwIt((short)(0x6F00+((CardException)
e.getReason()));
    }
}

```

Para cifrar e decifrar os dados, assim como nas operações criptográficas simétricas, foi utilizado o método *doFinal* da classe

Cipher. O que difere os métodos *doFinal* utilizados para cifrar e decifrar, é a sua inicialização. A inicialização do método *doFinal* para cifrar, ficou conforme trecho de código apresentado na tabela 19. Já a tabela 20 apresenta a inicialização da cifra para a decifragem dos dados.

Tabela 19: Inicialização da cifra em modo cifragem assimétrica

```
cifraRSA.init(chavePrivada, Cipher.MODE_ENCRYPT);
```

Tabela 20: Inicialização da cifra em modo decifragem assimétrica

```
cifraDES.init(this.guardiao.getChaveDES(),  
Cipher.MODE_DECRYPT);
```

A definição do algoritmo utilizado, é setado na inicialização do objeto *Cipher* utilizado para cifrar e decifrar. No exemplo, o objeto *Cipher* teve sua inicialização executada através do código apresentado na tabela 21:

Tabela 21: Inicialização do objeto Cipher com algoritmo RSA

```
try{  
    cifraRSA=Cipher.getInstance(Cipher.ALG_RSA_PKCS1,  
false);  
  
    }catch(CryptoException ex){  
        ISOException.throwIt((short)(0x6F00+ex.getReason()));  
    }catch(Exception ex){  
        ISOException.throwIt((short)(0x6F20+  
((CardRuntimeException) ex).getReason()));  
    }  
}
```

A implementação do método responsável pela cifragem, é apresentada na tabela 22.

Tabela 22: Fragmento de código para cifrar um array de bytes com algoritmo RSA

```

protected void cifraDadosChaveAssimetrica(APDU apdu,byte
[] buffer){
    //Inicia a cifra com a chave privada armazenada no cartão, em
    modo de cifragem
    cifraRSA.init(this.guardiao.getChavePrivada()),
    Cipher.MODE_ENCRYPT);
    //Verifica o número de bytes para cifragem
    short dataLen = (short) (buffer[ISO7816.OFFSET_LC] &
    0x00FF);
    //busca o tamanho chave. Este passo é importante para definir
    o número máximo de byte permitidos
    short tamanhoChave =
this.guardiao.getChavePrivada().getSize();
    //Verifica se o número de bytes recebidos é maior que o
    tamanho da chave -11. Este é o tamanho máximo aceito
    //Caso seja maior que o aceito, é retornado um erro à aplicação
    host
    if (((short)(dataLen*8)) > ((short)(tamanhoChave-11)))
    ISOException.throwIt((short)(0x6F20+(dataLen*8)));
    //inicia um array para armazenar o retorno. O tamanho do
    array é o mesmo da chave.
    byte [] retorno = new byte[(short)(tamanhoChave/8)];
    try{
        cifraRSA.doFinal(buffer, ISO7816.OFFSET_CDATA,
        dataLen, retorno, (short)0);
    }catch(CardRuntimeException ex){
        ISOException.throwIt((short)(0x6F00+0X08) );
    }catch(Exception ex){
        ISOException.throwIt((short)(0x6F00+0X0A) );
    }
    apdu.setOutgoing();
    apdu.setOutgoingLength(ok);
    apdu.sendBytesLong(retorno, (short)0, ok);
}

```

3.3.11 Outras operações criptográficas

Além das operações para cifrar e decifrar dados com criptografias simétrica e assimétrica, também foram implementados métodos para cálculo de *hash* e assinatura de dados. Para cálculo do *hash*, foi utilizado o método *doFinal()* da classe *MessageDigest*, antes porém é necessário inicializar o objeto *MessageDigest*, isto foi feito com o fragmento de código apresentado na tabela 23.

Tabela 23: Inicialização do objeto para cálculo de hash

```
MessageDigest
calculadorDeHash=MessageDigest.getInstance(MessageDigest.ALG_S
HA, false);
```

Uma vez inicializado, para calcular o *hash* de um *array* de *bytes*, basta chamar os métodos *reset()*, seguido pelo método *doFinal()*, conforme exemplo:

Tabela 24: Fragmento de código para cálculo de hash

```
protected void calculaHash(APDU apdu,byte[]buffer){
    short dataLen =(short) (buffer[ISO7816.OFFSET_LC] &
0x00FF);
    calculadorDeHash.reset();
    short bytesCalculados;
    bytesCalculados = calculadorDeHash.doFinal(buffer,
ISO7816.OFFSET_CDATA, dataLen, buffer,
ISO7816.OFFSET_CDATA);
    apdu.setOutgoing();
    apdu.setOutgoingLength(ok);
    apdu.sendBytesLong(buffer, ISO7816.OFFSET_CDATA,
bytesCalculados);
}
```

Para o método responsável por assinar um *array* de *bytes*, o qual

normalmente deverá ser um *hash*, foi utilizado um objeto da classe *Signature*, que foi inicializado através da linha de código apresentada na tabela 25

Tabela 25: Inicialização de um objeto para assinar dados

```
Signature
assinatura=Signature.getInstance(Signature.ALG_RSA_SHA_PKCSI,
false);
```

O procedimento realizado para assinar os dados, é muito semelhante ao procedimento executado nos métodos de cifragem e decifragem. Primeiro inicializa-se o objeto da classe *Signature*, depois assina-se um conjunto de *bytes* através da chamada a um método da classe *Signature*, neste caso foi utilizado o método *sign*, que recebe os mesmos parâmetros do método *doFinal* da classe *Cipher*. O código para assinar dados é apresentado na tabela 26.

Tabela 26: Fragmento de código para assinatura de dados

```
protected void assinaDados(APDU apdu,byte[]buffer){
    assinatura.init(this.guardiao.getChavePrivada(),
Signature.MODE_SIGN);
    short dataLen =(short) (buffer[ISO7816.OFFSET_LC] & 0x00FF);
    short bytesAssinados=0;
    try{
        bytesAssinados = assinatura.sign(buffer,
ISO7816.OFFSET_CDATA, dataLen, buffer, ISO7816.OFFSET_CDATA);
    }catch(Exception ex){
        ISOException.throwIt((short)(0x6F00+0X0A) );
    }
    apdu.setOutgoing();
    apdu.setOutgoingLength(ok);
    apdu.sendBytesLong(buffer, ISO7816.OFFSET_CDATA,
bytesAssinados);
}
```

4 TRABALHOS FUTUROS

O Registro de Identidade Civil (RIC) deve alcançar níveis expressivos de uso nos próximos anos, por isso se faz necessário um estudo sobre suas características de segurança, realizando testes de segurança no mesmo e realizando eventuais críticas ao documento. Este trabalho exige cartões reais do RIC para sua execução.

Como forma de continuação deste trabalho, sugere-se ainda um estudo sobre protocolos de segurança para identificação/autenticação do titular, buscando o melhor uso dos recursos disponibilizados no RIC. Para que este trabalho seja realizado, será necessário primeiro um estudo mais detalhado sobre como acessar, extrair e decodificar as informações armazenadas no RIC, utilizando cartões reais do RIC.

É importante também, a validação das informações apresentadas neste trabalho sobre o desenvolvimento de aplicações para execução embarcadas no novo documento, para isto é necessário um cartão real do RIC, onde deverão ser embarcadas aplicações desenvolvidas conforme apresentado neste trabalho, buscando validar ou refutar tais informações. Este estudo é importante, uma vez que o novo documento ainda está em desenvolvimento e não foi possível realizar tais validações com as informações existentes atualmente.

Por fim, sugere-se um estudo para para o desenvolvimento de um *applet* gerenciador de certificados de atributos para execução embarcada no novo documento, apresentado as mudanças necessárias, assim como também demonstrando que este *applet* não afeta a segurança no RIC.

5 CONCLUSÃO

O presente trabalho tinha como objetivo principal estudar o processo de desenvolvimento de aplicações para execução embarcada no RIC, porém como foi apresentado ao longo do trabalho, existem barreiras para o desenvolvimento deste tipo de aplicações. As principais barreiras são: a falta de um processo de homologação de aplicativos para execução embarcada e a capacidade armazenamento do novo documento. O processo de homologação é importante uma vez que a aplicação em questão irá compartilhar o chip de contato do RIC com a aplicação responsável por gerir os certificados e chaves do titular.

Mesmo com conhecimento das barreiras existentes e sabendo da necessidade de alterações no projeto para permitir embarcar novas aplicações ao RIC, estudou-se como seria o desenvolvimento destas aplicações, implementando e exemplificando o processo. Para isto assumiu-se algumas premissas básicas sobre o RIC, isto foi necessário, pois com a mudança no cronograma de implantação do RIC, no período de desenvolvimento deste trabalho não existiam cartões reais para testes e as documentações existentes não alcançavam este nível de detalhes.

Este trabalho também realizou um estudo sobre a tecnologias similares implantadas em outros países, onde ficou comprovado que o RIC é um documento mais completo que outros similares. Neste estudo também ficou claro a falta de uma política para permitir que os avanços trazidos pelo RIC agilizem processos, principalmente processos públicos. Como apresentado neste trabalho, outros países ao lançar os seus documentos eletrônicos, lançaram também projetos de governo digital incentivando o uso do novo documento. Esta medida não foi realizada no Brasil.

Concluindo, sugere-se um estudo mais um detalhado sobre o RIC utilizando cartões reais do mesmo, como proposto nos trabalhos Futuros.

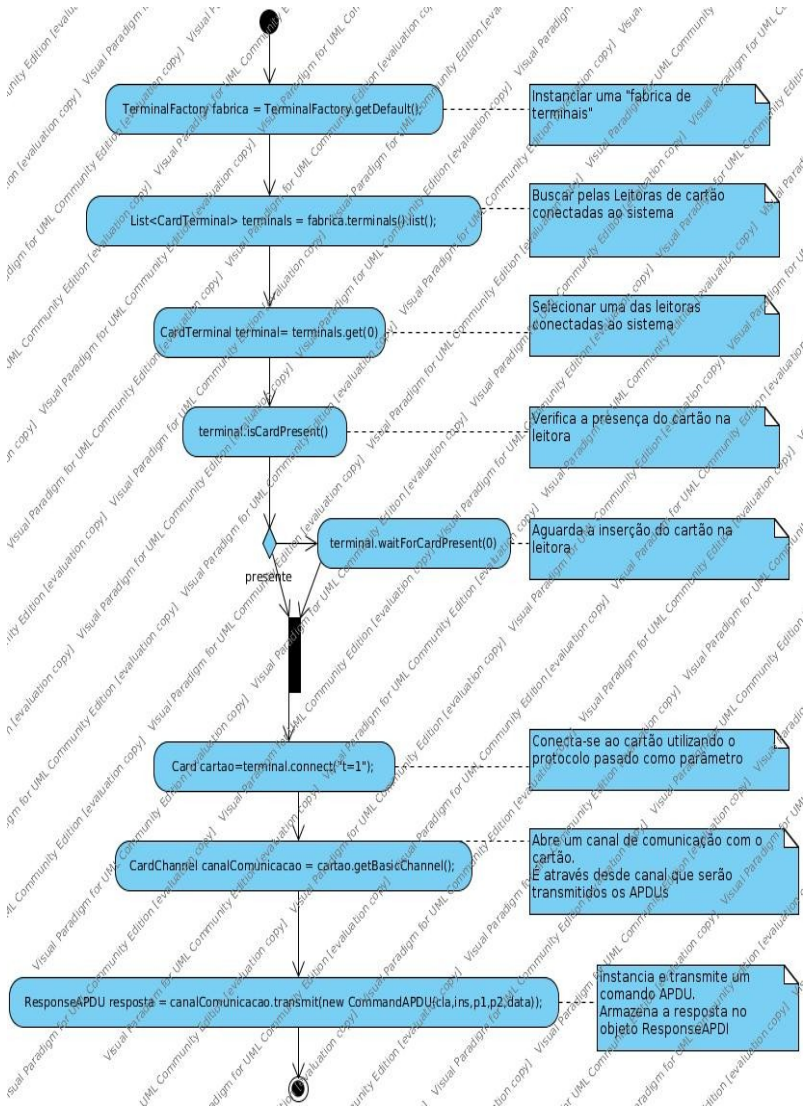
6 REFERÊNCIA

- [1] RIC – Site do ministério da justiça, Maio de 2012 disponível em : <<http://portal.mj.gov.br/ric>>
- [2] LABORATORIOS, R. S. A. PKCS# 15: Cryptographic Token Information Format.
- [3] BRASIL – Lei nº 9454, de 7 de abril de 1997, disponível em: <http://www.planalto.gov.br/ccivil_03/Leis/L9454.htm>
- [4] BRASIL – Lei nº 12058, de 13 de Outubro de 2009, disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2009/lei/112058.htm>
- [5] BRASIL - DECRETO nº 7.166, de 5 de maio de 2010, disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2010/decreto/d7166.htm>
- [6] Resolução mj nº2 de 10 de setembro de 2010, disponível em: <<http://ws.mp.mg.gov.br/biblio/informa/011013936.htm>>
- [7] International Civil Aviation Organization . ICAO doc. 9303:documentos de viagem de leitura mecânica.
- [8] BRASIL – Projeto de Lei 3860/2012, de maio de 2012, disponível em : <<http://www.camara.gov.br/proposicoesWeb/fichadetramitacao?idProposicao=544497>>
- [9] [Projeto define regras para implantação do número único de registro civil](http://www.anoreg.org.br/index.php?option=com_content&task=view&id=19313&Itemid=9) - site da Associação dos Notários e Registradores do Brasil, Maio de 2012 disponível em : <http://www.anoreg.org.br/index.php?option=com_content&task=view&id=19313&Itemid=9>
- [10] Adavalli, Surender Reddy. **Smart Card Solution: Highly secured Java Card Technology**. 2003.
- [11] CARD, Java. 2.1. 1 Virtual Machine Specification. **Sun Microsystems**, 2000.
- [12] CARD, Java. 2.1. 1 Runtime Environment (JCRE) Specification. **Sun Microsystems**, 2000.
- [13] GLOBALPLATFORM, Card Specification. Version 2.1. 1. 2003.
- [14] PC/SC Workgroup Mission, Maio de 2013, disponível em : <http://www.pcscworkgroup.com/overview/mission.php>
- [15] PC/SC Workgroup Objectives, Maio de 2013, disponível em : <http://www.pcscworkgroup.com/overview/objectives.php>
- [16] PC/SC Workgroup Membership Overview, Maio de 2013,

- disponível em :
<http://www.pcscworkgroup.com/membership/overview.php>
- [17] ISO/IEC 7816-4 Identification cards – Integrated circuit cards – part 4: Organization, security and commands for interchange. **International Standard**, 2005.
- [18] FIPS, PUB. 140-2: Security Requirements for Cryptographic Modules. **National Institute of Standards and Technology**, 2001.
- [19] Menezes, A.J., Oorschot, P.C., Vanstone, S.A., Handbook of Applied Cryptography, CRC Press, 1996, disponível em formato eletrônico em <http://www.cacr.math.uwaterloo.ca/hac>
- [20] MAGALHÃES, Paulo Sérgio; SANTOS, Henrique Dinis dos. Biometria e autenticação. 2003.
- [21] Novo Registro de Identidade Civil(RIC), Maio de 2013 disponível em :
<<http://www.brasil.gov.br/para/servicos/documentacao/conheca-o-novo-registro-de-identidade-civil-ric>>
- [21] CARTÃO DE CIDADÃO - Enquadramento, Maio de 2013 disponível em : <http://www.cartaodecidadao.pt/index.php?option=com_content&task=view&id=17&Itemid=26&lang=pt>
- [22] CARTÃO DE CIDADÃO - Objetivos, Maio de 2013 disponível em : <http://www.cartaodecidadao.pt/index.php?option=com_content&task=view&id=18&Itemid=26&lang=pt>
- [23] CARTÃO DE CIDADÃO - Características, Maio de 2013 disponível em : <http://www.cartaodecidadao.pt/index.php?option=com_content&task=view&id=19&Itemid=26&lang=pt>
- [24] DNI ELECTRÓNICO – Presentación, Maio de 2013 disponível em :
<http://www.dnielectronico.es/informacion_general.html>
- [25] DNI electrónico – Índice, Maio de 2013 disponível em :
<<http://www.dnielectronico.es/index.html>>
- [26] eID website – The eID, Maio de 2013 disponível em :
<http://eid.belgium.be/en/find_out_more_about_the_eid/the_electronic_identity_documents/the_eid/>
- [27] eID website – The kids-ID, Maio de 2013 disponível em :
<http://eid.belgium.be/en/find_out_more_about_the_eid/the_electronic_identity_documents/the_kids_id/>
- [28] Relatório de Melhores Práticas Mundias. Unidade de Coordenação da Modernização Administrativa e Agência para a Sociedade do Conhecimento.2006.
- [29] Manual de Condutas Técnicas 1 – Volume1 : Requisitos, Materiais e Documentos Técnicos para Homologação de Cartões

Criptográficos (Smart Cards) no Âmbito da ICP-Brasil Versão 3.0.
Infra-Estrutura de chaves Públicas Brasileira, 2007.

7 ANEXO A



8 ANEXO B - CÓDIGO FONTE- CLASSE CORE

```
package core;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.OwnerPIN;
import javacard.framework.Util;

/**
 * Classe principal do Applet. Esta classe recebe e encaminha todo
 * APDU recebido pelo Applet
 * @author Leonardo Malagoli da Silva
 *
 */

public class Core extends Applet {

    protected static final byte INS_ALTERA_DADOS =
(byte)0x03;
    protected static final byte INS_REGISTRA_PONTO =
(byte)0x04;
    protected static final byte INS_EXPORTA_REGISTROS =
(byte)0x05;
    protected static final byte INS_GET_NOME = (byte)0x06;
    protected static final byte INS_GET_PIS_ASSINADO =
(byte)0x07;
    protected static final byte INS_GET_RAZAO_SOCIAL =
(byte)0x08;
    protected static final byte INS_GET_CNPJ = (byte)0x09;
    protected static final byte INS_AUT_PIN = (byte)0x0A;
    protected static final byte INS_ALT_PIN = (byte)0x0B;
    protected static final byte INS_INC_PIN = (byte)0x0C;
    protected static final byte INS_EXPORTA_CIFRADOS =
(byte)0x0D;
    protected static final byte INS_DECIFRA_REGISTRO =
(byte)0x0E;
    //protected static final byte INS_CIFRA_REGISTRO =
(byte)0x0F;
```

```

        //protected static final byte INS_GEN_RSA_KEY =
(byte)0x10;
        protected static final byte INS_EXP_PRI_RSA_KEY_MOD =
(byte)0x21;
        protected static final byte INS_EXP_PRI_RSA_KEY_EXP =
(byte)0x22;
        protected static final byte INS_EXP_PUB_RSA_KEY_MOD =
(byte)0x11;
        protected static final byte INS_EXP_PUB_RSA_KEY_EXP =
(byte)0x12;
        protected static final byte INS_PUT_CERT=(byte)0x13;
        protected static final byte INS_GET_CERT=(byte)0x15;
        protected static final byte INS_BUF_DAD=(byte)0x16;

        protected static final byte TENTATIVAS_MAXIMAS_PIN =
(byte)0x03;
        protected static final byte
TAMANHO_MAXIMO_PIN=(byte)0x0A;
        protected OwnerPIN pin;

        protected static final byte CLA_Applet = (byte)0x80;
        protected Persistencia persistencia;
        protected DadosPessoais dados;
        protected Cifra cifra;
        protected static final short MAX_APDU = 127;
        protected short enviados;
        protected byte[]memoriaTem= new byte[1024];
        protected boolean temBuffer;
        protected short tamanhoBuffer;
        protected short bufferEnviados;
        protected boolean temBufferEnvio;

        private Core() {
            this.persistencia= new Persistencia();
            this.dados= new DadosPessoais();
            this.cifra= new Cifra();
            this.enviados=0;
            this.pin= new
OwnerPIN(TENTATIVAS_MAXIMAS_PIN,
TAMANHO_MAXIMO_PIN);
            temBuffer=false;

```

```

        tamanhoBuffer=0;
        temBufferEnvio=false;
    }

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

    public void process(APDU apdu) throws ISOException {
        if (selectingApplet()){

ISOException.throwIt((short)ISO7816.SW_NO_ERROR);
        }
        byte[] cmd_apdu = apdu.getBuffer();
        byte cla = cmd_apdu[ISO7816.OFFSET_CLA];
        byte p1= cmd_apdu[ISO7816.OFFSET_P1];
        byte p2= cmd_apdu[ISO7816.OFFSET_P2];
        /*
        * Realiza um operação de bits desconsiderando todos
os bits com exceção do bit 8.
        * Confere apenas se o bit 8 está setado.
        */
        byte ins = cmd_apdu[ISO7816.OFFSET_INS];
        switch(ins){
        case INS_REGISTRA_PONTO:
            registraPonto(cmd_apdu,apdu);
            break;
        case INS_EXPORTA_REGISTROS:
            exportarDados(apdu,p1,p2,false);
            break;
        case INS_ALTERA_DADOS:
            alterarDados(apdu);
            break;
        case INS_GET_NOME:
            getNome(apdu);
            break;
        case INS_GET_PIS_ASSINADO:
            getPISAssinado(apdu,cmd_apdu);
            break;
        case INS_GET_RAZAO_SOCIAL:

```

```

        getRazao(apdu);
        break;
case INS_GET_CNPJ:
    getCNPJ(apdu);
    break;
case INS_AUT_PIN:
    autenticaPIN(apdu);
    break;
case INS_ALT_PIN:
    alteraPIN(apdu,true);
    break;
case INS_INC_PIN:
    alteraPIN(apdu,false);
    break;
case INS_EXPORTA_CIFRADOS:
    exportarDados(apdu,p1,p2,true);
    break;
case INS_DECIFRA_REGISTRO:
    decifrarRegistro(apdu);
    break;
case INS_EXP_PRI_RSA_KEY_MOD:
    this.cifra.getPrivateKeyMod(apdu);
    break;
case INS_EXP_PRI_RSA_KEY_EXP:
    this.cifra.getPrivateKeyExp(apdu);
    break;
case INS_EXP_PUB_RSA_KEY_MOD:
    this.cifra.getPublicKeyMod(apdu);
    break;
case INS_EXP_PUB_RSA_KEY_EXP:
    this.cifra.getPublicKeyExp(apdu);
    break;
case INS_PUT_CERT:
    short len=cmd_apdu[ISO7816.OFFSET_LC];
    short deslocamento=Util.makeShort(p1, p2);
    this.cifra.putCertificate(cmd_apdu,
ISO7816.OFFSET_CDATA, deslocamento, len);
    break;
case INS_GET_CERT:
    this.getCertificate(apdu);
    break;

```

```

        case INS_BUF_DAD:
            this.buferizaDados(apdu);
            break;
        default:
            ISOException.throwIt(( short)
ISO7816.SW_INS_NOT_SUPPORTED);
            break;
    }
}

/**
 * Altera o PIN
 * @param apdu APDU com o novo PIN
 * @param validaPIN true caso deva ser verificado se o PIN
atual foi validado ou false caso contrario
 */
protected void alteraPIN(APDU apdu,boolean validaPIN){
    if(validaPIN){
        if(!pin.isValidated()){

ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED
);
        }
    }
    //busca tamanho do novo PIN
    byte
tamanhoPIN=(byte)apdu.setIncomingAndReceive();
    //verifica se o tamanho do novo PIN é maior que zero e
menor que o tamanho máximo aceito.
    if (tamanhoPIN<(byte)1||
tamanhoPIN>TAMANHO_MAXIMO_PIN){

        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
    }
    //Redefine PIN

    pin.update(apdu.getBuffer(),ISO7816.OFFSET_CDATA ,
tamanhoPIN);
    ISOException.throwIt(ISO7816.SW_NO_ERROR);

```

```

    }

    /**
     * Autentica o PIN
     * @param apdu APDU com o PIN
     */
    protected void autenticaPIN(APDU apdu){
        //busca tamanho do novo PIN
        byte
        tamanhoPIN=(byte)apdu.setIncomingAndReceive();
        //verifica se o tamanho do novo PIN é maior que zero e
        menor que o tamanho máximo aceito.
        if (tamanhoPIN<(byte)1||
        tamanhoPIN>TAMANHO_MAXIMO_PIN){

            ISOException.throwIt(ISO7816.SW_DATA_INVALID);
        }
        //Autentica PIN
        if ( !pin.check(apdu.getBuffer(),
        ISO7816.OFFSET_CDATA, tamanhoPIN) )

        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED
        );
    }

    /**
     * Armazena um registro de Ponto no cartão
     * @param cmd_apdu Buffer do APDU com o registro
     * @param apdu APDU
     */
    protected void registraPonto(byte cmd_apdu[],APDU apdu){
        byte lc = (byte)(cmd_apdu[ISO7816.OFFSET_LC] &
        0x00FF);

        byte p1 = cmd_apdu[ISO7816.OFFSET_P1];
        if(p1!=0x00){
            temBuffer=true;
            Util.arrayCopy(cmd_apdu,
            ISO7816.OFFSET_CDATA, memoriaTem, (short) 0, lc);
            tamanhoBuffer=(short) (tamanhoBuffer+lc);
        }else{
            Util.arrayCopy(cmd_apdu,

```

```

ISO7816.OFFSET_CDATA, memoriaTem, tamanhoBuffer, lc);
        tamanhoBuffer=(short) (tamanhoBuffer+lc);
        byte [] data=new
byte[Persistencia.TAMANHOREGISTRO];
        Util.arrayCopy(memoriaTem, (short)0, data,
(short) 0, tamanhoBuffer);
        tamanhoBuffer=0;
        temBuffer=false;
        this.persistencia.registraPonto(data);
    }

}

/**
 * Altera dados pessoais
 * @param apdu APDU com os novos dados Pessoais
 */
protected void alterarDados(APDU apdu){
    if(!pin.isValidated()){

```

```

ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED
);

```

```

    }
    byte[]retorno=dados.alteraDados(apdu.getBuffer());
    persistencia.limparDados();
    //short le = (short)retorno.length;
    apdu.setOutgoing();
    apdu.setOutgoingLength((short)retorno.length);
    apdu.sendBytesLong(retorno, (short)0, (short)retorno.length);
    }

```

```

protected void getNome(APDU apdu){
    byte[]retorno=dados.getNome();
    short le = (short)retorno.length;
    apdu.setOutgoing();
    apdu.setOutgoingLength(le);
    apdu.sendBytesLong(retorno, (short)0, le);
    }

```

```

/**

```



```

* Método Assina PIS+TIMESTAMP e envia à aplicação Host
* @param apdu APDU
* @param cmd_apdu Buffer do comando APDU contendo o
timestamp
*/
protected void getPISAssinado(APDU apdu,byte []cmd_apdu){
    if(temBuffer){
        if(!temBufferEnvio){
            byte[]pis=dados.getPIS();
            byte
lc=cmd_apdu[ISO7816.OFFSET_LC];
            byte[]times=new
byte[tamanhoBuffer];
            Util.arrayCopy(memoriaTem,
enviados, times, (short) 0, tamanhoBuffer);

            times=this.cifra.decifraDadosChaveAssimetrica(times, false);
            byte[]retorno= new byte[(short)
(pis.length+times.length)];
            Util.arrayCopy(pis, (short)0, retorno,
(short)0, (short) pis.length);
            Util.arrayCopy(times, (short)0,
retorno, (short) pis.length, (short) times.length);
            byte [] aux =
this.cifra.assinaDados(retorno,false);
            short le = (short)aux.length;
            if(le>Core.MAX_APDU){
                temBufferEnvio=true;
                temBuffer=true;
                tamanhoBuffer=(short)(le-
Core.MAX_APDU);

                bufferEnviados=(short)0;
                Util.arrayCopy(aux,
Core.MAX_APDU, memoriaTem, (short)0, tamanhoBuffer);
                le=Core.MAX_APDU;
            }
            apdu.setOutgoing();
            apdu.setOutgoingLength(le);
            apdu.sendBytesLong(aux, (short)0, le);
        }else{
            short le;

```

```

short off=0;
if(tamanhoBuffer>Core.MAX_APDU)
{
    le=Core.MAX_APDU;
    tamanhoBuffer=(short)
(tamanhoBuffer-Core.MAX_APDU);
    off=bufferEnviados;
    bufferEnviados=(short)
(bufferEnviados+le);
} else {
    le=tamanhoBuffer;
    tamanhoBuffer=0;
    off=bufferEnviados;
    bufferEnviados=0;
    temBuffer=false;
    temBufferEnvio=false;
}
    apdu.setOutgoing();
    apdu.setOutgoingLength(le);
    apdu.sendBytesLong(memoriaTem, off, le);
}
}

```

```

protected void getRazao(APDU apdu){
    byte[]retorno=dados.getRazaoSocial();
    short le = (short)retorno.length;
    apdu.setOutgoing();
    apdu.setOutgoingLength(le);
    apdu.sendBytesLong(retorno, (short)0, le);
}

```

```

protected void getCNPJ(APDU apdu){
    byte[]retorno=dados.getCNPJ();
    short le = (short)retorno.length;
    apdu.setOutgoing();
    apdu.setOutgoingLength(le);
    apdu.sendBytesLong(retorno, (short)0, le);
}

```

```

protected void getCertificate(APDU apdu){
    byte [] cmd_apdu=apdu.getBuffer();
    byte p1 = cmd_apdu[ISO7816.OFFSET_P1];
    if(p1==0x00){
        enviados=0;
    }
    byte[]certificadoTitular=this.cifra.getCertificate();

    //verifica quantos byte já foram enviados e quanto falta
enviar
    short restantes = (short)(certificadoTitular.length-
enviados);

    //verificar a necessidade de encadear novos comandos
boolean encadear= restantes>Core.MAX_APDU;
short tamanhoEnvio = encadear ? Core.MAX_APDU :
restantes;

    apdu.setOutgoing();
    apdu.setOutgoingLength(tamanhoEnvio);
    apdu.sendBytesLong(certificadoTitular, enviados, tamanhoEnvio);

    //verifica e trata o encadeamento de novos comandos
    if(encadear){
        enviados+=tamanhoEnvio;
        //ISOException.throwIt((short)24864);//Indicator de que há
mais bytes a serem enviados
    }
    else{
        enviados=0;
    }

}

/**
 * Exporta um conjunto de registros do cartão.
 * Os dados exportados podem ou não ser cifrados.
 * @param apdu APDU

```

```

    * @param p1 Primeiro registro a ser exportado
    * @param p2 Quantidade de registros que será exportado
    * @param encripta true para cifrar os registros exportados
    */
    protected void exportarDados(APDU apdu,byte p1,byte
p2,boolean encripta){
        if(!pin.isValidated()){

ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED
);
        }
        short lc=0;
        byte [] resp=new byte[0];
        if(!temBuffer){
            resp = this.persistencia.exportarDados(p1,
(byte)0x01);
            enviados=0;
            if(encripta){
                //ISOException.throwIt(ISO7816.SW_
CONDITIONS_NOT_SATISFIED);
                resp = this.cifra.encripta(resp,false);
            }
            if(resp.length>MAX_APDU){
                temBuffer=true;
                tamanhoBuffer=(short) (resp.length-
MAX_APDU);
                Util.arrayCopy(resp, MAX_APDU,
memoriaTem, (short)0, tamanhoBuffer);
                lc=MAX_APDU;
            }else{
                lc=(short) resp.length;
            }
        }else{
            if(tamanhoBuffer>MAX_APDU){
                lc=MAX_APDU;
                tamanhoBuffer=(short)
(tamanhoBuffer-MAX_APDU);
                enviados=(short)
(enviados+MAX_APDU);
            }else{

```

```

        lc=tamanhoBuffer;
        temBuffer=false;
        tamanhoBuffer=0;
    }
    resp=new byte[lc];
    Util.arrayCopy(memoriaTem, enviados, resp,
(short)0, lc);
    }
    apdu.setOutgoing();
    apdu.setOutgoingLength(lc);
    apdu.sendBytesLong(resp, enviados,lc);
}

/**
 * Bufferiza um conjunto de dados
 * @param apdu APDU com os dados a serem bufferizados
 */
protected void buferizaDados(APDU apdu){
    if(!temBuffer){
        temBuffer=true;
        tamanhoBuffer=0;
        enviados=0;
    }
    byte[]cmd_apdu=apdu.getBuffer();
    byte lc = cmd_apdu[ISO7816.OFFSET_LC];

    Util.arrayCopy(cmd_apdu,
ISO7816.OFFSET_CDATA, memoriaTem, tamanhoBuffer, lc);
    tamanhoBuffer=(short) (tamanhoBuffer+lc);
    byte[]retorno=new byte[2];
    retorno[0]=lc;
    retorno[1]=(byte) tamanhoBuffer;
    apdu.setOutgoing();
    apdu.setOutgoingLength((short)2);
    apdu.sendBytesLong(retorno, (short)0, (short)2);
}

/**
 * Decifra um conjunto de bytes previamente bufferizados
 * @param apdu APDU
 */

```

```

protected void decifrarRegistro(APDU apdu){
    byte[]retorno=new byte[0];
    short lc=0;
    if(temBufferEnvio){
        if(tamanhoBuffer>MAX_APDU){
            lc=MAX_APDU;
            apdu.setOutgoing();
            apdu.setOutgoingLength(lc);
            apdu.sendBytesLong(memoriaTem, enviados, lc);
            enviados+=lc;
            tamanhoBuffer=(short) (tamanhoBuffer-lc);
        }else {
            lc=(short)tamanhoBuffer;
            temBuffer=false;
            temBufferEnvio=false;
            tamanhoBuffer=0;
            apdu.setOutgoing();
            apdu.setOutgoingLength(lc);
            apdu.sendBytesLong(memoriaTem, enviados, lc);
            enviados=0;
        }
    }else {
        if(temBuffer){
            byte[]temp=new byte[tamanhoBuffer];
            Util.arrayCopy(memoriaTem, (short)0,
temp, (short)0, tamanhoBuffer);
            temBuffer=false;
            tamanhoBuffer=0;
            retorno=this.cifra.decripta(temp,
false);

            //retorno=temp;
            if(retorno.length>MAX_APDU){
                lc=MAX_APDU;
                temBuffer=false;
                temBufferEnvio=true;
                tamanhoBuffer=(short)
(retorno.length);
                Util.arrayCopy(retorno,
(short)0, memoriaTem, (short)0, tamanhoBuffer);
                enviados=0;
                apdu.setOutgoing();

```

```

        apdu.setOutgoingLength(lc);
        apdu.sendBytesLong(memoriaTem,
enviados, lc);

        enviados+=lc;
        tamanhoBuffer=(short) (tamanhoBuffer-
lc);

        } else {
            lc=(short)retorno.length;
            apdu.setOutgoing();

        }

        apdu.setOutgoingLength((short)retorno.length);
        apdu.sendBytesLong(retorno, (short)0,
(short)lc);
    }
}

}

}

}

/**
 * Compara dois arrays
 * @param a Primeiro Array
 * @param b Segundo Array
 * @return true caso os dois arrays sejam iguais ou false caso
nã sejam
 */
protected boolean comparaArray(byte[] a, byte[] b){
    byte aSize =(byte) a.length;
    byte bSize=(byte) b.length;
    if(aSize==bSize){
        for(short i=0;i<a.length;i++){
            if(a[i]!=b[i]){
                return false;
            }
        }
        return true;
    } else {
        return false;
    }
}
}
}

```

9 ANEXO C- CÓDIGO FONTE – CLASSE DADOS PESSOAIS

```
package core;
```

```
import javacard.framework.ISO7816;
```

```
import javacard.framework.Util;
```

```
/**
```

```
 * Classe responsável por armazenar e gerir os dados pessoais do titular
```

```
 * @author Leonardo Malagoli da Silva
```

```
 *
```

```
 */
```

```
public class DadosPessoais {
```

```
    protected byte[] nome;
```

```
    protected byte[] PIS;
```

```
    protected byte[] razaoSocial;
```

```
    protected byte[] CNPJ;
```

```
    protected DadosPessoais() {
```

```
        this.nome=new byte[0];
```

```
        this.PIS=new byte[0];
```

```
        this.razaoSocial=new byte[0];
```

```
        this.CNPJ=new byte[0];
```

```
    }
```

```
    protected DadosPessoais( byte[]nomeAux, byte[]PISAux,  
byte[]razaoAux, byte[]CNPJAux){
```

```
        this.nome=nomeAux;
```

```
        this.PIS=PISAux;
```

```
        this.razaoSocial=razaoAux;
```

```
        this.CNPJ=CNPJAux;
```

```
    }
```

```
    protected void
```

```
alteraDados(byte[]nomeAux,byte[]PISAux,byte[]razaoAux,byte[]CNPJ  
Aux){
```

```
        this.nome=nomeAux;
```

```
        this.PIS=PISAux;
```

```
        this.razaoSocial=razaoAux;
```

```
        this.CNPJ=CNPJAux;
```

```
    }
```



```

/**
 * Altera os dados pessoais armazenados no cartão
 * @param dados Novos dados
 * @return Novo nome
 */
protected byte[] alteraDados(byte[] dados){
    byte cont=0;
    short contnome=0;
    short contPIS=0;
    short contRazao=0;
    short contCNPJ=0;
    //O separador de campos dentro da String de dados é o
    carácter ; codificado ISO-8859-1.
    for(short
i=ISO7816.OFFSET_CDATA;i<dados.length;i++){
        if(cont==0){
            if(dados[i]==59){
                contnome=i;
                cont++;
            }
        }
        else if(cont==1){
            if(dados[i]==59){
                contPIS=i;
                cont++;
            }
        }
        }else if(cont==2){
            if(dados[i]==59){
                contRazao=i;
                cont++;
            }
        }
        }else if(cont==3){
            if(dados[i]==59){
                contCNPJ=i;
                cont++;
            }
        }
    }
    short tamanhoNome=(short) (contnome-
ISO7816.OFFSET_CDATA);

```

```

        this.nome=new byte[tamanhoNome];
        short tamanhoPIS=(short)(contPIS-1-contnome);
        this.PIS=new byte[tamanhoPIS];
        short tamanhoRazao=(short)(contRazao-1-contPIS);
        this.razaoSocial=new byte[tamanhoRazao];
        short tamanhoCNPJ=(short)(contCNPJ-1-contRazao);
        this.CNPJ=new byte[tamanhoCNPJ];
        Util.arrayCopy(dados, ISO7816.OFFSET_CDATA,
this.nome,(short)0 , tamanhoNome);
        Util.arrayCopy(dados, (short)(contnome+1), this.PIS,
(short)0 , tamanhoPIS);
        Util.arrayCopy(dados, (short)(contPIS+1),
this.razaoSocial,(short)0 , tamanhoRazao);
        Util.arrayCopy(dados, (short)(contRazao+1),
this.CNPJ,(short)0 , tamanhoCNPJ);
        return this.nome;
    }

    public byte[] getNome(){
        return this.nome;
    }

    public byte[] getPIS(){
        return this.PIS;
    }

    public byte[] getRazaoSocial(){
        return this.razaoSocial;
    }

    public byte[] getCNPJ(){
        return this.CNPJ;
    }
}

```

10 ANEXO D – CÓDIGO FONTE – CLASSE FUNCOESCRIPTOGRAFICASAUXILIARES

```
package core;
```

```
import javacard.framework.ISO7816;  
import javacard.framework.ISOException;  
import javacard.framework.JCSystem;  
import javacard.framework.Util;
```

```
/**
```

```
 * Classe auxiliar com funções criptográficas auxiliares
```

```
 * @author Leonardo Malagoli da Silva
```

```
 *
```

```
 */
```

```
public class FuncoesCriptograficasAuxiliares {
```

```
    /**
```

```
     * Verifica se é necessário e se for realiza o padding de um  
conjunto que serão cifrados
```

```
     * @param message dados que será realizado o padding
```

```
     * @return dados com o padding
```

```
     */
```

```
    public byte[] pad(byte[] message) {
```

```
        short len = (short)message.length;
```

```
        short novoTamanho = (short)(len + (8 - (len % 8)));
```

```
        //short novoTamanho=(short)48;
```

```
        byte[] padded= new byte[novoTamanho];
```

```
        // ISOException.throwIt((short)(0x6F20) );
```

```
        Util.arrayCopy(message, (short)0, padded, (short)0,(short) len);
```

```
        padded[(short)len] = (byte) 0x80;
```

```
        return padded;
```

```
    }
```

```
    /**
```

```
     * Remove o padding da mensagem
```

```
     * @param message dados com o padding
```

```
     * @return dados sem padding
```

```
     */
```

```
    public byte[] removePad(byte[] message) {
```

```
        short len = (short)message.length;
```

```
        short posicao=0;
```

```

        for(short i = (short)(len-1);i>=0;i--){
            if(message[i]==(byte)0x80){
                posicao=i;
                i=0;
            }
        }
        //short novoTamanho=(short)48;
        byte[] retorno= new byte[posicao];
        // ISOException.throwIt((short)(0x6F20) );
        Util.arrayCopy(message, (short)0, retorno, (short)0,(short)
posicao);
        return retorno;
    }

    /**
     * Trata os bytes de um comando APDU retornando somente os
bytes do campo DATA
     * @param apduBuffer Buffer de um comando APDU
     * @return DATA do comando APDU
     */
    protected byte[]trataAPDU(byte[]apduBuffer){
        short dataLen = (short)
(apduBuffer[ISO7816.OFFSET_LC] & 0x00FF);

        byte[]retorno=JCSsystem.makeTransientByteArray(dataLen,
JCSsystem.CLEAR_ON_DESELECT);
        Util.arrayCopy(apduBuffer,
ISO7816.OFFSET_CDATA, retorno, (short)0, dataLen);
        return retorno;
    }
}

```

11 ANEXO E – CÓDIGO FONTE – CLASSE GUARDIAODACHAVE

```
package core;
```

```
import javacard.framework.CardException;  
import javacard.framework.CardRuntimeException;  
import javacard.framework.ISO7816;  
import javacard.framework.ISOException;  
import javacard.framework.JCSystem;  
import javacard.framework.Util;  
import javacard.security.CryptoException;  
import javacard.security.DESKey;  
import javacard.security.KeyBuilder;  
import javacard.security.KeyPair;  
import javacard.security.PrivateKey;  
import javacard.security.PublicKey;  
import javacard.security.RSAPrivateKey;  
import javacard.security.RSAPublicKey;  
import javacardx.crypto.Cipher;
```

```
/**
```

```
 * Classe responsável por armazenar as chaves e certificados do titular
```

```
 * @author Leonardo Malagoli da Silva
```

```
 *
```

```
 */
```

```
public class GuardiaoDaChave {  
    private DESKey chaveDES;    //declara uma chave DES  
    utilizada para cifrar o objeto;  
    private RSAPublicKey chavePublica;  
    private RSAPrivateKey chavePrivada;  
    private byte[] certificadoTitular;  
    public static short MAX_SIZE_CERT=2048;  
  
    public GuardiaoDaChave(){  
  
        //Inicializa Chave DES  
        //declara um array transitório para armazenar  
    temporariamente a chave gerada  
  
        byte[]chave=JCSystem.makeTransientByteArray((short)8,  
JCSystem.CLEAR_ON_RESET);
```

```

        //cria um chave não inicializada
        this.chaveDES=(DESKey)
KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
KeyBuilder.LENGTH_DES,false);
        //Define chave
        this.chaveDES.setKey(chave, (short) 0);

        //Inicializa Chaves RSA
        KeyPair parDechaves;
        try{
                parDechaves = new
KeyPair( KeyPair.ALG_RSA, KeyBuilder.LENGTH_RSA_1024 );
                parDechaves.genKeyPair();
                this.chavePublica = (RSAPublicKey)
parDechaves.getPublic();
                this.chavePrivada = (RSAPrivateKey)
parDechaves.getPrivate();
        }
        catch (CryptoException e) {
                switch (e.getReason()) {
                        case CryptoException.ILLEGAL_USE :
ISOException.throwIt(CryptoException.ILLEGAL_USE ); break;
                        case CryptoException.ILLEGAL_VALUE :
ISOException.throwIt(CryptoException.ILLEGAL_VALUE ); break;
                        case CryptoException.INVALID_INIT :
ISOException.throwIt(CryptoException.INVALID_INIT ); break;
                        case
CryptoException.NO_SUCH_ALGORITHM :
ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM );
break;
                                case CryptoException.UNINITIALIZED_KEY
: ISOException.throwIt(CryptoException.UNINITIALIZED_KEY );
break;
                                default:
ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM );
break;
                }
        }
        catch (Exception e) {
                ISOException.throwIt((short)(0x6F00+
((CardException) e).getReason() ));

```

```

        }
        certificadoTitular= new byte[MAX_SIZE_CERT];
    }

    protected RSAPublicKey getChavePublica() {
        return chavePublica;
    }

    protected void setChavePublica(RSAPublicKey chavePublica)
    {
        this.chavePublica = chavePublica;
    }

    protected RSAPrivateKey getChavePrivada() {
        return chavePrivada;
    }

    protected void setChavePrivada(RSAPrivateKey chavePrivada)
    {
        this.chavePrivada = chavePrivada;
    }

    protected DESKey getChaveDES() {
        return chaveDES;
    }

    protected void setChaveDES(DESKey chaveDES) {
        this.chaveDES = chaveDES;
    }

    /**
     * Armazena um certificado na memória
     * No primeiro envio de bytes do novo certificado, os dois
    primeiros bytes indicam o número total de bytes do certificado
     * @param source array de byte contendo o certificado
     * @param sourceOff posição inicial do certificado no source
     * @param len tamanho do certificado em bytes
     */
    protected void putCertificate(byte[] source,short
    sourceOff,short deslocamentoDestino, short len ){

```

```

        if(deslocamentoDestino==0){
            short
tamanhoNovoCertificado=Util.getShort(source,
ISO7816.OFFSET_CDATA);
            certificadoTitular= new
byte[tamanhoNovoCertificado];
        }
        if ((short) (len+deslocamentoDestino) >
MAX_SIZE_CERT) {

            ISOException.throwIt(ISO7816.SW_FILE_FULL);
        }
        if(deslocamentoDestino==0){
            Util.arrayCopy(source, (short)(sourceOff+2),
certificadoTitular, (short)0, (short)(len-2));
        }else{
            Util.arrayCopy(source, sourceOff, certificadoTitular,
deslocamentoDestino, len);
        }

    }

/**
 * Busca um certificado
 */
protected byte[] getCertificate(){
    return this.certificadoTitular;
}
}

```


12 ANEXO F – CÓDIGO FONTE - PERSISTENCIA

```
package core;
```

```
import javacard.framework.ISO7816;  
import javacard.framework.ISOException;  
import javacard.framework.Util;
```

```
/**
```

```
 * Classe responsável por armazenar e gerir os dados mantidos pelo  
 * applet. Mais especificamente os registros de pontos.
```

```
 * @author Leonardo Malagoli da Silva
```

```
 *
```

```
 */
```

```
public class Persistencia {
```

```
    //protected byte [][] registros;
```

```
    protected short pontProximoRegistro;
```

```
    static final short TAMANHOLISTA=30;
```

```
    static final short TAMANHOREGISTRO=192;
```

```
    //static final byte TAMANHOPIS=12;
```

```
    protected byte [] memoria;
```

```
    protected Persistencia(){
```

```
        this.memoria=new
```

```
byte[TAMANHOLISTA*TAMANHOREGISTRO];
```

```
        this.pontProximoRegistro=0;
```

```
    }
```

```
    /**
```

```
     * Armazena um registro de Ponto no cartão
```

```
     * @param registro Registro de Ponto
```

```
     */
```

```
    protected void registraPonto(byte[] registro){
```

```
        Util.arrayCopy(registro,
```

```
(short)0,memoria,this.pontProximoRegistro,(short) registro.length);
```

```
        this.pontProximoRegistro=(short)
```

```
(this.pontProximoRegistro+TAMANHOREGISTRO);
```

```
if(pontProximoRegistro==(TAMANHOLISTA*TAMANHOREGISTR  
O)){
```

```
    this.pontProximoRegistro=0;
```

```

        }
    }

    /**
     * Limpa os dados da memória
     */
    protected void limparDados(){
        this.memoria=new
byte[TAMANHOLISTA*TAMANHOREGISTRO];
        this.pontProximoRegistro=0;
    }

    /**
     * Exporta um conjunto de registros de Ponto
     * @param p1 Posição inicial dos registros que serão
exportados
     * @param p2 Quantidade de registros que serão exportados
     * @return registros entre P1 e o limite definido em p2
     */
    protected byte[] exportarDados(byte p1,byte p2){

        byte [] resp= new byte[(short)
(p2*TAMANHOREGISTRO)];
        Util.arrayCopy(memoria,(short)
(p1*TAMANHOREGISTRO),resp,(short)0,(short)
(p2*TAMANHOREGISTRO));
        return resp;
    }

}

```

13 ANEXO G- CÓDIGO FONTE – CLASSE CIFRA

```
package core;
```

```
import javacard.framework.APDU;  
import javacard.framework.APDUException;  
import javacard.framework.CardRuntimeIOException;  
import javacard.framework.ISO7816;  
import javacard.framework.ISOException;  
import javacard.framework.JCSystem;  
import javacard.framework.PINException;  
import javacard.framework.SystemException;  
import javacard.framework.TransactionException;  
import javacard.framework.Util;  
import javacard.security.CryptoException;  
import javacard.security.MessageDigest;  
import javacard.security.Signature;  
import javacardx.crypto.Cipher;
```

```
/**
```

```
 * Classe com funções criptográficas  
 * @author Leonardo Malagoli da Silva  
 *  
 */
```

```
public class Cifra {
```

```
    protected GuardiaoDaChave guardiao;  
    protected FuncoesCriptograficasAuxiliares auxiliar;  
    protected Cipher cifraDES;  
    protected Cipher cifraRSA;  
    protected MessageDigest calculadorDeHash;  
    protected Signature assinatura;
```

```
    public Cifra() {
```

```
        guardiao = new GuardiaoDaChave();  
        auxiliar = new FuncoesCriptograficasAuxiliares();  
        try {
```

```
            cifraDES =
```

```
Cipher.getInstance(Cipher.ALG_DES_CBC_ISO9797_M2, false);
```

```
            cifraRSA =
```

```
Cipher.getInstance(Cipher.ALG_RSA_PKCS1, false);
```

```
            calculadorDeHash =
```

```

MessageDigest.getInstance(MessageDigest.ALG_SHA,
                           false);
        assinatura =
Signature.getInstance(Signature.ALG_RSA_SHA_PKCS1,
                     false);

        } catch (CryptoException ex) {
            ISOException.throwIt((short) (0x6F00 +
ex.getReason()));
        } catch (Exception ex) {
            ISOException.throwIt((short) (0x6F20 +
((CardRuntimeException) ex)
                                   .getReason()));
        }
    }

/**
 * Cifra um conjunto de bytes com a chave Simétrica
armazenada no cartão.
 * Este método realiza o padding automaticamente.
 * @param dados Conjunto de dados que serão cifrados
 * @param tratarAPDU true caso o parâmetro dados seja um
comando APDU ou false caso contenha somente os dados a serem
cifrados
 * @return um array de bytes com os dados cifrados.
 */
protected byte[] encripta(byte[] dados, boolean tratarAPDU) {
    if (tratarAPDU) {
        dados = auxiliar.trataAPDU(dados);
    }
    if((dados.length%8)!=0){
        dados = auxiliar.pad(dados);// realiza o
"Enchimento" dos bytes
    }
    byte[] retorno;
    cifraDES.init(this.guardiao.getChaveDES(),
Cipher.MODE_ENCRYPT);// inicializa

```

// a

```

        // cifra
        short dataLen = (short) dados.length;
        if (cifraDES.getAlgorithm() ==
Cipher.ALG_DES_CBC_ISO9797_M2) {
            retorno = new byte[((short) (dataLen + 8))];

        } else {
            retorno = new byte[dataLen];
        }
        // ENCRYPT INCOMING BUFFER
        try {
            cifraDES.doFinal(dados, (short) 0, dataLen,
retorno, (short) 0);
        } catch (Exception ex) {
            ISOException.throwIt((short) (0x6F20 +
((CardRuntimeException) ex)
                                .getReason()));
        }
        return retorno;
    }

/**
 * Cifra um conjunto de bytes com a chave Simétrica
armazenada no cartão e envia os dados à aplicação Host
 * Este método NÃO realiza o padding dos dados.
 * @param apdu APDU com os dados a serem cifrados
 */
    protected void encriptaSemEnchimento(APDU apdu) {
        cifraDES.init(this.guardiao.getChaveDES(),
Cipher.MODE_ENCRYPT);
        short dataLen = apdu.setIncomingAndReceive();
        byte[] apdubuf = new byte[dataLen];//
JCSysm.makeTransientByteArray(dataLen,

                                // JCSysm.CLEAR_ON_DESELECT);
        apdubuf = apdu.getBuffer();
        byte[] retorno;
        if (cifraDES.getAlgorithm() ==
Cipher.ALG_DES_CBC_ISO9797_M2) {
            retorno = new byte[((short) (dataLen + 8))];

```

```

        } else {
            retorno = new byte[dataLen];
        }
        // ENCRYPT INCOMING BUFFER
        try {
            cifraDES.doFinal(apdubuf,
ISO7816.OFFSET_CDATA, dataLen, retorno,
                (short) 0);
        } catch (Exception ex) {
            ISOException.throwIt((short) (0x6F20 +
((CardRuntimeException) ex)
                .getReason()));
        }

        apdu.setOutgoing();
        apdu.setOutgoingLength((short) retorno.length);
        apdu.sendBytesLong(retorno, (short) 0, (short)
retorno.length);
    }

/**
 * Decifra um conjunto de dados com a chave simétrica
armazenada no cartão
 * @param dados Conjunto de dados que serão decifrados
 * @param tratarAPDU true caso o parâmetro dados seja um
comando APDU ou false caso contenha somente os dados a serem
decifrados
 * @return um array de bytes com os dados decifrados
 */
protected byte[] decripta(byte[] dados, boolean tratarAPDU) {
    if (tratarAPDU) {
        dados = auxiliar.trataAPDU(dados);
    }
    cifraDES.init(this.guardiao.getChaveDES(),
Cipher.MODE_DECRYPT);
    short dataLen = (short)dados.length;
    // byte[] retorno =
JCSysyem.makeTransientByteArray(dataLen,JCSysyem.CLEAR_ON_D
ESELECT);
    byte [] retorno = new byte[dataLen];
    // Decrypta o buffer recebido

```

```

        try {
            cifraDES.doFinal(dados, (short)0, dataLen,
returno,(short) 0);
        } catch (ArithmeticException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X01));
        } catch (ArrayStoreException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X02));
        } catch (ClassCastException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X03));
        } catch (IndexOutOfBoundsException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X04));
        } catch (NegativeArraySizeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X05));
        } catch (NullPointerException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X06));
        } catch (SecurityException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X07));
        } catch (APDUException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X10));
        } catch (CryptoException ex) {
            switch (ex.getReason()) {
                case CryptoException.ILLEGAL_USE:
                    ISOException.throwIt((short)
dados.length);
                    break;
                case CryptoException.ILLEGAL_VALUE:
                    ISOException.throwIt(CryptoException.ILLEGAL_VALUE);
                    break;
                case CryptoException.INVALID_INIT:
                    ISOException.throwIt(CryptoException.INVALID_INIT);
                    break;
            }
        }
    }
}

```

```

        case
CryptoException.NO_SUCH_ALGORITHM:

ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM);
        break;
        case
CryptoException.UNINITIALIZED_KEY:

ISOException.throwIt(CryptoException.UNINITIALIZED_KEY);
        break;
        default:

ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM);
        break;
    }
    } catch (ISOException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X12));
    } catch (PINException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X13));
    } catch (SystemException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X14));
    } catch (TransactionException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X15));
    } catch (CardRuntimeException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X08));
    } catch (RuntimeException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X09));
    } catch (Exception ex) {
        ISOException.throwIt((short) (0x6F00 +
0X0A));
    }
    //retorno = auxiliar.removePad(retorno);

```



```

        return retorno;
    }

    /**
     * Decifra um conjunto de bytes com a chave privada do cartão
     e envia à aplicação Host
     * @param apdu APDU contendo os dados a serem decifrados
     */
    protected void decriptaRSA(APDU apdu) {
        byte a[] = apdu.getBuffer();
        short byteRead = (short)
(apdu.setIncomingAndReceive());
        cifraRSA.init(this.guardiao.getChavePrivada(),
Cipher.MODE_DECRYPT);
        short cyphertext = cifraRSA.doFinal(a, (short)
ISO7816.OFFSET_CDATA,
        byteRead, a, (short)
ISO7816.OFFSET_CDATA);
        // Send results
        apdu.setOutgoing();
        apdu.setOutgoingLength((short) cyphertext);
        apdu.sendBytesLong(a, (short)
ISO7816.OFFSET_CDATA, (short) cyphertext);
    }

    /**
     * Cifra um conjunto de dados com a chave pública assimétrica
     presente no cartão e envia os dados à aplicação Host
     * @param apdu APDU com os dados recebidos
     * @param buffer Buffer do apdu
     */
    protected void cifraDadosChaveAssimetrica(APDU apdu,
byte[] buffer) {
        // Inicia a cifra com a chave privada armazenada no
cartão, em modo de
        // cifragem
        cifraRSA.init(this.guardiao.getChavePrivada(),
Cipher.MODE_ENCRYPT);
        // Verifica o número de bytes para cifragem
        short dataLen = (short) (buffer[ISO7816.OFFSET_LC]

```

```

& 0x00FF);
// busca o tamanho chave. Este passo é importante para
definir o número
// máximo de byte permitidos
short tamanhoChave =
this.guardiao.getChavePrivada().getSize();
// Verifica se o número de bytes recebidos é maior que
o tamanho da chave
// -11. Este é o tamanho máximo aceito
// Caso seja maior que o aceito, é retornado um erro à
aplicação host
if (((short) (dataLen * 8)) > ((short) (tamanhoChave -
11)))
ISOException.throwIt((short) (0x6F20 +
(dataLen * 8)));
// inicia um array para armazenar o retorno. O tamanho
do array é o
// mesmo da chave.
byte[] retorno = new byte[(short) (tamanhoChave / 8)];
short ok = (short) 0;
try {
ok = cifraRSA.doFinal(buffer,
ISO7816.OFFSET_CDATA, dataLen,
retorno, (short) 0);
} catch (ArithmeticException ex) {
ISOException.throwIt((short) (0x6F00 +
0X01));
} catch (ArrayStoreException ex) {
ISOException.throwIt((short) (0x6F00 +
0X02));
} catch (ClassCastException ex) {
ISOException.throwIt((short) (0x6F00 +
0X03));
};
} catch (IndexOutOfBoundsException ex) {
ISOException.throwIt((short) (0x6F00 +
0X04));
} catch (NegativeArraySizeException ex) {
ISOException.throwIt((short) (0x6F00 +
0X05));

```

```

        } catch (NullPointerException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X06));
        } catch (SecurityException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X07));
        } catch (APDUException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X10));
        } catch (CryptoException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X11));
        } catch (ISOException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X12));
        } catch (PINException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X13));
        } catch (SystemException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X14));
        } catch (TransactionException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X15));
        } catch (CardRuntimeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X08));
        } catch (RuntimeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X09));
        } catch (Exception ex) {
            ISOException.throwIt((short) (0x6F00 +
0X0A));
        }
        apdu.setOutgoing();
        apdu.setOutgoingLength(ok);
        apdu.sendBytesLong(retorno, (short) 0, ok);
    }

```

/**

* Decifra um conjunto de bytes com a chave privada

assimétrica presente no cartão.

```
* @param dados Dados que serão decifrados
* @param tratarAPDU true caso o parâmetro dados seja um
comando APDU ou false caso o mesmo contenha somente os bytes
cifrados
* @return um array de bytes com os dados decifrados
*/
protected byte [] decifraDadosChaveAssimetrica(byte[]
dados,boolean tratarAPDU) {
    if (tratarAPDU) {
        dados = auxiliar.trataAPDU(dados);
    }
    cifraRSA.init(this.guardiao.getChavePrivada(),
Cipher.MODE_DECRYPT);
    short dataLen = (short) dados.length;
    short ok = 0;
    byte[]temp=new byte[(short)dataLen];
    try {
        ok = cifraRSA.doFinal(dados, (short)0,
dataLen,temp, (short)0);
        byte[]retorno=new byte[ok];
        Util.arrayCopy(temp, (short)0, retorno,
(short)0, ok);
        return retorno;
    } catch (ArithmeticException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X01));
    } catch (ArrayStoreException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X02));
    } catch (ClassCastException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X03));
    } catch (IndexOutOfBoundsException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X04));
    } catch (NegativeArraySizeException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X05));
    } catch (NullPointerException ex) {
        ISOException.throwIt((short) (0x6F00 +
```

```

0X06));
        } catch (SecurityException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X07));
        } catch (APDUException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X10));
        } catch (CryptoException ex) {
            switch (ex.getReason()) {
            case CryptoException.ILLEGAL_USE:
                ISOException.throwIt((short) ok);
                break;
            case CryptoException.ILLEGAL_VALUE:
                ISOException.throwIt(CryptoException.ILLEGAL_VALUE);
                break;
            case CryptoException.INVALID_INIT:
                ISOException.throwIt(CryptoException.INVALID_INIT);
                break;
            case
CryptoException.NO_SUCH_ALGORITHM:
                ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM);
                break;
            case
CryptoException.UNINITIALIZED_KEY:
                ISOException.throwIt(CryptoException.UNINITIALIZED_KEY);
                break;
            default:
                ISOException.throwIt(CryptoException.NO_SUCH_ALGORITHM);
                break;
            }
        } catch (ISOException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X12));

```

```

        } catch (PINException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X13));
        } catch (SystemException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X14));
        } catch (TransactionException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X15));
        } catch (CardRuntimeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X08));
        } catch (RuntimeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X09));
        } catch (Exception ex) {
            ISOException.throwIt((short) (0x6F00 +
0X0A));
        }
        return temp;
        //apdu.setOutgoing();
        //apdu.setOutgoingLength(ok);
        //apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, ok);
    }

/**
 * Calcula o Hash de um conjunto de dados e envia à aplicação
Host
 * @param apdu APDU com os dados
 * @param buffer Buffer do comando APDU
 */
protected void calculaHash(APDU apdu, byte[] buffer) {
    short dataLen = (short) (buffer[ISO7816.OFFSET_LC]
& 0x00FF);
    calculadorDeHash.reset();
    short ok = calculadorDeHash.doFinal(buffer,
ISO7816.OFFSET_CDATA,
dataLen, buffer,
ISO7816.OFFSET_CDATA);

```

```

        apdu.setOutgoing();
        apdu.setOutgoingLength(ok);
        apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, ok);
    }

    /**
     * Assina um conjunto de bytes com a chave privada do cartão
     * @param dados Dados que serão assinados
     * @param trataAPDU true caso o parâmetro dados seja um
comando APDU ou false caso o mesmo contenha somente os bytes que
serão assinados
     * @return dados assinados
     */
    protected byte[] assinaDados(byte[] dados,boolean trataAPDU)
    {
        assinatura.init(this.guardiao.getChavePrivada(),
Signature.MODE_SIGN);
        short dataLen;
        byte[]temp=new byte[0];
        if(trataAPDU){
            dataLen = (short)
(dados[ISO7816.OFFSET_LC] & 0x00FF);
            temp=new byte[(short)(128+dataLen)];
            Util.arrayCopy(dados,
ISO7816.OFFSET_CDATA, temp, (short)0, dataLen);
        }
        else{
            dataLen = (short) dados.length;
            temp=new byte[(short)(128+dataLen)];
            Util.arrayCopy(dados, (short)0, temp, (short)0,
dataLen);
        }
        short ok = 0;
        try {
            if(trataAPDU){
                ok = assinatura.sign(dados,
ISO7816.OFFSET_CDATA, dataLen, temp,dataLen);
            }else{
                ok = assinatura.sign(dados, (short)0,
dataLen, temp,dataLen);
            }
        }
    }
}

```

```
        }
    } catch (ArithmeticException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X01));
    } catch (ArrayStoreException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X02));
    } catch (ClassCastException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X03));
    } catch (IndexOutOfBoundsException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X04));
    } catch (NegativeArraySizeException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X05));
    } catch (NullPointerException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X06));
    } catch (SecurityException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X07));
    } catch (APDUException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X10));
    } catch (CryptoException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X11));
    } catch (ISOException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X12));
    } catch (PINException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X13));
    } catch (SystemException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X14));
    } catch (TransactionException ex) {
        ISOException.throwIt((short) (0x6F00 +
0X15));
```



```

        } catch (CardRuntimeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X08));
        } catch (RuntimeException ex) {
            ISOException.throwIt((short) (0x6F00 +
0X09));
        } catch (Exception ex) {
            ISOException.throwIt((short) (0x6F00 +
0X0A));
        }
        return temp;
        //Util.arrayCopy(dados, (short)0, buffer,
ISO7816.OFFSET_CDATA, dataLen);
        //Util.arrayCopy(temp, (short)0, buffer, (short)
(dataLen+ISO7816.OFFSET_CDATA), ok);
        //short tamanhoTotal = (short) (ok+dataLen);

        //apdu.setOutgoing();
        //apdu.setOutgoingLength(tamanhoTotal);
        //apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, tamanhoTotal);
    }

/**
 * Envia o Modulo da chave pública à aplicação Host
 * @param apdu
 */
protected void getPublicKeyMod(APDU apdu){
    byte[] buffer = apdu.getBuffer();
    short length =
this.guardiao.getChavePublica().getModulus(buffer,
ISO7816.OFFSET_CDATA);
    apdu.setOutgoing();
    apdu.setOutgoingLength(length);
    apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, length);
}

/**
 * Envia o Expoente da chave pública à aplicação Host
 * @param apdu

```

```

    */
    protected void getPublicKeyExp(APDU apdu){
        byte[] buffer = apdu.getBuffer();
        short length =
this.guardiao.getChavePublica().getExponent(buffer,
ISO7816.OFFSET_CDATA);
        apdu.setOutgoing();
        apdu.setOutgoingLength(length);
        apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, length);
    }

    /**
    * Envia o Modulo da chave privada à aplicação Host
    * @param apdu
    */
    protected void getPrivateKeyMod(APDU apdu){
        byte[] buffer = apdu.getBuffer();
        short length =
this.guardiao.getChavePrivada().getModulus(buffer,
ISO7816.OFFSET_CDATA);
        apdu.setOutgoing();
        apdu.setOutgoingLength(length);
        apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, length);
    }

    /**
    * Envia o Expoente da chave privada à aplicação Host
    * @param apdu
    */
    protected void getPrivateKeyExp(APDU apdu){
        byte[] buffer = apdu.getBuffer();
        short length =
this.guardiao.getChavePrivada().getExponent(buffer,
ISO7816.OFFSET_CDATA);
        apdu.setOutgoing();
        apdu.setOutgoingLength((short)154);
        apdu.sendBytesLong(buffer,
ISO7816.OFFSET_CDATA, length);
    }

```

```
protected byte [] getCertificate(){
    return this.guardiao.getCertificate();
}

/**
 * Armazena um certificado no cartão
 * @param source
 * @param sourceOff
 * @param deslocamentoDestino
 * @param len
 */
protected void putCertificate(byte[] source,short
sourceOff,short deslocamentoDestino,short len ){
    this.guardiao.putCertificate(source, sourceOff,
deslocamentoDestino, len);
}
}
```

14 ANEXO H - ARTIGO

RIC: Um estudo para desenvolvimento de
aplicações embarcadas

Leonardo Malagoli da Silva¹

¹Departamento de Informática e de Estatística – Universidade
Federal de Santa Catarina (UFSC)

Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

leonardomalagoli@inf.ufsc.br

Abstract. This paper presents a study about embedded application development in the new Brazilian civic identification named civic identification registry (RIC). Nowadays there is a need of better interaction between humans and machines and the RIC technology, proposed by Brazilian government, aims to ease these interactions. This will make the identification process more agile and reliable, whether or not physically present. The RIC technology will be available through smartcards that will contain two embedded chips. The first one, contactless counting with OCR-B codes, makes RIC a travel document following the International Civil Aviation Organization (ICAO) patterns. The second chip has the following characteristics: cryptographic functions and memory to store digital certificates.

Resumo. Este artigo apresenta um estudo sobre o desenvolvimento de aplicações com execução embarcada no novo documento de identificação civil brasileiro, denominado registro de identidade civil – RIC. Na atual geração nota-se uma necessidade cada vez maior de interação entre humanos e máquinas, o RIC proposto pelo governo brasileiro, tem como objetivo facilitar esta interação, tornando o processo de identificação mais rápido e seguro, seja na forma presencial, ou mesmo na forma não presencial. O RIC terá o formato de um smartcard e contará com dois chips embutidos. O primeiro, sem contato, somado a características físicas e aos códigos OCR-B faz do RIC um documento de viagem no padrão definido pela *International Civil Aviation Organization* – ICAO. Já o segundo chip terá capacidade para multiplicação, além de já ser emitido com o certificado digital do titular.

1. Introdução

Na atual geração nota-se uma necessidade cada vez maior de interação entre máquinas e humanos. Esta necessidade reflete-se nos processos de identificação e ou autenticação. Neste contexto foi lançado pelo governo federal o RIC, o qual trará consigo importantes inovações no processo de identificação/autenticação na forma presencial ou mesmo não presencial. O RIC foi criado através da lei 9454 de 7 de Abril de 1997 [BRASIL 1997], porém somente em 2010, através da resolução nº 2 de 10 de Setembro, o ministério da justiça publicou as especificações básicas do novo documento [Ministério da Justiça 2010].

O RIC é um documento no formato de smartcard, o qual possui dois chips embutidos. O primeiro chip somado a características físicas e ao código OCR-B, faz do RIC um documento de viagem com capacidade biométrica padrão ICAO [Ministério da Justiça 2010], e assim sendo, é aceito em vários países do mundo. O segundo chip é emitido com um certificado digital do titular, possibilitando a identificação também na internet. Além das características técnicas do novo documento, o mesmo também tem como vantagem em relação aos documentos de identidade atuais, a sua base de dados única para todo o país [Ministério da Justiça 2012]. A base de dados unificada, somada aos recursos biométricos, faz com que seja praticamente impossível um único cidadão possuir mais de uma identidade, como ocorre com o atual RG.

Este artigo tem por objetivo o estudo do novo documento, buscando conhecer as tecnologias empregadas, com intuito de implementar e descrever o processo de desenvolvimento de aplicações para execução embarcada ou com suporte ao RIC.

Com a emissão do RIC, surgirá uma demanda crescente por sistemas capazes de lidar com esta tecnologia. Além disso, o novo documento representa um avanço tecnológico no processo de identificação/autenticação civil, apresentando características ainda pouco exploradas no mundo. É importante citar também, que o projeto do RIC representa um grande investimento do governo federal justificando assim este esforço em conhecer o novo documento.

2. Características do RIC

O RIC terá o formato de um smartcard, e contará com dois chips, sendo um com e outro sem contato. O chip sem contato do RIC será utilizado para questões de *match-on-card*, nele serão armazenadas imagens para identificação/autenticação de forma biométrica [Ministério da Justiça 2010]. Com o RIC, será possível identificar biometricamente o titular através da biometria da digital, e através da biometria da face. Ambas as formas de identificação biométricas suportadas pelo RIC, estão entre as mais aceitas no mundo [Magalhães e Santos 2003] e são consideradas pela ICAO como formas de identificação biométrica de funcionamento mundial [ICAO 2008] .

O chip com contato será utilizado para identificação/autenticação na forma digital, visto que o mesmo terá o certificado digital do titular, além de possuir capacidade para realizar operações criptográficas tais como cifrar, decifrar, assinar e verificar assinatura. O chip com contato do RIC, também possuirá capacidade para multi-aplicação [Ministério da Justiça 2010].

3. Desenvolvimento de aplicações

O desenvolvimento de aplicações para execução embarcada no RIC, enfrenta grandes dificuldades, com destaque para:

- Cartões diferentes por estado;

- Capacidade de armazenamento;
- Homologação de aplicações

3.1 Diferentes cartões

Cada estado é livre para realizar concorrências públicas para escolha do seu fornecedor [Ministério da Justiça 2010]. Com isso, em diferentes estados diferentes fornecedores poderão ser escolhidos, e conseqüentemente diferentes sistemas operacionais e de entrada e saída poderão ser adotados. Uma forma de contornar esta característica, seria através do javacard, o qual cria uma abstração do hardware e do sistema operacional do cartão, fazendo com que uma aplicação desenvolvida nesta tecnologia possa ser executada em qualquer cartão que à suporte [SUN VMS 2000]. No entanto, tal tecnologia não foi incluída nas características técnicas básicas definidas pelo Ministério da Justiça.

3.2. Armazenamento no RIC

A capacidade de armazenamento é uma das grandes dificuldades encontradas por quem for desenvolver aplicações para execução embarcada no RIC. Apesar de o mesmo possuir dois chips, o chip sem contato do RIC é bloqueado para escrita de dados, o que torna impossível a instalação de novas aplicações [ICAO 2008]. O Bloqueio do chip sem contato do RIC, é um determinação da ICAO, e deve ser feito após a fase de personalização.

O chip de contato do RIC não está bloqueado, porém no mesmo encontram-se armazenados os dados relacionados com certificação digital (certificado do titular, cadeia de certificados para verificação do certificado e etc.), e 4 imagens de digitais [Ministério da Justiça 2010]. Estas imagens seguem o padrão definido pela ICAO, e deverão ter um tamanho estimado de 10KB cada uma [ICAO 2008]. Os dados relacionados a certificação digital, devem ocupar aproximadamente 16KB

[ICP-Brasil 2007], ou seja, restam apenas 8 KB para uso por outras aplicações.

A capacidade de armazenamento disponível no RIC, limita bastante as aplicações que poderiam ser embarcadas, aplicações que necessitam armazenar dados no cartão, serão bloqueadas por esta limitação.

3.3. Processo de Homologação

Os chips do RIC, deverão atender a requisitos definidos por duas instituições distintas. O chip de contato do RIC, deverá atender às definições da ICP-BRASIL. Já o chip sem contato deverá atender aos requisitos definidos pela ICAO[Ministério da Justiça 2010].

A ICAO através do doc.9303, define características físicas e lógicas como : dados armazenados no cartão, capacidade, interfaces e processo de autenticação. A ICAO também define características relacionadas à produção dos documentos, (processo de controle de produção, segurança nos locais de produção, no transporte entre outros). Entre os requisitos de segurança definidos para um documento padrão ICAO, está o bloqueio do chip sem contato após a fase de personalização do documento, o qual impede que novas aplicações sejam instaladas em tal chip [ICAO 2008].

A ICP-BRASIL, responsável pela homologação do chip de contato do RIC, permite que novos serviços sejam adicionais a um cartão criptográfico, porém tais serviços devem estar instalados durante a homologação do cartão perante a ICP-BRASIL [ICP-Brasil 2007]. Isto limita o desenvolvimento de novas aplicações ao RIC somente aos fabricantes de cartões.

4. Experimentos

O desenvolvimento de aplicações para execução embarcada no

RIC é um desafio, visto o pequeno espaço disponível para o armazenamento e a falta de informações sobre o sistema operacional do cartão. Por isso, para demonstrar o desenvolvimento de uma aplicação para execução embarcada, foi necessário assumir que todos os cartões do RIC atenderam à algumas premissas. São elas: Todos os cartões do RIC possuíam suporte a javacard em versão igual ou superior a versão 2.2.1 [SUN JCRE 2000]; Todos os cartões estarão de acordo com o padrão PKCS#15 [RSA 2000]. Estas premissas foram assumidas após análise dos cartões existentes no Laboratório de Segurança em Computação – LabSEC, onde praticamente todos possuíam tais características.

Para demonstrar o processo de desenvolvimento de aplicações para execução embarcada no RIC, foi desenvolvido um *applet* gerenciador de ponto eletrônico, este applet interage com uma aplicação desktop para registrar e armazenar os registros de ponto do trabalhador no RIC.

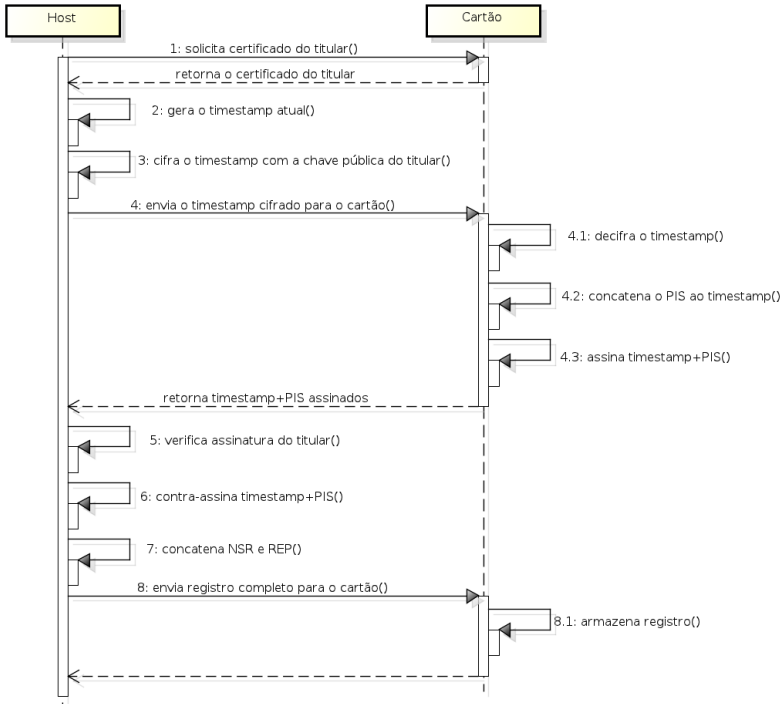


Ilustração 1: Protocolo de comunicação entre aplicação host e o applet

O sistema desenvolvido utiliza técnicas criptográficas para autenticar o applet e gerar um registro autêntico que não possa ter sua autoria negada por nenhuma das partes. Para isso, o certificado digital do titular armazenado no cartão é utilizado para autenticar o titular e assinar o registro. A aplicação desktop também possui um certificado digital e também assina o registro. Desta forma o registro gerado é assinado por ambas as partes. O protocolo utilizado é apresentado na figura 1.

Este experimento tem como objetivo exemplificar o desenvolvimento de uma aplicação para execução embarcada no RIC.

4.1 Etapas do desenvolvimento de um applet JavaCard

O desenvolvimento de um *applet javacard*, assim como o desenvolvimento de qualquer aplicação *Java*, começa pela codificação de um ou mais arquivos de classes. Após a fase de codificação, os arquivos gerados são compilados com um compilador *Java*, por exemplo o *javac*, e convertidos em arquivos *.cap* e *.exp*, sendo o arquivo *.cap* o binário o qual será interpretado pela máquina virtual *javacard* [SUN VMS 2000].

Todo *applet javacard*, possui uma classe que estende a classe *Applet* do pacote *javacard.framework*. Esta classe será a porta de comunicação entre o *JCRE* e o nosso *applet*. Uma classe que estende a classe *Applet* deve implementar os métodos *install*, *process*, *select* e *deselect* [SUN JCRE 2000]. Para o desenvolvimento do *applet* Ponto Eletrônico, a classe *Core* foi definida como uma extensão da classe *Applet*.

Além dos métodos obrigatórios, também foram implementados métodos para tratar cada um dos possíveis *bytes* de instruções. Após a seleção do *applet*, todos os APDUs seguintes são direcionados ao *applet*, onde são recebidos no método *process*, método este responsável por receber o APDU e dar o encaminhamento correto.

14.1.1 4.2. Manipulação de dados em *javacard*(cópia de *arrays*, tipos, operações, API e etc...).

Javacard possui uma grande API, a qual fornece vários métodos e classes que facilitam o desenvolvimento de aplicações embargadas. As principais são:

- *APDU* – classe que cria uma abstração dos pacotes físicos transmitidos entre o cartão e o leitor, seguindo o formato APDU definido na norma ISO7816-4. Esta classe possui vários métodos, entre eles, métodos que permitem identificar o protocolo utilizado na comunicação, o tamanho configurado dos bloco de saída e entrada, ler e enviar *bytes*.

- *Util* – A classe *Util* contém funções utilitária comuns, alguns métodos são implementados como funções nativas por questões de desempenho. Todos os seus métodos são estáticos. A classe *Util* possui métodos para cópia e comparação de *arrays*, concatenação e separação de *bytes* entre outros.
- ISO7816 – A classe ISO7816 define constantes relacionadas a ISO7816-3 e ISO7816-4. Todas as suas constantes são estáticas.
- *Applet* – A classe *Applet* é talvez é classe mais importante da API, visto que todo o *applet* para executar no cartão deve ter ao menos uma classe que estenda a classe abstrata *Applet*.
- *JCSystem* – A classe *JCSystem* possui vários métodos para controle de execução do *applet*, gerenciamento de recursos, compartilhamento de objetos e operações atômicas, além de controlar a persistência a transitoriedade dos objetos. Todos seus métodos são estáticos.

14.1.2 4.3. Armazenar dados no cartão

O armazenamento de dados no cartão, ou seja, dos registros de batidas de ponto, será realizada através de um objeto persistente. Objetos persistentes são armazenadas na área de EEPROM, por isso a performance na leitura e escrita de informações neste objeto é muito inferior. A escrita em um objeto persistente é cerca de 1000 vezes mais lenta que a escrita em um objeto não persistente.

No *applet* gerenciador de certificados digitais, foi definida uma classe exclusiva para tratar do armazenamento

dos dados, esta classe foi denominada *Persistencia* e possui um *array* de *bytes* onde são armazenados os registros. Nesta classe foram disponibilizados métodos para manipulação da memória, são eles: *registraPonto*, *limpaDados* e *exportarDados*, conforme segue:

- *registraPonto*, método que recebe como parâmetro um *array* de *bytes* contendo um registro de ponto, já com a assinatura e adiciona este registro a memória. Adicionar o registro a memória significa inseri-lo no *array* de *bytes* de memória, que nada mais é que, fazer com que os dados fiquem armazenados na EEPROM, visto que o objeto *Persistencia* é um objeto persistente como citado anteriormente.

- *exportaDados*, método que recebe dois parâmetros do tipo *byte*, e retorna todos os registros desde a posição representada pelo primeiro parâmetro até o total de registros representados pelo segundo parâmetro.

14.1.3 4.4. Comunicação entre *applet javacard* e aplicação *host*;

A comunicação entre o *applet* e a aplicação *host* é feita através de pequenos pacotes de dados denominados de APDU, o tratamento dado para enviar e receber estes pacotes do lado *host* e do lado do cartão são diferentes, visto que do lado *host* pode-se utilizar qualquer linguagem desktop ou web, enquanto do lado do cartão utiliza-se uma linguagem de programação embarcada.

Neste experimento, optou-se por utilizar a tecnologia *Java* dos dois lados da comunicação. No lado *host* ou *desktop* ou ainda aplicação externa ao cartão, foi utilizado o *Java Standard Edition* - JSE. Já no lado *Applet* ou cartão, foi utilizado a tecnologia *javacard*.

14.1.3.1 Lado *Host* ou *Desktop* - Do lado *host*, foram

utilizadas as classes TerminalFactory, CardTerminal, Card, CardChannel, CommandAPDU e ResponseAPDU. Todas do pacote javax.smartcardio.

14.1.3.2

14.1.3.3 Lado Applet - Do lado *Applet*, foi utilizada a classe APDU e a interface ISO7816, ambas da API javacard, mais especificamente do pacote javacard.framework.

14.1.3.4

O recebimento de um APDU pelo *applet* é bastante simplificado em *javacard*, o JCRE recebe o pacote e o repassa ao *applet* através de uma chamada ao método *process* do mesmo, passando como parâmetro um objeto APDU com os *bytes* recebidos [SUN JCRE 2000]. Após o recebimento do APDU, os dados recebidos são tratados com auxílio da interface ISO7816.

4.5. Tratamento de Exceções

O suporte ao tratamento de exceções em *javacard* é muito similar ao tratamento dado na linguagem *Java* (JSE), assim como em JSE, existe a classe *Throwable* que estende diretamente a classe *Object* e serve como uma raiz na hierarquia de classes de exceções. O desenvolvedor também pode criar as suas próprias classes de exceções, para isso, a classe criada pelo desenvolvedor deve estender a classe *CardException* caso seja uma exceção verificada, ou seja, que é verificada durante a compilação, ou ainda estender a classe *RuntimeException*, caso seja uma exceção de tempo de execução

4.6. Controle de acesso e privilégios através de PIN e PUK

O controle de acesso através dos códigos PIN e PUK são realizados em *javacard* através da classe *OwnerPIN*, a qual implementa a interface PIN. A classe *OwnerPIN* implementa todas as funções básicas relacionadas a um PIN, como

inicializar/alterar o seu valor, realizar o controle de tentativas de validação, controle sobre o bloqueio do PIN ao ultrapassar o limite de tentativas, controla também a validade do PIN apresentado para uma determinada sessão CAD, mantendo-se ativo durante a mesma, e oferecendo métodos para consultar esta validade.

A implementação da classe *OwnerPIN* protege o PIN contra ataques baseados na previsão de fluxo do programa, se a transação está em andamento, o estado interno, não deve ser condicionalmente atualizado durante a apresentação do PIN. A classe *OwnerPIN* garante que todas as matrizes criadas durante os processos de inicialização/atualização e validação são transitórios do tipo *CLEAR_ON_RESET*, ou seja, serão excluídos ou “limpos” durante o *reset* do cartão.

Para a implementação de um PIN global do cartão, são combinadas as classes *OwnerPIN* e a interface *Shareable*. A instância *OwnerPIN* é manipulada somente pelo contexto proprietário, porém para os demais contextos, é disponibilizada uma interface pública através de uma interface *Shareable*.

No *applet* Gerenciador de Ponto Eletrônico, a implementação do controle através do PIN, foi realizado na classe *Core*, esta classe, é a classe principal do *applet*, sendo esta que estende a classe *Applet* da API *javacard*. Para dar a classe *Core* a capacidade de controle de acesso através de código PIN, foram feitas as seguintes mudanças na classe:

- criado um atributo do tipo *OwnerPIN*, o qual possui controlador de acesso do tipo *protected*, ou seja, pode ser acessado por qualquer classe dentro do mesmo pacote, além de classes de outros pacotes que estendam a classe *Core*;
- métodos de iniciação, atualização e validação do PIN, sendo estes acessados através dos códigos de instruções 0C, 0B e 0A respectivamente. Para atualizar o PIN, é necessário

primeiro apresenta o PIN atual válido, através do código de instrução 0A;

- os métodos que tratam da exportação de dados, seja de forma cifrada ou não ou ainda os métodos que tratam da atualização de dados pessoais, receberam as linhas apresentadas na tabela 27, as quais verificam se o PIN já foi validado na sessão CAD atual.

```
if(!pin.isValidated()){  
  
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED  
);  
}
```

Tabela 27: Verifica se PIN foi validado

14.1.4 4.7. Criptografia em *javacard*;

Em *javacard*, são disponibilizadas uma variedade de classes e interfaces para implementação de operações relacionadas à segurança através dos pacotes *javacard.security* e *javacard.cripto*.

No pacote *javacard.security* são disponibilizadas interfaces que criam abstração das chaves utilizadas em diferentes algoritmos de criptografia, *DESKey*, *DSAKey*, *DSAPrivateKey*, *DSAPublicKey*, *Key*, *PrivateKey*, *PublicKey*, *RSAPrivateCrtKey*, *RSAPrivateKey*, *RSAPublicKey* e *SecretKey*. São disponibilizadas também, classes para geração e armazenamento de chaves ou pares de chaves(*KeyBuilder* e *KeyPair*), classe base para algoritmos de *hash* (*MessageDigest*), classe para geração de número aleatórios (*RandomData*) e classe base para algoritmos de assinatura(*Signature*). Além da exceção *CryptoException*.

No pacote `javacard.cripto` é implementada a interface `KeyEncryption` e a classe `Cipher`, a qual é uma classe abstrata base para algoritmos de criptografia.

No Experimento, foram implementadas duas operações criptográficas simétricas. São elas: criptografar os registros armazenados no cartão e exportar à aplicação *host* e decifrar registros cifrados recebidos da aplicação *host*. A implementação da chave simétrica, foi feita utilizando a interface `DESKey`, ou seja, foi criado um atributo `DESKey`, para armazenar a chave do usuário. Para gerar a chave, foi utilizado o método estático `buildKey` da classe `KeyBuilder`.

Para cifrar e decifrar os dados, foi utilizado o método `doFinal` da classe `Cipher`. A diferença entre os métodos de cifrar e decifrar, está na inicialização do algoritmo, a qual é realizada através do método `init` da classe `Cipher`.

Também foram definidas métodos para cifrar e decifrar um conjunto de *bytes* recebidos com criptografia assimétrica.

Para a implementação das chaves públicas e privadas, foram utilizadas as interfaces `RSAPublicKey` e `RSAPrivateKey`, além das classes `KeyPair` e `Cipher`. Estas interfaces foram utilizadas para armazenar respectivamente as chaves pública e privada. Já as classes `KeyPair` e `Cipher` foram utilizadas respectivamente para gerar as chaves e para realizar as operações de cifragem e decifragem dos dados. Para gerar as chaves, foi utilizado o método `gemKeyPair()`, o qual gera um par de chaves assimétrica, o algoritmo utilizado e o tamanho da chave são definidos na inicialização do objeto `KeyPair` no qual foi chamado o método `gemKeyPair()`. Após gerar o par de chaves, foram utilizados os métodos `getPublic` e `getPrivate` para buscar respectivamente as chaves pública e privada do par

de chaves gerados.

Para cifrar e decifrar os dados, assim como nas operações criptográficas simétricas, foi utilizado o método *doFinal* da classe *Cipher*. O que difere os métodos *doFinal* utilizados para cifrar e decifrar, é a sua inicialização. A definição do algoritmo utilizado, é setado na inicialização do objeto *Cipher* utilizado para cifrar e decifrar

Além das operações para cifrar e decifrar dados com criptografias simétrica e assimétrica, também foram implementados métodos para calculo de *hash* e assinatura de dados. Para calculo do *hash*, foi utilizado o método *doFinal()* da classe *MessageDigest*, antes porém é necessário inicializar o objeto *MessageDigest*. Uma vez inicializado, para calcular o *hash* de um *array* de *bytes*, basta chamar os métodos *reset()*, seguido pelo método *doFinal*.

Para o método responsável por assinar um *array* de *bytes*, o qual normalmente deverá ser um *hash*, foi utilizado um objeto da classe *Signature*. O procedimento realizado para assinar os dados, é muito semelhante ao procedimento executado nos métodos de cifragem e decifragem. Primeiro inicializa-se o objeto da classe *Signature*, depois assina-se um conjunto de *bytes* através da chamada a um método da classe *Signature*, neste caso foi utilizado o método *sign*, que recebe os mesmos parâmetros do método *doFinal* da classe *Cipher*.

5. Conclusão e Trabalhos futuros

Este artigo tinha o objetivo principal de estudar o processo de desenvolvimento de aplicações para execução embarcada no RIC, porém como foi apresentado ao longo do trabalho, existem barreiras ao desenvolvimento de tais aplicações. As principais barreiras são: a falta de um processo de homologação para aplicações de execução embarcada e a

capacidade de armazenamento do novo documento. O processo de homologação é importante uma vez que a aplicação em questão irá compartilhar o chip de contato do RIC com a aplicação responsável por gerir os certificados e chaves do titular.

Como forma de continuação deste trabalho, sugere-se um estudo sobre protocolos de segurança para identificação/autenticação do titular, buscando o melhor uso dos recursos disponibilizados no RIC. Para que este trabalho seja realizado, será necessário primeiro um estudo mais detalhado sobre como acessar, extrair e decodificar as informações armazenadas no RIC, utilizando cartões reais do RIC.

É importante também, a validação das informações apresentadas neste trabalho sobre o desenvolvimento de aplicações para execução embarcadas no novo documento, para isto é necessário um cartão real do RIC, onde deverão ser embarcadas aplicações desenvolvidas conforme apresentado neste trabalho, buscando validar ou refutar tais informações. Este estudo é importante, uma vez que o novo documento ainda está em desenvolvimento e não foi possível realizar tais validações com as informações existentes atualmente.

Por fim, sugere-se um estudo para para o desenvolvimento de um *applet* gerenciador de certificados de atributos para execução embarcada no novo documento, apresentado as mudanças necessárias, assim como também demonstrando que este *applet* não afeta a segurança no RIC.

6. Referências:

Resolução mj nº2 de 10 de setembro de 2010, disponível em: <<http://ws.mp.mg.gov.br/biblio/informa/011013936.htm>>

BRASIL – Lei nº 9454, de 7 de abril de 1997, disponível em: <http://www.planalto.gov.br/ccivil_03/Leis/L9454.htm>

RIC – Site do ministério da justiça, Maio de 2012 disponível em : <<http://portal.mj.gov.br/ric>>

MAGALHÃES, Paulo Sérgio; SANTOS, Henrique Dinis dos. Biometria e autenticação. 2003.

International Civil Aviation Organization . ICAO doc. 9303:documentos de viagem de leitura mecânica, 2008.

CARD, Java. 2.1. 1 Virtual Machine Specification. **Sun Microsystems**, 2000.

Manual de Condutas Técnicas 1 – Volume1 : Requisitos, Materiais e Documentos Técnicos para Homologação de Cartões Criptográficos (Smart Cards) no Âmbito da ICP-Brasil Versão 3.0. **Infra-Estrutura de chaves Públicas Brasileira**, 2007.

LABORATORIOS, R. S. A. PKCS# 15: Cryptographic Token Information Format, 2000.

CARD, Java. 2.1. 1 Runtime Environment (JCRE) Specification. **Sun Microsystems**, 2000.