Universidade Federal de Santa Catarina

Ciências da Computação

Luca Fachini Campelli

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA COLETA DE DADOS E EMISSÃO DE PASSAPORTES ELETRÔNICOS NA PLATAFORMA JAVACARD

Florianópolis/SC
2018

Luca Fachini Campelli

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA COLETA DE DADOS E EMISSÃO DE PASSAPORTES ELETRÔNICOS NA PLATAFORMA JAVACARD

Trabalho de Conclusão de Curso
para a graduação no curso de
Ciências da Computação
UFSC

Florianópolis, 2018

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LUCA FACHINI CAMPELLI

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Ciências de Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Prof. Dr. Jean Everson Martina
Universidade Federal de Santa Catarina - UFSC

Banca: Mestre Lucas Pandolfo Perin
Universidade Federal de Santa Catarina - UFSC

Banca: Mestre Felipe Coral Sasso
Universidade Federal de Santa Catarina - UFSC

Florianópolis, 5 de dezembro de 2018

Dedico este trabalho a todas as pessoas que me iluminaram o caminho perante a escuridão.

Resumo

Foi desenvolvido um software para computador capaz de coletar as informações do usuário, e emitir um passaporte eletrônico em um Javacard que segue o padrão ICAO 9303. Este cartão é munido de todas as informações necessárias para a verificação da identidade do usuário, incluindo informações biométricas como identificação facial, e digitais, juntamente com todos os mecanismos de segurança descritos pelo documento ICAO 9303.

Palavras-chave: Javacard, Passaporte,JMRTD, Segurança, Passaporte-Eletrônico, e-Passport, SmartCard, JCOP

Abstract

A computer software was developed capable of collecting user information, and emiting an electronic passport on a Javacard that follows the ICAO 9303 standard. This card posesses all required information to verify the user's identity, including biometrics such as face recognition and fingerprints, together with all security mecanisms described in the ICAO 9303 document.

Key-Words: Javacard, Passaport,JMRTD, Security, Electronic-Passport, e-Passport, SmartCard, JCOP

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AA Active Authentication

APDU Aplication Protocol Data Unit

BAC Basic Access Control

BER Basic Encoding Rules

BSI Bundesamt für Sicherheit in der Informationstechnik (Escritório Federal de Segurança em Tecnologia da Informação)

CA Chip Authentication

DG Data Group

EAC Extended Access Control

GUI Graphical User Interface

ICAO International Civil Aviation Organization

ITI Instituto de Teccnologia da Informação

JMRTD Java Machine Readable Travel Documents

JNI Java Native Interface

MRE Ministério das Relações Exteriores

MRZ Machine Readable Zone

OCR Optical Character Recognition

PA Passive Authentication

PACE Password Authenticated Connection Establishment

SOD Security Object Data

TA Terminal Authentication

TLV Type-Length-Value

UCL Université Catholique de Louvain

Lista de excertos de código

Sumário

1 Introdução

Quando começaram a ser utilizados, os passaportes funcionavam apenas por meio físico, na forma de uma caderneta ou documento escrito, com todas as informações sendo verificadas manualmente e visualmente,
comparando-as com o outros papéis ou documentos autenticados, abrindo brechas para falsificações como cópias de assinaturas ou selos de autenticidade.

Conforme o documento foi evoluindo, até se tornar a caderneta que conhecemos hoje, várias formas de prevenir falsificações como marcas
d'água, padrões de impressão e manufatura do papel foram utilizadas, porém a inovação mais impactante e discutida foi a inserção de um chip eletrônico dentro do passaporte (R.K. Roberto: 2015).

Embora seja possível verificar as informações sobre o nome, idade, número do documento, e a foto do portador visualmente, mais informações ainda podem ser verificadas por meio deste chip interno do passaporte, e mais rapidamente, permitindo até que a aferição do passaporte seja automatizada. Este chip pode guardar as informações biométricas do dono, junto com as informações visuais, e ainda mais importante, possui diversos protocolos de segurança para garantir a autenticidade dos dados conferidos (International Civil Aviation Organization: 2015).

1.1 Motivação

A emissão de passaportes brasileiros é efetuada somente pela Casa da Moeda do Brasil, e, para uma pessoa comum fazer o pedido e receber o seu passaporte podem se passar de sete (7) dias, a alguns meses, dependendo dos horários de agendamento no posto de emissão (Carlos Henrique Ramos Lima: 2015). Os cartões Javacard são produzidos por diversas empresas, o que estimularia a competitividade. Uma dessas empresas é a NXP Semiconductors, uma das empresas líderes no mercado de segurança em sistemas embarcados e internet das coisas. Além disso, ainda existem máquinas capazes de imprimir nestes cartões instantâneamente, e com o auxílio da aplicação sendo desenvolvida como parte deste trabalho, a emissão de um passaporte pode ser feita logo depois da entrevista agendada, com a pessoa saindo do local já com o cartão em mãos.

Diversos softwares como o ReadID (InnoValor: 2013) existem para a leitura de passaportes e, daqueles escritos em Java, em sua maioria se utilizam da biblioteca JMRTDcomo base para seu funcionamento. Sendo assim, seria possível emitir um passaporte na plataforma Javacard, utilizando bibliotecas open-source, que seja equivalente ao passaporte oficial utilizado hoje em dia?

1.2 Objetivos

O objetivo principal é a criação do sistema de coleta de dados e emissão de um passaporte eletrônico em um cartão Javacard, onde os dados sejam armazenados depois possam ser extraídos e validados em qualquer tipo de máquina que possua este sistema.

Desta forma, alguns objetivos específicos são propostos para que o objetivo se dê por alcançado:

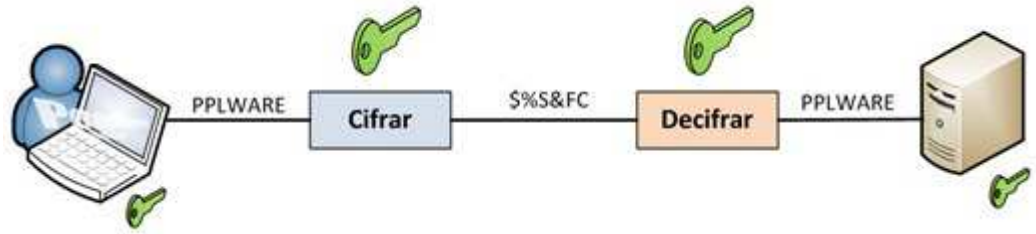
1. Desenvolver o sistema para coletar os dados básicos do usuário.
2. Desenvolver o sistema para tirar a foto do usuário e extrair dela os dados para o reconhecimento facial.

3. Desenvolver o sistema para coletar os dados biométricos do usuário como digital e íris.

4. Utilizar as informações coletadas para preencher o cartão, cumprindo as especificações do padrão ICAO9303.

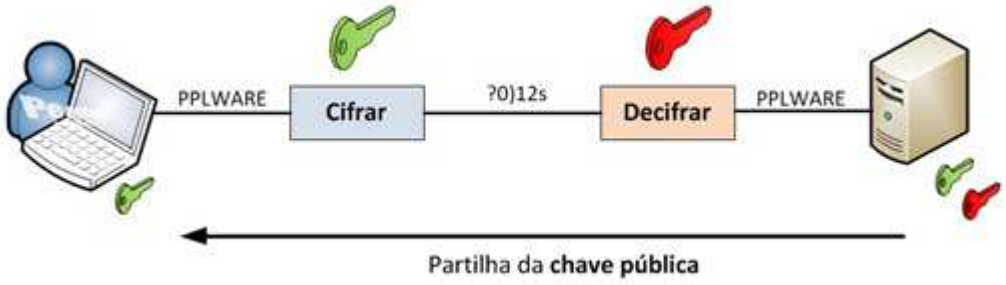
5. Obter um cartão equivalente a um passaporte oficial, na questão de sua estrutura interna e dados armazenados

Figura 1: Esquema de criptografia simétrica



Fonte: https://pplware.sapo.pt/tutoriais/networking/criptografia-simetrica-e-assimetrica-sabe-a-diferenca/

Figura 2: Esquema de criptografia assimétrica



Fonte: https://pplware.sapo.pt/tutoriais/networking/criptografia-simetrica-e-assimetrica-sabe-a-diferenca/

2 Fundamentações teóricas e práticas

2.1 Criptografia

É possível cifrar a informação para que apenas quem possua a chave utilizada para cifrar possa acessa-la. O ato de embaralhar o significado da mensagem se chama cifrar, e o inverso, decifrar. Para tal, utiliza-se algum tipo de algoritmo que, em conjunto com uma chave, seja capaz de mascarar a informação original e que seja reversível, para que com o uso da chave, se possa desfazer a criptografia e acessar o conteúdo original da mensagem. Esta chave pode ser acordada préviamente ou no momento que a troca de mensagens se iniciar.

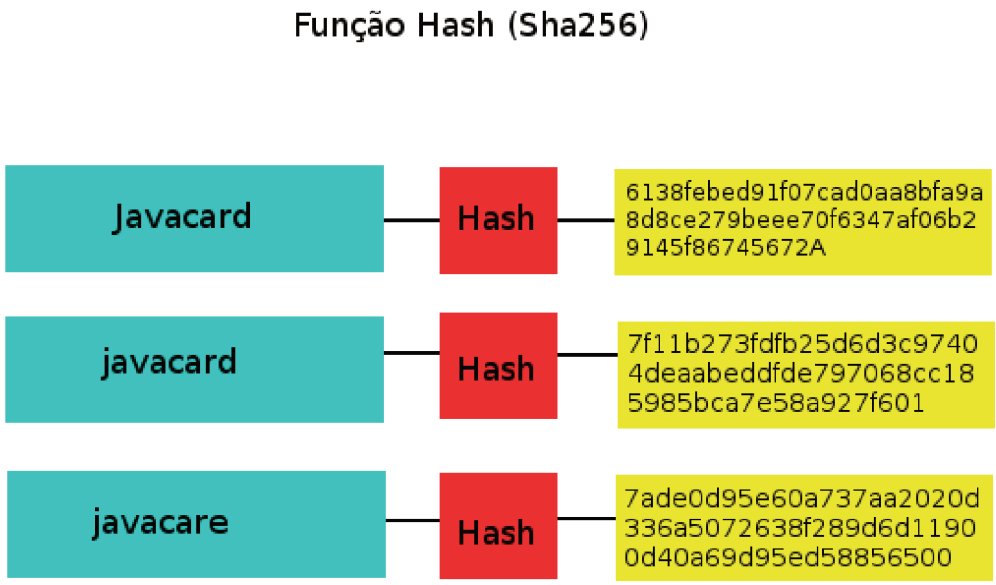
Nesta área existem dois tipos de protocolos para cifrar mensagens: a criptografia Simétrica e a Assimétrica. Na criptografia simétrica, uma única chave é utilizada por ambas as partes, para cifrar e decifrar a mensagem. DES e AES são exemplos de algoritmos que implementam a criptografia simétrica.

Na criptografia assimétrica, são utilizadas duas chaves distintas, uma pública e uma privada. Ambas podem ser usadas para cifrar, porém o que uma cifrar, somente poderá ser decifrado pela outra. Devido às dificuldades de se utilizar este protocolo ele é mais utilizado para certificações digitais. RSA e DSA são exemplos de algoritmos que utilizam a criptografia assimétrica.

2.2 Função Hash

Esta função é amplamente utilizada para garantia de autenticidade de dados tanto em criptografia quanto em várias outras áreas. Ela funciona embaralhando os bits de uma mensagem à um ponto que seja impossível inverter o processo. Ela aceita uma mensagem de qualquer tamanho, porém sempre retornará uma cadeia de bytes de tamanho fixo, não importando qual seja a entrada, sendo este tamanho diferente para cada tipo de função hash. Para que seja considerada segura, a função Hash deve possuir uma resistência a colisão forte, ou seja, dadas duas mensagens X e Y distintas entre si, $H(X)! = H(Y)$, para todas as mensagens X e Y distintas.

Figura 3: Função Hash



Fonte: Própria

Desta forma, é possível se confirmar a integridade de uma mensagem, aplicando a função hash sobre ela antes de ser enviada, e depois de recebida, comparando a hash da mensagem com a hash recebida, pois se a mensagem foi alterada, por menor que seja a alteração, os dois resultados da função serão completamente diferentes. Um dos algoritmos mais utilizados para esta função é o SHA 2 - Secure Hash Algorithm 2(Algoritmo de Hash Seguro 2), projetado pela NSA, e possui seis versões diferentes, onde a mudança principal está no número de bits que eles retornam: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 (Brilliant.org: 2018).

2.3 Assinaturas Digitais

A confiança entre duas partes de uma troca de mensagens pode não ser suficiente para que informação sensível seja transferida. Uma parte "A" pode forjar uma mensagem alegando ter sido enviada pela parte "B", ou "B" pode simplesmente negar ter enviado qualquer mensagem, mesmo que o tenha. Desta forma, um meio de proteger âmbas as partes contra fraude mútua são as assinaturas digitais (*How Digital Signatures Work*).

Assinar uma informação digitalmente significa garantir que esta informação foi inegavelmente escrita pelo remetente, e que não foi modificada durante o envio ao destinatário. Para isto é utilizada a função Hash, e criptografia assimétrica.

Por exemplo, para Alice enviar uma mensagem M assinada digitalmente para Bob, ela deve primeiro passar a mensagem por uma função Hash H, obtendo-se $H(M)$. Depois, ela deve possuir um par de chaves assimétricas, e cifrar $H(M)$ com sua chave privada sk obtendo $C_{sk}(H(M)) = S$ e juntar o resultado ao final da mensagem formando MS.

Ao receber a mensagem assinada MS, se Bob quiser verificar a assinatura de Alice, ele deve separar S de M, e decifrar S com a chave pública, pk de Alice, utilizando o mesmo algoritmo de criptografia, assim obtendo $C_{pk}(C_{sk}(H(M')) = H(M')$. Para terminar, Bob precisa passar a mensagem M pela mesma função Hash que Alice utilizou e então comparar os resultados da Hash $H(M)$ feita da mensagem e da Hash $H(M')$ obtida da assinatura. Se $H(M) = H(M')$ então pode-se concluir que:

- A mensagem não foi alterada, pois qualquer alteração na mensagem teria alterado o resultado de $H(M)$ e assim $H(M)! = H(M')$.
- A mensagem foi inegavelmente escrita por Alice, já que somente a sua chave privada poderia ter assinado a mensagem, para ser decodificada pela sua chave pública.

2.4 Certificados Digitais

Certificados digitais são documentos eletrônicos que garantem a identidade de um certo elemento. Este que pode ser uma entidade, um site da internet ou um terminal de cartões. Um certificado é composto de algumas informações da entidade, a chave pública da entidade, e uma assinatura digital sobre o certificado. Dependendo do tipo de certificado, esta assinatura adicionada ao final do documento é cifrada ou com a chave privada de quem o criou, denominado de certificado auto-assinado, ou com a chave privada de uma autoridade certificadora (CA - Certificate Authority), que é uma entidade em que o dono do certificado, e a entidade remetente que o verifica possuem confiança.

No Brasil, o Instituto de Tecnologia da Informação, ITI, criou a ICP-Brasil, ou Infraestrutura de Chaves Públicas. Esta é uma estrutura hierárquica de certificação com raiz única, onde o ITI possui o papel da Autoridade Certificadora Raiz, podendo credenciar e descredenciar os demais participantes da cadeia. O ITI também possui um registro de todas as entidades que são Autoridades Certificadoras, sob a AC-Raiz (*Autoridades Certificadoras*).

2.5 O Documento ICAO 9303

O documento 9303 da *Iternational Civil Aviation Organization* - ICAO, é o documento que regulamenta passaportes eletrônicos, e é o padrão utilizado para toda formatação de passaportes. ICAO é uma agência especializada das Nações Unidas. Foi estabelecida em 1944 para gerenciar a administração e governância da Convenção de Aviação Civil Internacional (Convenção de Chicago)(International Civil Aviation Organization: 2015).

O documento é escrito em 12 partes, cada uma descreve um aspecto sobre como um passaporte eletrônico deve ser desenvolvido, a primeira parte dando uma introdução sobre o as características de um passaporte eletrônico, e definindo siglas e acrônimos para os documentos seguintes.

A segunda parte detalha especificações sobre a segurança física do cartão, desde o design interno, quanto a produção, transporte e criação do cartão.

A terceira parte especifica as características sobre a apresentação de todos os tipos de passaporte, como fonte, linguagem, campos de preenchimento e representação das informações.

A quarta parte específica a formatação dos dados para passaportes legíveis por máquina, que possuem forma de livreto como são hoje, mostrando dimensões e formatos dos campos, também chamado de tipo TD3.

A quinta parte especifica como deve ser o layout dos dados para passaportes do tipo cartão plástico impresso, chamado TD1. Este cartão possui uma zona legível por máquina (MRZ), que possui os dados básicos de seu portador, formatados para leitura tanto humana quanto por máquina, impresso em seu exterior junto com todas as outras informações no verso, e uma imagem do portador.

A sexta parte se assemelha à quinta, pois descreve as características de cartões do tipo TD2, que possuem de diferente do tipo TD1 apenas suas dimensões, tendo todas as informações necessárias em um lado do cartão, permitindo informações opcionais serem adicionadas ao verso.

A sétima parte se refere à vistos legíveis por máquina. Estes se assemelham aos tipos de passaportes TD1 e TD2, porém possuem as informações referentes aos vistos.

A oitava parte não foi preenchida ainda, e é mantida reservada para uso futuro.

A nona parte fala sobre como se deve dispor das informações biométricas do portador do passaporte, formatos e dados, tomando como principal a identificação facial da pessoa, e como identificações secundárias as digitais e íris.

A décima parte especifica a estrutura logica de dados (LDS) do cartão, e como se deve armazenar os dados biométricos dentro do cartão.

A décima primeira parte especifica como funcionam os protocolos de segurança necessários à comunicação do cartão e obtenção dos dados biométricos para verificação da identidade da pessoa.

Por último a décima segunda parte especifica o funcionamento de todas as assinaturas digitais necessárias à certificação do cartão via chaves públicas, e sua infra-estrutura.

2.6 Javacard

O Javacard é um cartão eletrônico que possui dois componentes principais, um chip de contato e um processador interno. Este chip de contato se faz presente do lado externo do cartão, da mesma forma que cartões de crédito ou débito modernos, e cuida da comunicação do cartão com a leitora. O processador interno funciona como um computador, ele roda um sistema operacional Java, o que torna este cartão um Javacard.

Ele também possui uma memória interna capaz de armazenar dados. O que diferencia este cartão de cartões de crédito por exemplo, é o fato de o cartão possuir um máquina virtual Java (JVM) instalada. A JVM faz possível o cartão possuir aplicativos em sua memória, conhecidos como applets, e dependendo do programa que os acessar, pode escolher qual deles executar. Estes applets são diferentes dos aplicativos Java desenvolvidos para computador, pois devem levar em conta as limitações de processamento e memória dos cartões (CHEN: 2000) .

Para tal, existe uma biblioteca de desenvolvimento independente conhecida como Java Card Development Kit oferecida pela empresa Oracle, que também disponibiliza o kit de desenvolvimento java convencional. Dentro do cartão, os applets conversam com o

terminal por meio de mensagens, chamadas APDU's, que são em sua maioria padronizadas em formatos específicos. O Chip de contato fornece energia e faz a comunicação entre a leitora ou terminal e o processador.

Estes cartões possuem diversos mecanismos de segurança embutidos na própria JVM, por exemplo, os aplicativos rodam isolados uns dos outros, portanto os dados de um aplicativo nunca poderão serem lidos por outro. Os cartões também suportam funções de criptografia, que dependem dos aplicativos para serem utilizadas(CHEN: 2000).

2.7 O applet e sua estrutura lógica de dados

A biblioteca JMRTD possui consigo um applet a ser instalado e executado no cartão para o funcionamento da biblioteca. Este applet foi feito juntamente com a biblioteca no padrão ICAO9303, portanto ele pode ser lido inclusive por outras bibliotecas que implementem o padrão. Este applet tem como função armazenar todas as informações referentes a um passaporte, e portanto possui uma estrutura de dados lógica interna descrita pelo padrão ICAO9303:

- DG1 — Informação da Zona Legível por Máquina
- DG2 — Características Faciais (Foto)
- DG3 — Informações de identificação adicionais (Digitais)*
- DG4 — Informações de identificação adicionais (Iris)*
- DG5 — Fotografia impressa na frente do cartão*
- DG6 — Reservado para uso futuro*
- DG7 — Assinatura escrita digitalizada*
- DG8 — Características de dados* **
- DG9 — Características estruturais* **
- DG10 — Características substanciais*
- DG11 — Detalhes pessoais adicionais*
- DG12 — Detalhes do documento adicionais*
- DG13 — Detalhes opcionais*
- DG14 — Informações da chave pública para Controle de Acesso Extendido***
- DG15 — Informações da chave pública para Autenticação Ativa***
- DG16 — Pessoas para contato*

* - Opcional
** - Ainda não definido
*** - Condicional, apenas se suportado

Além destes existem mais dois arquivos que não armazenam dados do usuário mas sim dados do cartão, estes são: COM e SOD.

O arquivo COM, ou Arquivo de Cabeçalho e de Presença de Grupos de Dados, possui a função de armazenar a presença dos arquivos de dados, ou seja, quais arquivos estão presentes no cartão, e informações de versionamento do sistema de arquivos do cartão.

O arquivo SOD, ou Document Security Object - Objeto de Segurança do Documento, possui a função de armazenar o resultado da função hash de cada arquivo, ou seja, no processo de emissão do cartão, cada arquivo é passado também pela função hash, e seu resultado é armazenado dentro do arquivo SOD. Ele também armazena todas as informações de segurança do cartão como os algoritmos utilizados para a função hash e criptografia.

O cartão possui também uma zona que é ilegível por meios externos, informações sensíveis a criptografia como chaves privadas são armazenadas ali na sua criação, e depois só podem ser lidas e utilizadas pelo próprio cartão.

3 Trabalhos Correlatos e Revisão Bibliográfica

Várias bibliotecas e aplicativos funcionais existem, que fazem possível a leitura de passaportes eletrônicos, ou por via de chips ou de NFC (Near Field Communication), escritos em várias linguagens de programação onde muitos poucos permitem a emissão e personalização destes passaportes. Para a linguagem Java, vários se utilizam também da biblioteca JMRTD.

A pesquisa foi efetuada utilizando-se a plataforma de busca Google, com as seguintes frases pesquisadas, sendo analisada toda a primeira e segunda páginas da busca:

- "e-passport reader python"
- "e-passport reader java"
- "e-passport reader c++"
- "e-passport creation"
- "passport creation python"
- "passport creation java"

Com isso, as bibliotecas encontradas foram:

3.1 pypassport

É uma biblioteca desenvolvida em python que permite a interação com passaportes eletrônicos. Semelhante a biblioteca JMRTD, ela permite a leitura das informações e execução dos protocolos BAC, PA e AA. Ele também permite a criação do conteúdo de um passaporte eletrônico, porém não permite sua personalização (HOUZARD and ROGER: 2009).

Sua última versão, 1.0, foi lançada em 30 de Agosto de 2009, possui as funcionalidades de leitura dos grupos de dados e executar os protocolos BAC, PA, AA, além de poder criar informações de um passaporte e para isso utiliza JAVA, mas não possui a capacidade para escrever estas informações em um cartão, nem possui um applet conjunto para tal. Esta biblioteca também possui métodos para salvar os dados lidos de um passaporte em formato XML, ou texto.

3.2 e-passport viewer

Aplicativo Android que se utiliza da biblioteca pypassport, desenvolvido pelos mesmos autores da biblioteca como parte de seu mestrado na UCL, Bélgica. Este aplicativo é uma GUI construída em cima da biblioteca, e portanto possui como funcionalidades: ler as informações do cartão, salvar as informações lidas como texto, exportar parte das informações (Foto, Assinaturas, Certificados), salvar um sumário das informações em XML ou PDF, e verificar a integridade e autenticidade dos dados do cartão (HOUZARD and ROGER: 2013).

3.3 readid

Um aplicativo android desenvolvido pela empresa InnoValor para leitura de passaportes eletrônicos em smartphones com capacidade NFC. Possui internamente um leitor OCR, o que facilita a extração das informações do MRZ para a execução do protocolo BAC. Possui também a capacidade de executar PA e AA, e mostrar todas as informações sobre o passaporte na tela da aplicação, incluindo MRZ, Foto, Certificados e dados dos arquivos COM e SOD e resultado da execução dos protocolos.

O aplicativo também suporta carteiras de motorista eletrônicas, limitadas a área da Alemanha. Por ter sido desenvolvido especialmente com este propósito, não provê funcões de personalização nem criação de passaportes. (InnoValor: 2013)

3.4 e-passport NFC reader

Um aplicativo Android desenvolvido como hobby e aprendizado para leitura de passaportes eletrônicos em smartphones com capacidades NFC. Se utiliza da biblioteca JMRTD para seu funcionamento, permitindo a execução dos protocolos BAC, e PA. Também permite ser utilizado por aplicativos Android de terceiros, por via de uma api aberta. Como é um aplicativo mobile, não permite a personalização de passaportes.(TANANAEV: 2016)

3.5 epassport reader

Um programa desenvolvido em Java como material de aprendizado e prova de conceito para a leitura de passaportes em dispositivos Android. Se utiliza da biblioteca JMRTD para seu funcionamento e execução do protocolo BAC, porém não permite a personalização de passaportes.(BOZHANOV: 2016)

3.6 Considerações

Cada aplicativo encontrado suporta as operações básicas de leitura e autenticação dos passaportes, lendo e mostrando as informações contidas nos arquivos DG 1 e DG 2, além de realizarem todos o protocolo BAC. Porém nenhum dos aplicativos encontrados possui a funcionalidade de personalizar um passaporte em branco.

O trabalho que mais se relaciona com esta proposta é o e-passport viewer, de (HOUZARD and ROGER: 2013). Este que é um aplicativo para computador e checagem de fronteira que se utiliza de uma biblioteca criada pelos próprios desenvolvedores para a leitura de passaportes. O aplicativo permite a criação de um objeto simulado onde se faz a personalização de um passaporte, porém ele não possui a funcionalidade de enviar estas informações para um cartão ou passaporte, já que não possui um applet próprio para tal.

Desta forma, o aplicativo desenvolvido decorrente deste trabalho expande a funcionalidade dos aplicativos existentes, ao possuir como funcionalidade principal a criação de passaportes, além da funcionalidade de leitura, onde caso venha a ser utilizado oficialmente, poderá ser separado nos dois módulos, para a criação pelo órgão emissor, e leitura pelo órgão de fronteira.

Nas pesquisas foram encontrados mais dois aplicativos correlatos: Golden Reader Tool, da BSI Alemanha e wzMRTD, de wazaa.org, estes que não puderam ser avaliados, pois seus respectivos domínios não se encontram mais acessíveis.

Nome:	E-Passport Viwer	ReadID	e-passport NFC Reader	Epassport Reader
Linguagem:	Python	Java	Java	Java
Plataforma:	PC	Android	Android	Android
Leitura:	Sim	Sim	Sim	Sim
Escrita:	Não	Não	Não	Não
Protocolos:	BAC, PA, AA	BAC, PA	BAC	BAC

Tabela 1: Comparação entre os aplicativos relacionados à leitura de passaportes

4 Desenvolvimento

O projeto foi desenvolvido tendo como base a versão 0.4.9 da biblioteca JMRTD descrita na próxima sessão, esta que possui um aplicativo executável mostrando as funcionalidades da biblioteca, que podia ler um passaporte e escrever em um cartão que possuísse o aplicativo, podendo apenas utilizar o protocolo BAC. Foi feita uma extensa pesquisa sobre os documentos ICAO 9303, e a documentação da biblioteca JMRTDpara se iniciar o projeto, incluindo sobre os protocolos.

4.1 As bibliotecas utilizadas

JMRTD - *Java Machine Readable Travel Documents*: é uma biblioteca para a criação, edição, e aferição de passaportes eletônicos escrita na linguagem de programação Java. Ela foi desenvolvida juntamente com a biblioteca Scuba, que permite a comunicação com cartões eletrônicos compativeis com o padrão ISO 7816, que descreve os padrões de comunicação entre o terminal e o cartão. Ela possui a capacidade de criar, e editar novos passaportes, além que resgatar as informações de um passaporte pronto (OOSTDIJK: 2010).

SCUBA - *Smart Card Utilities for Better Access*: é uma biblioteca para a comunicação com cartões eletrônicos da plataforma javacard, escrita na linguagem de programação Java. Ela faz a ponte entre o JMRTDe o javacard possuindo a capacidade de transmitir mensagens e prover a comunicação com os cartões (*Smart Card Utilities for Better Access, Project SCUBA*).

STASM - *Active Shape Models with STASM*: é uma biblioteca de reconhecimento facial escrita na linguagem de progrmação C++. Ela se utiliza de modelos de formas para reconhecer um rosto em uma foto e retirar os pontos de referência do rosto para que depois seja feito o reconhecimento da pessoa, fazendo uso da biblioteca Open-CV. Possui a maioria dos pontos de reconhecimento parecida com o padrão ISO(*Information technology — Biometric data interchange formats — Part 5: Face image data* 2008), e foi escolhida por ser de fácil utilização e possuir uma extensa documentação com exemplos mínimos disponíveis, facilitando a sua compreensão, além de ser complementar à biblioteca Open-CV (Milborrow and F.Nicolls: 2014).

LIBFPRINT - É uma biblioteca de coleção e verificação de digitais biométricas escrita na linguagem de programação C. Ela provê funções para se coletar uma digital e guardá-la como uma imagem, e depois extrair as minutas desta imagem para que se compare com uma digital recém tirada. Foi escolhida por ter uma documentação extensa, exemplos mínimos e ter sido recomendada pelo orientador deste projeto (*LibFPrint*).

OPEN-CV - *Open Source Computer Vision Library*: é uma biblioteca para manipulação de imagens e visão computacional escrita na linguagem de programação C++ e Java. Ela possui funcionalidades para conversão de imagens, e acessar a câmera do dispositivo para tirar fotos ou videos. Ela também provê visão computacional, permitindo identificar formas e rostos na imagem. A biblioteca Stasm amplia a funcionalidade para rostos, detectando mais pontos de interesse no rosto da pessoa, e foi escolhida por ser

a mais bem recomendada biblioteca de manipulação de imagens e visão computacional, com relação a sua utilização (*OpenCV Library*).

EJBCA - *Open Source PKI Certificate Authority*: é uma biblioteca para criação e manejo de certificados digitais auto-assinados escrita na linguagem de programação Java. É necessária para o funcionamento da biblioteca JMRTD, para criação dos certificados verificáveis por cartão (CVC), necessários para a execução dos protocolos de segurança do cartão (*EJBCA Open Source PKI Certificate Authority*).

BOUNCYCASTLE - *Bouncy Castle Security Provider*: é um provedor de segurança, uma biblioteca que possui funções criptograficas e de geração e acordo de chaves para criptografia escrita na linguagem de programação Java. Toda a segurança da biblioteca do JMRTDé feita com ele, portanto por motivos de compatibilidade foi-se utilizado o bouncycastle como provedor para o sistema final (*Bouncy Castle*).

LDAPTIVE - é uma biblioteca modular para a leitura de arquivos no formato ldif. Este formato, descrito no *RFC 2849*, é utilizado pela ICAO para os arquivos contendo os certificados necessários aos dispositivos de fronteira em postos de checagem de passaportes. A biblioteca possui funções para ler as entradas e separar seus atributos, facilitando a obtenção dos certificados (*LDAPtive*).

4.2 Os protocolos de segurança

Para o acesso as informações do cartão diversos protocolos de segurança devem ser efetuados para garantir a troca segura de mensagens entre o terminal e o cartão, e ter a certeza de que o terminal e o cartão não foram adulterados de alguma maneira. Estes protocolos devem ser corretamente configurados durante a criação do cartão para que garantam a segurança das informações que nele estão guardadas. Todos os protocolos estão completamente descritos no documento ICAO 9303 parte 11 e no documento BSI-TR 03110-1 (*Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token* – 2015).

BAC - *Basic Access Control* - Controle de Acesso Básico Antes de fazer qualquer tipo de operação no cartão, deve-se efetuar este protocolo. Ele utiliza as informações do número do documento, data de nascimento e de validade do cartão para criar chaves simétricas seguras para a troca de mensagens entre o terminal e o cartão em uma semântica de desafio resposta, criando assim um canal seguro de comunicação. Esta informação estará impressa na frente do cartão, e portanto a execução com sucesso deste protocolo não só garante um canal de comunicação seguro, mas também confirma que as informações impressas batem com as armazenadas no cartão.

PACE - *Password Authenticated Connection Establishment* - Este protocolo deve ser utilizado se suportado pelo cartão, e deve ter preferência sobre o BAC. Ele consiste em um acordo de chaves Diffie-Hellman em que a troca de mensagens é criptografada com uma senha simples, derivada das mesmas informações do MRZ que o protocolo BAC, garantindo uma maior segurança contra escuta. Após a execução deste protocolo um canal de mensagem seguro é criado, além de confirmar as informações do MRZ. Caso não seja suportado este protocolo, BAC deve ser executado em seu lugar.

PA - *Passive Authentication* - Autenticação Passiva Este protocolo faz uso dos arquivos COM e SOD para seu funcionamento. O arquivo SOD armazena a hash de cada Grupo de Dados presente no cartão, e o arquivo COM indica sua presença. Desta forma o protocolo se inicia verificando se para cada arquivo cuja presença esteja indicada em COM, incluindo o próprio arquivo COM, se sua hash coincide com a hash armazenada em SOD. Se todas coincidirem então nenhum arquivo foi alterado desde a fabricação do cartão. O segundo passo consiste em utilizar o certificado de assinatura encontrado no SOD, construindo uma corrente de certificação até um certificado assinado por uma CA reconhecida, e garantindo que cada certificado da corrente seja válido. Passados por estes dois passos então se pode confirmar que os dados do cartão são válidos e não foram alterados.

AA - *Active Authentication* - Autenticação Ativa Este protocolo deve ser executado após o PA, pois ele confirma que o SOD foi lido de um cartão com um chip válido. Ele consiste em uma troca de mensagens de desafio-resposta onde a aplicação envia uma mensagem ao cartão e este responde com esta mensagem cifrada pela chave privada do cartão. A aplicação então decifra utilizando a chave pública armazenada no arquivo DG 15 e, caso a resposta coincida com a mensagem enviada, então o chip não foi adulterado, já que a chave privada interna do cartão, inacessível externamente, é o par da chave pública gravada no arquivo DG 15.

EAC - *Extended Access Control* - Controle de Acesso Extendido Este protocolo deve ser feito após o BAC ou PACE e é necessário para se obter acesso as biometrias adicionais do cartão. Ele consiste em autenticar o terminal para o cartão, e o cartão para o terminal em duas etapas chamadas Chip Authentication e Terminal Authentication respectivamente, utilizando-se de um par de chaves assimétricas. Etapas que por si só são dois protocolos separados. Ao término do protocolo um canal de comunicação de maior segurança entre o cartão e o terminal é gerado, e o acesso às biometrias adicionais é liberado. Este protocolo substitui o protocolo AA, pois evita a utilizaço de semântica de mensagem, onde a manipulação do conteúdo do desafio pode vir a revelar a chave utilizada.

Protocolo	Função	Observação
BAC PACE	Gera canal de comunicação seguro e previne leitura não autorizada do cartão	Não previne substituição do chip, ou cópia idêntica dos arquivos internos
AA EAC (CA)	Previne cópia do arquivo SOD e garante que o chip é autêntico e não foi substituído	Não previne acesso não autorizado
PA	Garante a autenticidade do conteúdo dos arquivos do cartão	Não previne cópia nem substituição do chip Não previne acesso não autorizado
EAC (TA)	Permite o acesso as biometrias adicionais, previne leitura não autorizada a essas biometrias	Não previne cópia nem substituição do chip

Tabela 2: Funções dos protocolos de segurança (International Civil Aviation Organization: 2015)

4.3 JNI

Java Native Interface ou JNI é uma interface que visa a utilização de código em C/C++ juntamente com o código em Java. Ela funciona mudando o contexto de execução para C/C++ apenas na hora de executar o código, e depois retorna ao contexto Java. Isso faz com que a performance da execução dos códigos via JNI tenham uma performance mais parecida com C/C++.

Para utilizar JNI, primeiro se deve criar uma classe que possua métodos com a palavra de comando *native*, que será chamada de classe nativa Java, depois, utilizar o comando *javah -o ¡nome da lib criada¿.h -classpath ¡pasta para salvar o arquivo¿ ¡classe nativa Java¿* para que a JNI crie uma classe *.h* ou *.hpp* que deverá ser implementada em C/C++. Esta classe criada não deve ser alterada, pois foi gerada pelo programa, sua implementação deve ser realizada em uma classe *.c/.cpp* correspondente, copiando exatamente a assinatura dos métodos.

Uma nota a se fazer, dentre as entradas dos métodos nativos, existem uma entrada extra feita pelo programa, que não deve ser preenchida durante as chamadas, chamado de *JNIEnv*, que é o contexto JNI. As outras entradas estão envolvidas em *wrappers* que os caracterizam como objetos Java, já que as linguagens representam seus tipos de maneiras diferentes, assim *int* em C não é diretamente convertido para um *int* em Java. Estes objetos possuem funções para convertê-los em tipos C/C++ para poderem ser trabalhados, mas depois, é necessário utilizar o objeto JNIEnv para envolver o retorno num *wrapper* específico para o tipo do retorno, antes de retornar da função, para o contexto Java.

Outra nota a se fazer, como o contexto é mudado durante a execução do código em C/C++, não há como o código em C/C++ enviar mensagens para o código em Java, sem retornar da função.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class stasmlib_StasmController */

#ifndef _Included_stasmlib_StasmController
#define _Included_stasmlib_StasmController
#ifdef __cplusplus
extern "C" {
#endif
/*
* Class:     stasmlib_StasmController
* Method:    getImageFeaturePoints
* Signature: (Ljava/lang/String;)[F
*/
JNIEXPORT jfloatArray JNICALL
    Java_stasmlib_StasmController_getImageFeaturePoints
    (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

Excerto de código 1: Arquivo gerado pelo JNI para Stasm (.h)

```
#include "StasmBridgeLib.h"
... includes ...

JNIEXPORT jfloatArray JNICALL
    Java_stasmlib_StasmController_getImageFeaturePoints
        (JNIEnv *env, jobject obj, jstring filename) {

    char* path = (char*) env->GetStringUTFChars(filename, 0);

    cv::Mat_<unsigned char> img(cv::imread(path,
        CV_LOAD_IMAGE_GRAYSCALE));

    if (!img.data) {
        printf("Cannot load %s\n", path);
        exit(1);
    }

    int foundface;
    // x,y coords (note the 2)
    float landmarks[2 * stasm_NLANDMARKS];
    if (!stasm_search_single(&foundface, landmarks,
            (const char*) img.data, img.cols, img.rows,
                path, "../data")) {
        printf("Error in stasm_search_single: %s\n",
            stasm_lasterr());
        exit(1);
    }

    if (!foundface)
        printf("No face found in %s\n", path);
    else {
        // draw the landmarks on the image as
            white dots (image is monochrome)
        stasm_force_points_into_image(landmarks, img.cols, img.rows);
        for (int i = 0; i < stasm_NLANDMARKS; i++)
            cv::circle(img,cv::Point(cvRound(landmarks[i * 2]),
                cvRound(landmarks[i * 2 + 1])),
                    2,cv::Scalar(255,255,255),-1);
    }

    cv::imwrite("minimal.jpg", img);
    cv::imshow("stasm minimal", img);
    cv::waitKey();

    jfloatArray ret = env->NewFloatArray(stasm_NLANDMARKS * 2);

    jfloat* body = env->GetFloatArrayElements(ret, 0);

    for (int i = 0; i < stasm_NLANDMARKS * 2; i++) {
        body[i] = landmarks[i];
```

```
    }

    env->SetFloatArrayRegion(ret,0,stasm_NLANDMARKS * 2, body);

    return ret;

}
```

Excerto de código 2: Arquivo que implementa o .h do JNI utilizado para Stasm (.c)

Depois do código estar pronto, deve se compilar os arquivos com o compilador adequado (*gcc/g+* com a flag *-shared*, e dentro da seção de diretórios a serem incluidos, incluir as pastas *include e include/linux*, do local de instalação do Java na máquina. Dentro da pasta *include* deve existir o arquivo *jni.h*, e por útlimo, utilizar a flag *-o* para nomear a biblioteca *¡nome desejado¿.so*.

Isto criará a biblioteca utilizável pelo JNI. Com isso, é necessário agora carregar a biblioteca no programa Java com a função *System.LoadLibrary*, e então, quando os métodos da classe nativa forem chamados, eles serão executados pela biblioteca que foi criada em C/C++.

4.4 Início do Desenvolvimento

O programa foi desenvolvido com a linguagem de programação Java, utilizando o sistema JNI para a integração de bibliotecas que existem somente em C e C++. Utilizando uma parte do código da versão 0.4.9 da biblioteca JMRTD, é possível enviar informações para o cartão, desta forma pode-se fácilmente coletar e enviar as informações. A biblioteca OpenCV foi usada para o processamento da foto da pessoa, e possui funções para tirar a foto com a camera da máquina. Com a biblioteca Stasm, se pode retirar os pontos faciais da pessoa, e depois tratá-los para se adequarem ao padrão ICAO. Foi necessário utilizar a API nativa do java para integrar as bibliotecas Stasm e libfprint ao projeto, pois elas existem apenas em C. Com a biblioteca libfprint foi possível a extração da digital para sua inserção.

Foi estipulado como objetivo primário, reproduzir os resultados de um cartão previamente personalizado pelo programa da versão 0.4.9 do JMRTD. Desta forma iniciou-se o projeto, seguindo um padrão de desenvolvimento semelhante ao SCRUM, onde com reuniões quinzenais com o orientador, foram-se delineando pequenas metas para cada quinzena.

A primeira parte a ser desenvolvida foi a conexão com o cartão. Para isso foi criada a classe CardCom, que foi utilizada para testar e aprender as funcionalidades da biblioteca JMRTD, com o objetivo principal de implementar a conexão com o cartão. Para isso são utilizados métodos e objetos da biblioteca javax.smartcardio, SCUBA e JMRTD para conectar com o cartão e abrir um serviço de comunicação, com um objeto da classe PassportService, da biblioteca JMRTD. Este objeto possui métodos para enviar APDUs que permitem a execução de protocolos e a leitura do cartão.

```
//Abre uma terminal factory para detectar os terminais
TerminalFactory terminal = TerminalFactory.getDefault();
//listam-se eles
List<CardTerminal> readers = terminal.terminals().list();
//escolhe-se a primeira
```

```
CardTerminal reader = readers.get(0);

for (int i = 0; i < 3 && !reader.isCardPresent(); i++) {
    //Se espera conectar um smartcard
    reader.waitForCardPresent(5000);
}
if (reader.isCardPresent()) {
    service = new PassportService(
        new TerminalCardService(reader));
    service.open();
```

Excerto de código 3: CardCom - Conexão com o cartão

Depois de efetuada a conexão, iniciou-se o desenvolvimento da funcionalidade de leitura, dentro da mesma classe, em um formato que permitisse a execução de testes sempre que adicionado algo novo. Antes de começar a leitura, todas as funções de segurança da biblioteca JMRTD se utilizam do provedor de segurança BouncyCastle, então ele foi adicionado a lista de provedores Java.

Para cada arquivo do applet, foi feita uma função para sua leitura. Em todas as funções, o início é semelhante, onde se deve utilizar o *PassportService* para acessar uma *InputStream* para o arquivo, e dela criar o objeto do arquivo, depois de criado o arquivo a manipulação é diferente, já que cada arquivo possui estruturas internas diferentes.

A função *canSelectFile* utilizada no excerto envia uma APDU com a instrução *SelectFile* para o cartão. Se a resposta da mensagem for 0x9000, ou seja, sucesso, então o arquivo existe e é selecionável, assim, é possível abrir uma *InputStream* para o arquivo.

```
private void readDG1() throws CardServiceException, IOException {
    InputStream dg1Stream;
    DG1File input;

    if (canSelectFile(service.EF_DG1)) {
        System.out.println("DG1 Presente");
        dg1Stream = service.getInputStream(service.EF_DG1);
        input = new DG1File(dg1Stream);
        System.out.println(input.getMRZInfo().toString());
    } else {
        System.out.println("Nao foi possivel acessar o arquivo DG1");
    }
}
```

Excerto de código 4: Código de leitura do arquivo DG1

Após a leitura de cada arquivo, e aferidas que as leituras estavam corretas via *System.out.prints* iniciou-se o processo de escrita. Para isso, foi utilizada a classe *PassportPersoService* da versão 0.4.9 da biblioteca JMRTD. Esta que é utilizada para a personalização do cartão, e possui funções para enviar os arquivos e configurar os protocolos de segurança.

Assim, ainda na classe CardCom, foram-se desenvolvidas as funções de escrita, uma para cada tipo de arquivo, e para o protocolo BAC. Para cada arquivo, a função é semelhante tal qual as funções de leitura, onde após preenchido um objeto do arquivo, deve-se utilizar a classe *PassportPersoService* para enviar as APDUs de *createFile* para o cartão

alocar o espaço necessário para o arquivo e anotá-lo na tabela de endereços, depois o comando *selectFile* para que o arquivo seja escrito com o comando *writeFile*.

```
private void SendDG1() throws CardServiceException, IOException {

    //Cria um bloco MRZ com os dados coletados.
    MRZInfo info = new MRZInfo(code, issuingState,
        lastName, firstNames, DOCUMENTNUMBER,
        nationality, DATEOFBIRTH, gender, DATEOFEXPIRY, cpf);
    System.out.println(info.toString());
    //Com o bloco MRZ cria-se um arquivo DG1
    DG1File dg1 = new DG1File(info);

    if (!perso.isOpen()) {
        perso.open();
    }

    perso.createFile(service.EF_DG1,
        (short) dg1.getEncoded().length);
    perso.selectFile(service.EF_DG1);

    System.out.println("Enviando arquivo DG1");
    perso.writeFile(service.EF_DG1,
        new ByteArrayInputStream(dg1.getEncoded()));

}
```

Excerto de código 5: Código de escrita do arquivo DG1

Para o envio das imagens, e consequentemente o arquivo DG2, a imagem é carregada do computador em formato jpeg utilizando a biblioteca *ImageIO* para depois preencher o objeto DG2 e enviá-lo ao cartão. Para preencher o objeto DG2 a imagem deve estar em formato binário. Para isso, a imagem tem de ser convertida a um *byte array* para ser enviada ao cartão.

Também se iniciaram os testes para o reconhecimento facial, a retirada de pontos faciais da foto. A Java Native Interface (JNI) foi utilizada para criar uma ligação entre a biblioteca Stasm escrita em C++, compilada com a ferramenta JNI. O código foi baseado no exemplo mínimo existente no site da biblioteca, que cumpre com as necessidades do trabalho.

Após retirados os pontos faciais, verificou-se que, o padrão ICAO exige que, para estes pontos seja seguido o padrão ISO/IEC 19794-5 para identificação dos pontos. A biblioteca Stasm encontra os pontos de uma forma diferente, e por isso foi necessário realizar um mapeamento dos pontos encontrados pelo Stasm para os pontos presentes no padrão ISO, sendo que alguns pontos a biblioteca Stasm identificou a menos, outros a mais. A função *resolveFPS* faz este mapeamento, que está descrito no Anexo A ao final deste trabalho.

```
private void SendDG2() throws IOException, CardServiceException {

    //Escolher a foto
    File file = null;
```

```
JFileChooser chooser = new JFileChooser();
int choice = chooser.showOpenDialog(null);

if (choice == JFileChooser.APPROVE_OPTION) {
    file = chooser.getSelectedFile();
}

ArrayList<FaceInfo> faces = new ArrayList<>();
ArrayList<FaceImageInfo> images = new ArrayList<>();

if (file == null) {
    return;
}

//Reconhecimento facial
float[] extractedFeatures =
new stasmlib.StasmController().
getImageFeaturePoints(file.getPath());

FeaturePoint[] fps = resolveFPS(extractedFeatures);

//Imagem para o cartao
BufferedImage portrait = ImageIO.read(file);
//Carrega a imagem jpg em java
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ImageIO.write(portrait, "jpg", baos);
baos.flush();
//Transforma em byteArray
byte[] imageBytes = baos.toByteArray();
//-Imagem para o cartao

FaceImageInfo i1 = new FaceImageInfo(
    //Gender
    Gender.UNSPECIFIED,
     //EyeColor
    FaceImageInfo.EyeColor.UNSPECIFIED,
    //FeatureMask
    0x0000,
    //HairColor
    (int) FaceImageInfo.HAIR_COLOR_UNSPECIFIED,
     //Expression
    (int) FaceImageInfo.EXPRESSION_UNSPECIFIED,
    //poseAngle
    new int[3],
    //poseAngleUncertainty
    new int[3],
    //faceImagetype
    (int) FaceImageInfo.FACE_IMAGE_TYPE_BASIC,
    // image color space
    (int) FaceImageInfo.IMAGE_COLOR_SPACE_UNSPECIFIED,
    //image source
```

```
            (int) FaceImageInfo.SOURCE_TYPE_UNKNOWN,
            //deviceType
            0x0000,
            //quality
            0x0000,
             //featurePoints
            fps,
            //Image Dimensions
            portrait.getWidth(), portrait.getHeight(),
            //ImageInputstream
            new ByteArrayInputStream(imageBytes),
            //imageLength
            imageBytes.length,
            //nova foto
            FaceImageInfo.IMAGE_DATA_TYPE_JPEG);

        images.add(i1);

        FaceInfo f1 = new FaceInfo(images);

        faces.add(f1);

        DG2File dg2 = new DG2File(faces);

        if (!perso.isOpen()) {
            perso.open();
        }

        perso.createFile(service.EF_DG2,
            (short) dg2.getEncoded().length);
        perso.selectFile(service.EF_DG2);

        System.out.println("Enviando arquivo DG2");
        perso.writeFile(service.EF_DG2,
            new ByteArrayInputStream(dg2.getEncoded()));

}
```

Excerto de código 6: Código de escrita do arquivo DG2

Para o envio de digitais, e consequentemente o arquivo DG3, foi utilizada a interface JNI novamente, para se conectar com a biblioteca libfprint. Esta teve de ser um pouco modificada, para que duas funções internas da API ficassem visíveis externamente antes que a biblioteca fosse montada, para ser usada na compilação da classe JNI. Estas funções retiram as minutas de uma imagem, e são utilizadas quando se tira uma digital, para gerar um array de minutas, mas como o padrão ICAO9303 não prevê salvar as minutas no cartão, apenas a imagem, elas são descartadas, sendo a imagem da digital salva como um arquivo, e depois carregada para o cartão. Estas funções então são utilizadas para realizar a verificação da digital. O conteúdo da classe .h também foi baseado nos exemplos mínimos disponíveis, pois eles já supriam as necessidades do programa, incluindo de verificação da digital.

Após a leitura de uma digital, a biblioteca gera uma imagem em formato .pgm, que é

então codificada em formato jpeg para ser armazenada. Já que não foi possível por grande parte do projeto realizar o protocolo EAC, uma alteração no arquivo do applet teve de ser feita para que fosse pulada a verificação se o protocolo foi bem sucedido antes de se acessarem as digitais, permitindo assim que estas informações sejam acessadas apenas com o BAC. Isto foi revertido para manter a integridade e segurança do cartão.

```
...
//Ler a imagem direto como byteArray
byte[] img = FileUtils.readFileToByteArray(
    new File("finger_standardized.pgm"));
Mat temp = Imgcodecs.imdecode(
    new MatOfByte(img),
    Imgcodecs.CV_LOAD_IMAGE_UNCHANGED); //decodifica em uma mat

//----------------Converter Para Jpeg----------------------------------

Imgproc.resize(temp, temp,
    new Size(), 0.5, 0.5, Imgproc.INTER_LINEAR);

//com este parametro
MatOfInt params = new MatOfInt(
    Imgcodecs.CV_IMWRITE_JPEG_QUALITY, 30);

//Salva a imagem em formato JPEG
Imgcodecs.imwrite(finger + ".jpg", temp, params);

// Ler a imagem JPG direto como byteArray
byte[] imgJPG = FileUtils.readFileToByteArray(
    new File(finger + ".jpg"));

System.out.println(imgJPG.length);

//-------------------- Fim leitura da imagem
//Cria uma entrada de digital para um dedo
return new FingerImageInfo(finger + 1,
        1, 1, 100, //view count, view number, quality%
        FingerImageInfo.IMPRESSION_TYPE_SWIPE, //impression type
        temp.width(), temp.height(), //Dimensions w,h
        new ByteArrayInputStream(imgJPG), //image bytes
        imgJPG.length, //image size in bytes
        FingerInfo.COMPRESSION_JPEG); //compression type
}
```

Excerto de código 7: Código da leitura de uma digital

Nesta parte do desenvolvimento, a classe *CardCom* foi criada com o intuito de prover a comunicação, mas durante o andamento do projeto, acabaram sendo adicionados os diversos testes e aprendizados sobre o sistema nesta classe. Desta forma a classe possuía não só funções para a conexão, mas toda a funcionalidade de leitura e escrita com o applet. Foi decidido então refatorar o código para separar esta classe em, *CardConnection, CardReader, CardWriter, SecurityProtocols*, cada uma com as funções de conectar com o cartão, ler, escrever e realizar os protocolos de segurança, respectivamente. Os códigos

referentes as funções ficaram praticamente inalterados.

Como não é possível a troca de mensagens entre a interface nativa (JNI) e a aplicação, quando é necessária a execução de uma função nativa não é possível, por exemplo, enviar um sinal a uma Thread Java avisando do início da captura da digital. Aparentemente, também, alguns tipos de Threads, como as de repintura de janelas, parecem parar de funcionar durante a execução da biblioteca em C.

Como alguns protocolos de segurança se utilizam de certificados, foi criada uma classe chamada *myCertificateFactory*, que é responsável por criar correntes de certificados de teste para escrever os cartões, armazenando-os em uma *KeyStore* Java para serem enviados ao cartão, e de carregar os diversos certificados disponíveis pelas listas de certificados públicas da ICAO e do MRE brasileiro, para a execução dos protocolos.

Para a criação de certificados, é utilizada a classe *JcaX509v3CertificateBuilder* da biblioteca BouncyCastle, ela possui funções de preenchimento e criação de certificados auto assinados e assinados por outros certificados. A classe auxiliar *JcaContentSignerBuilder* cria um objeto *ContentSigner*, que é utilizado para assinar um certificado. No caso de um certificado autoassinado, a chave privada seria a do próprio certificado, e não de outro que o assina.

```
public Certificate generateSignedX509Certificate
(X500Name dnName, long validity, PublicKey pk,
PrivateKey sk, String signatureAlgorithm,
boolean isCA) throws CertificateException,
InvalidKeyException, SignatureException,
NoSuchAlgorithmException, NoSuchProviderException {

    Date lastDate;
    try {

        if (pk == null) {
            throw new NullPointerException("Public Key NULL");
        }
        if (sk == null) {
            throw new NullPointerException("Private Key NULL");
        }

        long now = System.currentTimeMillis();
        Date startDate = new Date(now);

        //Usa Timestamp como numero do certificado
        BigInteger certSerialNumber =
            new BigInteger(Long.toString(now));

        Calendar calendar = Calendar.getInstance();
        calendar.setTime(startDate);
        calendar.add(Calendar.YEAR, 10); // <-- 10 Yr validity

        Date endDate = calendar.getTime();

        ContentSigner contentSigner =
            new JcaContentSignerBuilder(
            signatureAlgorithm).build(sk);
```

```
            JcaX509v3CertificateBuilder certBuilder =
                new JcaX509v3CertificateBuilder(dnName,
                certSerialNumber, startDate, endDate, dnName, pk);

            // Extensions --------------------------
            // Basic Constraints
            BasicConstraints basicConstraints =
                new BasicConstraints(isCA); // <-- true for CA

            certBuilder.addExtension(
                new ASN1ObjectIdentifier("2.5.29.19"),
                true, basicConstraints);
            // Basic Constraints is usually marked as critical.

            // -------------------------------------
            return new JcaX509CertificateConverter().
                setProvider(bcProvider).
                getCertificate(certBuilder.build(contentSigner));

        } catch (Exception e) {

            throw new CertificateEncodingException("getSelfCert: "
                    + e.getMessage());
        }
    }
```

Excerto de código 8: Função de criação de um certificado da classe MyCertificateFactory

Foram criadas então, janelas e classes dedicadas à captura da foto e das digitais, e uma janela principal para acessar as funcionalidades de escrita ou leitura, também classes para a verificação dos certificados.

4.5 Aprofundamento da coleta da foto e das digitais

Algumas características específicas tiveram de ser levadas em conta na hora de tirar a foto e as digitais, para garantir que o padrão ICAO9303 fosse obedecido, e que o cartão pudesse comportar toda informação necessária.

Para a foto, utlizou-se a biblioteca *OpenCV* para acessar a câmera por via do objeto *VideoCapture* da biblioteca *OpenCV*, no Excerto de Código número 9 a variável cv, em uma thread que mostra o vídeo na tela. Quando se aperta o botão para tirar a foto, ela é primeiro codificada em jpeg com 100% de qualidade, e é passada para a biblioteca Stasm para a retirada dos pontos de reconhecimento facial. Depois, se codifica novamente a foto original em jpeg mas desta vez com 50% da qualidade, após ter sido escalada em 50% nos dois eixos cartesianos para que esta seja enviada ao cartão. Isto efetivamente reduz o tamanho da imagem enviada a pelo menos um quarto do seu tamanho, o que garante que sobre espaço de armazenamento para as outras imagens. O padrão ICAO 9303 cobre também o formato da foto, que deve ser JPEG-2000, com extensão *.jp2* ou *.j2k*. Este não é um formato comumente utilizado, e a biblioteca de manipulação de imagem "ImageIO", própria do ambiente Java não possui codificadores para tal.

A biblioteca ImageIO pode ter codificadores adicionados por meio de plug-ins chama-

dos SPI's. A biblioteca JMRTDpossui consigo os SPI's para leitura e escrita de imagens no formato JPEG-2000. Porém estas SPI's contidas junto da biblioteca JMRTDnecessitam de outra biblioteca, chamada JJ2000(JACKSON: 2014) com o código fonte de diversos codificadores especialmente para o formato JPEG-2000. Após a compilação dos codificadores, estes foram adicionados ao pacote Java do aplicativo, e com uma pequena modificação no codificador de leitura para que a biblioteca reconhecesse que ele podia ler o formato, a leitura das imagens dos passaportes oficiais, e escrita das imagens em formato jpeg2000 tornou-se possível, assim, após a redução de tamanho, a imagem é codificada diretamente para formato jpeg2000.

O mesmo processo de formatação também foi feito com as digitais. A imagem resultante da coleta é escalada em 50% nos dois eixos cartesianos, e codificada em jpeg, com 50% da qualidade antes de ser armazenada em memória, e enviada ao cartão.

```
MatOfByte mob = new MatOfByte();
MatOfInt params = new MatOfInt(
    Imgcodecs.CV_IMWRITE_JPEG_QUALITY, 100);

if (cv.read(frame)) {

    if (frame.empty()) {
        System.out.print("Failed to capture Image");
        return;
    }

    Imgcodecs.imencode(".jp2", frame, mob, params);
    byte[] ba = mob.toArray();
    try {

        File picture = new File("PictureTakenFull.jp2");
        FileOutputStream fos = new FileOutputStream(picture);
        fos.write(ba);
        fos.close();

        ImageWorks.extractPointsFromImageAndResolve(picture);

        Imgproc.resize(frame, frame, new Size(320, 240));

        params = new MatOfInt(Imgcodecs.CV_IMWRITE_JPEG_QUALITY, 50);

        Imgcodecs.imencode(".jp2", frame, mob, params);

        ba = mob.toArray();
        picture = new File("PictureTakenHalf.jp2");
        fos = new FileOutputStream(picture);
        fos.write(ba);
        back.placeImage(picture,ba);
        fos.close();

    } catch (Exception ex) {
        ex.printStackTrace();
```

```
    }
}

cv.release();
```

Excerto de código 9: Código da conversão de uma foto

Com estes parametros, e utilizando uma webcam "LG-Webpro2" com resolução de 640x420 pixels, e o aparelho de coleta de digitais integrado a um notebook "Dell Vostro 4600" com resolução média de 160x500 pixels[1], sem se modificar as imagens, o tamanho da foto facial original, em média era de 20Kb, e de cada digital, 13Kb. Com 10 digitais, e a foto do portador, o espaço necessário para o armazenamento no cartão é de 150Kb, desconsiderando o resto das informações necessárias para as outras funções, o que é inviável, já que o modelo com mais memória interna de javacard disponível no mercado, NXP J3A081 80K, possui 80Kb de memória. Com a manipulação das imagens, o tamanho da foto caiu para 5.5Kb, e cada digital em média caiu para 4.5kb, deixando assim um tamanho total de imagens armazenadas de aproximadamente 51Kb, deixando ainda 29Kb de espaço para todas as outras informações necessárias[2]

4.6 Implementação dos protocolos de segurança

Para a implementação do protocolo de segurança de Autenticação Ativa, a classe de personalização de cartões da versão 4.9 do JMRTDjá possui um método que envia as informações necessárias para o cartão, porém se estava encontrando dificuldades para enviar a chave privada do cartão, utilizada neste protocolo. Depois de analisar tanto o método da biblioteca quanto as funções efetuadas pelo cartão ao receber tal instrução, percebeu-se uma discrepância entre os dados enviados e os dados lidos.

A comunicação entre o cartão e o terminal se dá por meio das APDUs, mas os dados que são enviados devem obedecer as *Basic Encoding Rules*, BER, que complementa o protocolo *Type-Length-Value*, ou TLV, descritas no anexo-D da parte 4 do documento ISO7816 (*Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange* 2008). Estas regras caracterizam o formato de codificação para os dados enviados dentro de uma APDU, e ditam que um objeto BERTLV, deve possui três campos: um ou mais bytes que descrevam uma TAG, esta que descreve que tipo de objeto se está enviando; um até três bytes que descrevam o comprimento em bytes da informação, e a informação em si.

Desta forma, a biblioteca possui um objeto chamado BERTLVWriter, que possui funções para o preenchimento de um buffer que segue o protocolo BERTLV, como *writeTag*, *writeLength* e *writeValue*. Porém, a função utilizava duas vezes *writeTag*, com a primeira tag identificando que os dados eram uma parte da chave, e a segunda que os dados são uma *String ASN1*, depois *writeLength* com o tamanho, para por fim *writeValue* com a informação. Já o cartão, que possui a classe espelho *BERTLVReader* fazia uma série de leituras a mais, lendo um identificador com *readTag*, depois um valor de tamanho com *readLength*, pulando estes bytes lidos com *skipValue*, lendo um novo identificador *readTag*, e após lendo o conteúdo restante dos dados *readValue*, o que fazia o cartão ler

[1]Sendo este um leitor de corrida, ou seja, para a captura é necessário correr o dedo pela faixa leitora, a resolução no eixo y se torna variável, de acordo com o quanto do dedo se correu pela leitora. Diversas passagens em diferentes dedos obtiveram assim uma média de 500 pixels de resolução no eixo y

[2]Foram-se escolhidos todos os valores para manipulação de imagem de forma a manter a possibilidade de análise a olho nu da foto da pessoa, e manter a verificabilidade de cada digital.

o primeiro identificador, tomar o segundo como tamanho de dados a serem pulados, pular um número de bytes, ler um novo identificador que agora faz parte da informação, e tomar um valor errôneo como tamanho da informação, o que causava uma exceção pois quase sempre o tamanho da informação restante era menor que o tamanho lido. Como no protocolo a segunda tag não existe, e o cartão não as utiliza, foram retiradas as primeiras três ações, ler tag, ler tamanho e pular, além de remover o envio da segunda tag da aplicação, resultou no funcionamento correto do envio da chave, e na correta configuração do protocolo.

A biblioteca JMRTD não possuía um método para a Autenticação Passiva, então ela teve de ser implementada. Durante a implementação, o maior problema enfrentado se deu em obter os certificados das CAs (Autoridades Certificadoras) necessários para que o protocolo fosse bem sucedido. A biblioteca Ldaptive foi utilizada para a leitura dos arquivos obtidos do repositório da ICAO, porém nem todos os certificados se encontravam lá. Com a biblioteca *LDaptive*, se pode ler um arquivo e obter cada entrada ou certificado que ele contém, depois é necessário decodificar em Base64 a informação para então criar um certificado.

```
FileReader file = new FileReader("certs/icaoPKD.ldif");
LdifReader reader = new LdifReader(file);
SearchResult result = reader.read();
Collection<LdapEntry> entries = result.getEntries();
for(LdapEntry entry : entries){

    //cf do tipo java.security.cert.CertificateFactory

    if(entry.getAttribute("userCertificate;binary") != null){
        X509Certificate c = (X509Certificate)
            cf.generateCertificate(new ByteArrayInputStream
                (Base64.getDecoder().decode(
                    entry.getAttribute("userCertificate;binary")
                        .getStringValue())));

        //lista de todos os certificados
        certChain.add(c);

    }
}
```

Excerto de código 10: Código da leitura de uma lista de certificados da ICAO

A verificação dos certificados consiste em gerar uma corrente entre o certificado pertencente ao passaporte, e um certificado pertencente a uma CA. Para isso, se compara o certificado alvo com todos os certificados vindos dos arquivos do repositório ICAO, e, se um deste assinou o certificado alvo, adiciona-se a corrente, este se torna o certificado alvo, e o processo se repete até se encontrar um CA.

```
public static ArrayList<X509Certificate>
    simpleCertificateChainBuilder
    (X509Certificate cert, Set<X509Certificate> certs) {

        ArrayList<X509Certificate> chain = new ArrayList();
```

```
X509Certificate current = cert;
chain.add(cert);

boolean ok = true;
while (ok) {
    ok = false;

    //Constroi a corrente de baixo para cima

    for (X509Certificate c : certs) {
        try{

            if(cert.equals(c)){
                chain.add(c);
                break;
            }

            current.verify(c.getPublicKey(), "BC");
            chain.add(c);
            current = c;
            certs.remove(c);
            ok = true;
            break;
        } catch (Exception e){
            /Certificado current nao assinou c
        }

    }


}
//Procura o CA na corrente criada
if(chain.size() > 1){
    boolean foundAnchor = false;
    for(X509Certificate c : chain){
        try{
            if(isSelfSigned(c)){
                foundAnchor = true;
                break;
            }
        }catch(Exception e ){
            e.printStackTrace();
        }
    }
    if(foundAnchor){
        return chain;
    }
}

return null;
```

```
}
```

Excerto de código 11: Código da verificação da corrente de Certificados

Caso a corrente se forme, então se tem certeza de que o certificado do passaporte veio de uma fonte confiável, mas ainda se deve verificar se ele não foi revogado, ou sua validade expirou. Foi pesquisado um excerto de código[3] para realizar a verificação da validade e certificação das CRL's de um certificado. Uma CRL é uma lista de revogação de certificados, que é obtida através de repositórios online amplamente conhecidos e confiados (*Certificate Revocation List (CRL)*). Este excerto confere as listas que são providas no certificado e retorna o resultado caso lá ele esteja revogado.

A segunda parte deste protocolo é conferir se os arquivos internos não foram modificados. Para isso, o arquivo SOD possui uma lista com a Hash de cada arquivo que foi colocado no cartão, incluindo dele mesmo. Estas Hash's são inseridas quando o cartão é criado, sempre sendo inseridas conforme os arquivos também vão sendo, e na hora da verificação, caso algum arquivo tenha sido modificado, a hash do arquivo lido não vai bater com a hash armazenada no SOD.

```
//COM possui uma lista de presenca contendo as TAGS dos arquivos
//tags eh um array com todas as possiveis tags
for (int i = 1; i < tags.length; i++) {
        MessageDigest SHA = MessageDigest.
         getInstance(digestAlgorithm);

    if (Arrays.contains(comtags, tags[i])) {
         System.out.println(tags[i]);
         DataGroup dg = reader.getDGFile(i);
         if(dg == null){
             continue;
         }
         byte[] file = dg.getEncoded();

         currentHash = SHA.digest(file);

         if (java.util.Arrays.
             equals(currentHash, sodHashes.get(i))) {
             result.put(i, Boolean.TRUE);
         } else {
             result.put(i, Boolean.FALSE);
             SODValidity = false;
         }
    }
}
```

Excerto de código 12: Código da verificação das Hash's armazenadas no SOD

Durante o desenvolvimento, não foi possível obter acesso a um cartão com suporte a criptografia de curvas elípticas para a implementação do protocolo EAC, além disso, infe-

[3]http://www.nakov.com/blog/2009/12/01/x509-certificate-validation-in-java-build-and-verify-chain-and-verify-clr-with-bouncy-castle

lizmente como para a execução do protocolo são necessárias chaves privadas pertencentes ao país, por isso não foi possível a execução deste protocolo em um passaporte oficial.

5 Funcionamento

5.1 Instalação

O aplicativo é distribuído através de um arquivo zip contendo um JAR, uma pasta "lib", e uma pasta "dist". Este arquivo zip deve ser descompactado em uma pasta de preferência criada apenas para a aplicação. Lá se deve executar o JAR com o comando do terminal: java -jar ¡nome do arquivo¿.jar e o programa executará.

5.2 Desinstalação

Para efetuar a desinstalação, basta excluir a pasta em que foi baixada a aplicação.

5.3 Descrição da aplicação

O aplicativo desenvolvido possui duas funções, coletar as informações do usuário e com ela gerar um novo passaporte válido, e verificar se as informações coletadas estão devidamente inseridas no cartão. A parte da verificação de identidade do proprietário não faz jús ao escopo deste trabalho. O aplicativo possui as funções de retirar uma foto da pessoa, e de coletar todas as suas digitais. Ele também necessita de certificados digitais e pares de chaves para que os protocolos de segurança possam ser executados, que devem ser armazenados em uma keystore java.

Ao iniciar o aplicativo para emitir um passaporte, deve-se preencher os campos referentes as informações do MRZ, como nome, número do documento, nascionalidade, validade do documento, como mostrado na figura 5.

Figura 4: Tela Inicial.

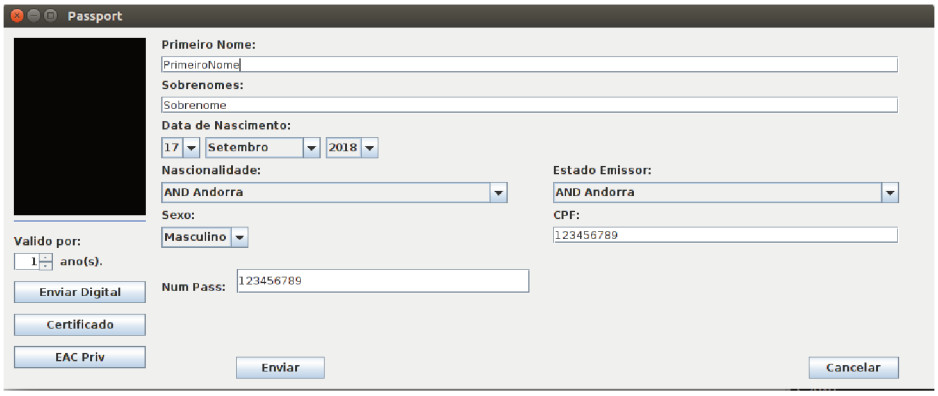
Fonte: Própria

A foto é enviada clicando-se duas vezes no espaço dedicado a ela, o que fará abrir uma janela para captura da foto. Isto ligará a primeira câmera encontrada na máquina, e apertando-se o botão, irá retirar uma foto, que aparecerá no espaço da janela principal, para ser enviada ao cartão.

Clicando no botão de digitais, a janela para coleta de digitais irá aparecer como mostrado na figura 6, mostrando cinco (5) espaços vazios. Clicando-se duas vezes no espaço referente a imagem de cada dedo, uma caixa de diálogo mostrará que a leitora está pronta para coleta da digital do respectivo dedo[4]. Primeiramente os dedos da mão

[4]Como explicado na seção de desenvolvimento, existe uma falta de comunicação entre a biblioteca em C rodando em JNI, e as Threads Java, portanto o efetivo tempo de preparo da leitora é maior do que o tempo levado para a caixa de diálogo aparecer

Figura 5: Tela de criação de um passaporte.



Fonte: Própria

Figura 6: Tela para coleta de digitais.

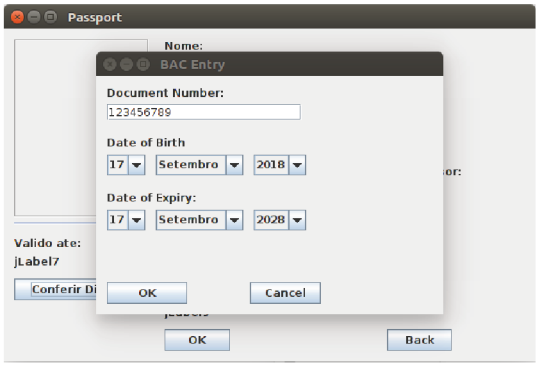
Fonte: Própria

direita estarão disponíveis, e ao se pressionar o botão para trocar a mão, os cinco dedos da mão esquerda se mostrarão disponíveis. Ao se escanear uma digital, a sua imagem irá aparecer no espaço indicado, e a caixa de diálogo se fechará automáticamente. Fazendo isso com todas as digitais, deve-se clicar no botão salvar, para que elas sejam efetivamente guardadas para serem enviadas ao cartão.

Todos os protocolos de segurança são configurados automáticamente ao se enviar as informações para o cartão: O protocolo BAC será configurado com as informações do MRZ, o protocolo PA, com certificados guardados em uma keystore na máquina e com os arquivos do cartão que estão sendo enviados, o protocolo AA terá um par de chaves gerado aleatóriamente na hora do envio.

Para se conferir os dados do cartão, deve-se clicar no botão de verificar da tela inicial, e preencher as informações do BAC na tela. Ao se clicar em OK, se o cartão ainda não foi inserido, será pedido para que se o faça, após isso, o cartão será lido e as informações mostradas na tela. Os protocolos serão executados enquanto o aplicativo lê o passaporte, e se algum falhar, uma mensagem será mostrada.

Figura 7: Tela para preenchimento das informações do protocolo BAC

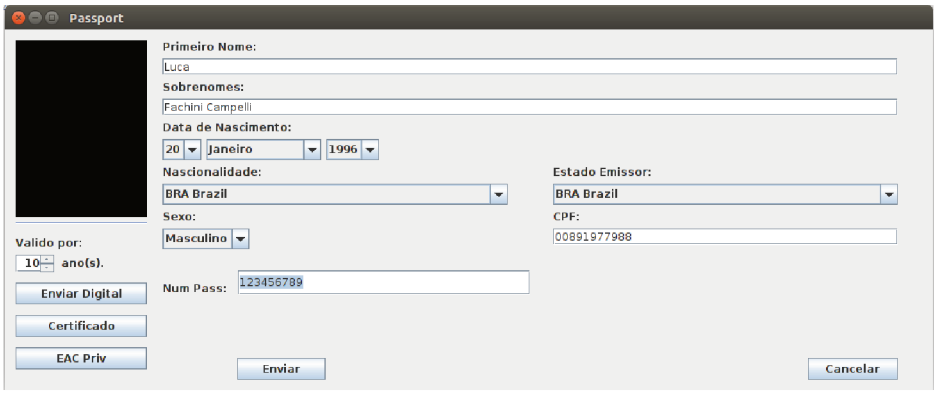


Fonte: Própria

6 Criando um passaporte

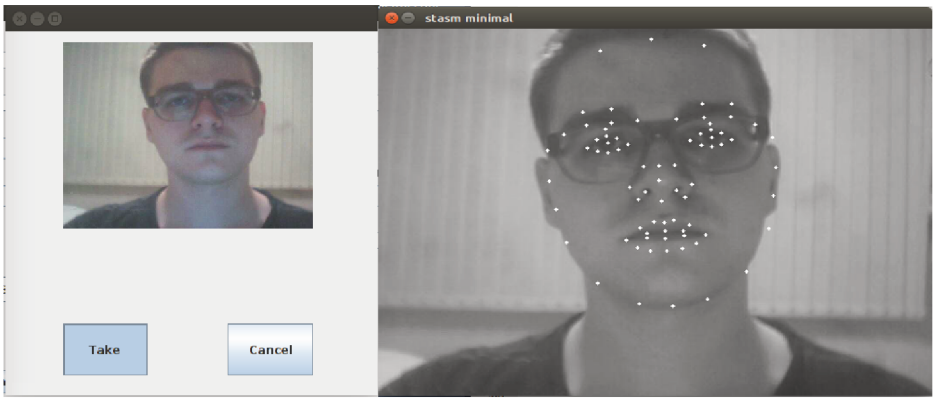
Inserido um cartão na leitora o aplicativo é executado, e a opção de criação do passaporte é selecionada. As informações do MRZ serão preenchidas conforme a figura 8. Clica-se duas vezes no lugar da foto na mesma tela e se tira uma foto. Após a foto, a biblioteca Stasm automaticamente mostra na tela a foto após a análise feita para procurar os pontos faciais, conforme a figura 9.

Figura 8: Tela de preenchimento com as informações do MRZ devidamente preenchidas



Fonte: Própria

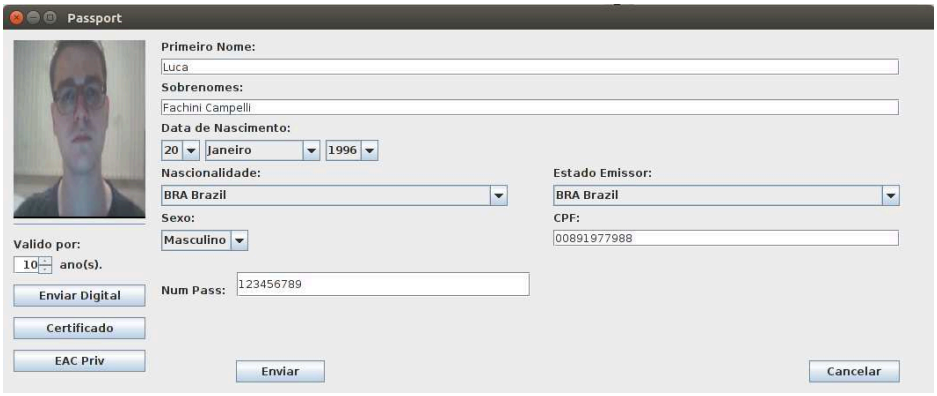
Figura 9: Tela de captura da foto e subsequente análise de pontos pela biblioteca Stasm



Fonte: Própria

Depois de tirada a foto, a tela de preenchimento estará igual a figura 10, o que para os fins deste trabalho é o necessário. Apertando no botão *Enviar* o aplicativo começará o processo de preencher o cartão. Não foi feita uma interface gráfica de usuário que mostre o progresso, durante o envio, a aplicação enviará mensagens para a os fluxos padrões de saída, que, no nosso caso como estamos rodando a aplicação dentro da IDE NetBeans, serão impressos na tela Output. Estes logs de envio estarão transcritos no Anexo B ao final deste trabalho. Obtendo-se a mensagem de sucesso, pode-se verificar o passaporte recém criado utilizando a função de verificação do aplicativo como mostra a figura 11.

Figura 10: Tela de preenchimento totalmente preenchida



Fonte: Própria

Figura 11: Leitura pelo aplicativo do cartão preenchido

Fonte: Própria

Figura 12: Leitura pelo aplicativo de um passaporte oficial

Fonte: Própria

7 Testes

Os testes serão efetuados comparando as leituras de um cartão criado utilizando o programa desenvolvido com um passaporte oficial da República Federativa do Brasil, tendo como parâmetros de comparação, as informações do MRZ, a foto, e o resultado do protocolo PA, já que o protocolo EAC não poderá ser efetuado pois não se é possível obter as chaves necessárias, por serem propriedade secreta do governo do Brasil.

O reconhecimento facial não será utilizado, pois após a leitura do passaporte oficial, constatou-se que ele não contém informação dos Feature Points.

Para os testes, foi criado um módulo do programa para ler as informações dos passaportes, utilizando-se da biblioteca JMRTD. Este módulo é capaz de realizar os protocolos BAC, e PA, o que garante a leitura, e garante que os passaportes possuem certificados válidos, além de ler e resgatar informações do MRZ e a foto do portador. O módulo de verificação tal como o de criação possuem o protocolo AA, porém este é substituível pelo protocolo EAC, por ser mais seguro, e desta forma os passaportes oficiais não o utilizam, então ele não será utilizado como parâmetro para comparação.

As leituras serão efetuadas separadamente, por leitoras diferentes, já que os passaportes funcionam baseados em RFID, e os cartões por chips de contato, e os resultados dos protocolos serão impressos nos logs de saída da IDE Netbeans, de onde será executado o aplicativo. Por isso também as leituras não poderão ser comparadas em outros aplicativos existentes, já que os cartões JavaCard não possuem comunicação por RFID, utilizada para a leitura pelos aplicativos. A leitora de RFID é do modelo *ACS ACR122U PICC*, já a leitora de chip é do modelo *SCM Microsystems Inc. SDI010 Smart Card Reader*.

A partir daqui o passaporte oficial será denominado apenas como passaporte, e o cartão personalizado será denominado de cartão.

Começando pelas informações do MRZ, primeiramente é possível determinar que o protocolo BAC foi realizado com sucesso, já que as informações necessárias foram lidas do cartão e do passaporte como mostram as figuras 11 e 12. Um adendo ao cartão: o aplicativo implementa um campo extra no documento que nos passaportes oficiais não é utilizado, chamado de "Personal Number" na biblioteca JMRTD, e que ocupa um dos espaços de dados opcionais previstos no documento ICAO 9303.

Comparando as duas leituras, pode-se ver que os resultados são equivalentes, e que todos os dados foram preenchidos corretamente no cartão, incluindo a foto, que foi colocada e lida corretamente.

O segundo passo é conferir os logs das execuções dos protocolos, no caso apenas o PA, pois o BAC já foi feito, e este é o único que ambos os documentos possuem, já que não é possível efetuar o EAC no passaporte, e ele não contém o AA.

Os certificados utilizados para a aferição do passaporte oficial, foram obtidos do site do Ministério de Relações Exteriores[5], que é uma das autoridades certificadoras que emitem certificados para os passaportes brasileiros. Já os certificados utilizados para o cartão foram gerados no momento de sua personalização, e são certificados de teste, onde um certificado auto-assinado atua como raíz, e o outro, assinado pelo raíz é o certificado do cartão.

Transcritos abaixo estarão os excertos dos logs de cada leitura, para comparação. Cada log iniciado pela palavra PA, mostra os passos que o protocolo executa, primeiro, aferindo a validade do certificado presente no documento, e depois, aferindo se a hash de

[5]https://certificados.serpro.gov.br/acmre/certificate-chain

cada grupo de dados (DG) presente bate com a hash armazenada no arquivo SOD. No excerto, os números avulsos caracterizam o ID do grupo de dados, sendo 97 o DG1, 117 o DG2, 99 o DG3,110 o DG14 e 111 o DG15. Os certificados encontrados terão duas linha para identifica-los, a primeira será a identificação de quem o emitiu, o Issuer, e a segunda, a quem ele se refere, o Subject.

Excerto do log do cartão:

```
        PA
Aferindo validade do certificado
Encontrou certificado :
I: CN=root.labsec, O=LABSEC/UFSC, L=FLORIANOPOLIS, C=DE
S: CN=root.labsec, O=LABSEC/UFSC, L=FLORIANOPOLIS, C=DE
Certificado Válido:
Corrente
Certificado AC Raíz:
I: CN=root.labsec, O=LABSEC/UFSC, L=FLORIANOPOLIS, C=DE
S: CN=root.labsec, O=LABSEC/UFSC, L=FLORIANOPOLIS, C=DE
Certificado Alvo:
I: CN=root.labsec, O=LABSEC/UFSC, L=FLORIANOPOLIS, C=DE
S: CN=card.labsec, O=LABSEC/UFSC, L=FLORIANOPOLIS, C=DE
97
117
111
SOD: Valido
COM ok
DG1 ok
DG2 ok
DG15 ok

\ac{sodfile}C=DE, L=FLORIANOPOLIS, O=LABSEC/UFSC, CN=card.labsec
```

Excerto log do passaporte:

```
        PA
Aferindo validade do certificado
Encontrou certificado :
I: CN=Autoridade Certificadora Raiz Brasileira v4,
    OU=Instituto Nacional de Tecnologia da Informacao - ITI,
    O=ICP-Brasil, C=BR
S: CN=Autoridade Certificadora Ministerio das Relacoes Exteriores,
    OU=Autoridade Certificadora Raiz Brasileira v4,
    O=ICP-Brasil, C=BR
Encontrou certificado :
I: CN=Autoridade Certificadora Raiz Brasileira v4,
    OU=Instituto Nacional de S: Tecnologia da Informacao - ITI,
    O=ICP-Brasil, C=BR
S: CN=Autoridade Certificadora Raiz Brasileira v4,
    OU=Instituto Nacional de Tecnologia da Informacao - ITI,
```

```
    O=ICP-Brasil, C=BR
Válido :

Corrente
Certificado AC Raíz:
I: CN=Autoridade Certificadora Raiz Brasileira v4,
    OU=Instituto Nacional de Tecnologia da Informacao - ITI,
    O=ICP-Brasil, C=BR
S: CN=Autoridade Certificadora Raiz Brasileira v4,
    OU=Instituto Nacional de Tecnologia da Informacao - ITI,
    O=ICP-Brasil, C=BR
Certificado:
I: CN=Autoridade Certificadora Raiz Brasileira v4,
    OU=Instituto Nacional de Tecnologia da Informacao - ITI,
    O=ICP-Brasil, C=BR
S: CN=Autoridade Certificadora Ministerio das Relacoes Exteriores,
    OU=Autoridade Certificadora Raiz Brasileira v4,
    O=ICP-Brasil, C=BR
Certificado Alvo:
I: CN=Autoridade Certificadora Ministerio das Relacoes Exteriores,
    OU=Autoridade Certificadora Raiz Brasileira v4,
     O=ICP-Brasil,C=BR,
S: CN=ePass DSCA v2,
    OU=Autoridade Certificadora do Ministerio das Relacoes Exteriores,
    O=ICP-Brasil, C=BR

97
117
99
110
SOD: Valido
DG1 ok
DG2 ok
DG14 ok

\ac{sodfile}C=BR, O=ICP-Brasil,
    OU=Autoridade Certificadora Raiz Brasileira v4,
    CN=Autoridade Certificadora Ministerio das Relacoes Exteriores
```

8 Conclusões

Para o desenvolvimento da aplicação, e execução deste trabalho, foram necessárias a leitura dos padrões ICAO 9303 para o conhecimento acerca da estrutura do passaporte, ISO/IEC 19794 para um conhecimento aprofundado sobre os dados biométricos e o documento BSI-TR 03110 para um conhecimento aprofundado dos protocolos de segurança. Além da ISO 7816, que estabelece o padrão para a comunicação com o cartão.

Ao longo do desenvolvimento, para enfrentar as dificuldades encontradas e se adequar ao padrão ICAO, todas as bibliotecas presentes foram sendo gradualmente integradas ao projeto.

Tendo em vista que não foi possível executar os protocolos constituíntes do EAC, por não se possuírem as chaves utilizadas, e o protocolo AA, por não ser mais utilizado em passaportes, em prol do protocolo EAC, a comparação teve de ser efetuada pela execução dos protocolos restantes, BAC e PA, e pelas leituras efetuadas dos dois documentos.

Assim, como demonstrado no capítulo anterior, nas imagens 11 e 12, pode-se constatar que as leituras dos arquivos DG 1, e DG 2 coincidem, exceto por dados que foram implementados na aplicação, como o CPF do usuário e os Feature Points para reconhecimento facial, que não existem no passaporte oficial, mas estão declarados no documento ICAO 9303 e no padrão ISO/IEC 19794. Além disso, por ser possível ler o passaporte oficial, pode-se afirmar que o protocolo BAC foi efetuado com sucesso pela aplicação.

Pela leitura dos logs apresentados no capítulo anterior, se vê que nos dois documentos, a corrente de certificação foi criada com sucesso, que os certificados são todos válidos, e que todas as hash's dos arquivos internos coincidem com as armazenadas no arquivo SOD, caracterizando o sucesso da execução do protocolo PA.

Sendo assim, pelas métricas estabelecidas no início deste documento, pode-se concluir que é possível a criação de um passaporte eletrônico na plataforma Javacard que seja equivalente a um passaporte oficial, embora não possua todas as suas funcionalidades.

9 Referências

Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – (2015). Standard. Bonn, Alemanha: Escritório Federal de Segurança em Tecnologia da Informação.

Bouncy Castle, The Legion of the. *Bouncy Castle*. URL: `http://www.bouncycastle.org/java.html` (visited on 04/13/2018).

BOZHANOV, Bozhidar (2016). *epassport Reader*. URL: `https://github.com/Glamdring/epassport-reader` (visited on 08/15/2018).

Brilliant.org (2018). *Secure Hashing Algorithms*. Brilliant.org. URL: `https://brilliant.org/wiki/secure-hashing-algorithms/#references` (visited on 11/23/2018).

Carlos Henrique Ramos Lima (2015). *Processo de Emissão de Passaporte*. Polícia Federal - Ministério da Segurança Pública. URL: `http://www.pf.gov.br/servicos-pf/passaporte/fluxo-de-funcionamento-do-servico-de-passaporte` (visited on 10/03/2018).

CHEN, Zhiqun (2000). *Java Card™ Technology for Smart Cards : architecture and programmer's guide*. England, UK: Addison-Wesley.

DocuSign. *How Digital Signatures Work*. URL: `https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq` (visited on 04/13/2018).

HOUZARD, Jean-Francois and Olivier ROGER (2009). *PyPassport, Python library to read the biometric ePassport*. URL: `https://code.google.com/archive/p/pypassport/` (visited on 08/10/2018).

— (2013). *E-Passport Viewer*. URL: `https://github.com/andrew867/epassportviewer/tree/master/pypassport-2.0/pypassport` (visited on 08/10/2018).

Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange (2008). Standard. England,UK: International Organization of Standardization and International Electrotechnical Comission.

IETF. *RFC 2849*. URL: `https://tools.ietf.org/html/rfc2849` (visited on 10/24/2018).

Information technology — Biometric data interchange formats — Part 5: Face image data (2008). Standard. England,UK: International Organization of Standardization and International Electrotechnical Comission.

Informação, Instituto Nacional da Tecnologia da. *Autoridades Certificadoras*. URL: `https://www.iti.gov.br/icp-brasil` (visited on 04/13/2018).

InnoValor (2013). *ReadID*. URL: https://www.readid.com/ (visited on 06/26/2018).

International Civil Aviation Organization (2015). *ICAO 9303: Machine Readable Travel Documents*. ICAO. URL: https://www.icao.int/publications/pages/publication.aspx?docnum=9303 (visited on 04/13/2018).

JACKSON, Andy (2014). *JJ2000*. URL: https://github.com/anjackson/jj2000 (visited on 11/08/2018).

LibFPrint Team. *LibFPrint*. URL: https://www.freedesktop.org/wiki/Software/fprint/libfprint/ (visited on 04/13/2018).

middleware, vt. *LDAPtive*. Virginia Tech Middleware Services. URL: http://www.ldaptive.org/ (visited on 10/24/2018).

Milborrow, S. and F.Nicolls (2014). "Active Shape Models with SIFT Descriptors and MARS". In: *VISAPP*. URL: http://www.milbo.users.sonic.net/stasm/ (visited on 04/13/2018).

OOSTDIJK, Martin (2010). *JMRTD*. URL: https://www.jmrtd.org (visited on 08/15/2018).

PrimeKey. *EJBCA Open Source PKI Certificate Authority*. URL: https://www.ejbca.org (visited on 04/13/2018).

R.K. Roberto (2015). *Passaporte Eletrônico*. Polícia Federal - Ministério da Segurança Pública. URL: http://www.pf.gov.br/servicos-pf/passaporte/passaporte-eletronico (visited on 10/03/2018).

TANANAEV, Anton (2016). *e-Passport NFC Reader*. URL: https://github.com/tananaev/passport-reader (visited on 08/15/2018).

Team, OpenCV. *OpenCV Library*. URL: https://opencv.org/ (visited on 04/13/2018).

Team, SCUBA. *Smart Card Utilities for Better Access, Project SCUBA*. URL: http://scuba.sourceforge.net (visited on 04/13/2018).

TechTarget. *Certificate Revocation List (CRL)*. URL: https://searchsecurity.techtarget.com/definition/Certificate-Revocation-List (visited on 04/13/2018).

Anexos

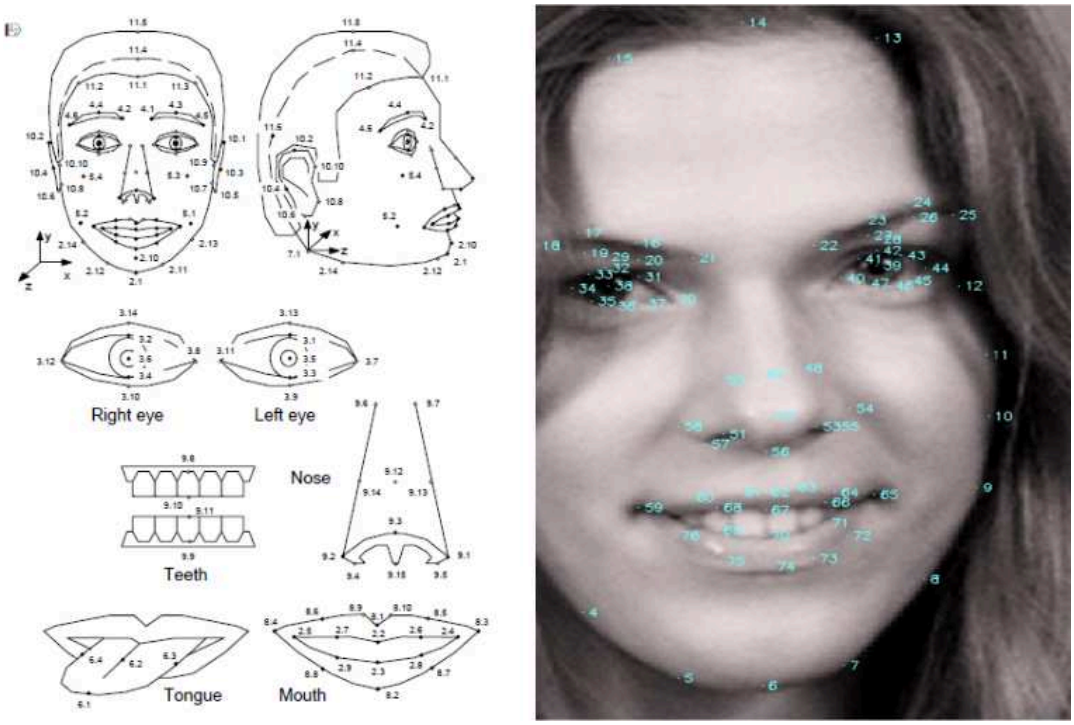
A Mapeamento entre Stasm e ISO/IEC 19794-5

Utilizando a biblioteca STASM para analizar as imagens de fotos e extrair os pontos de características faciais, foi notado que a biblioteca STASM mapeia os pontos diferentemente do que o padrão ISO/IEC 19794-5, incluindo pontos que não existem no STASM. Desta forma foi necessário mapear os pontos manualmente de um para outro, obtendo-se a tabela abaixo.

A nomenclatura do padrão ISO/IEC 19794-5, se dá por um número referente a área da face seguido de um ponto e um número para identificar a característica.

A nomenclatura da biblioteca STASM se dá pela ordem de aferição dos pontos, iniciando pela linha do maxilar esquerdo (Costeleta Esquerda) e seguindo em forma de espiral no sentido anti-horário, como mostra a imagem .

Figura 13: Comparação entre pontos Stasm e ISO 19794-5



Fonte: (*Information technology — Biometric data interchange formats — Part 5: Face image data* 2008) e *https://stackoverflow.com/questions/19374286/implement-gaze-estimation-using-stasm-facial-co-ordinates?rq=1*

Descrição do ponto	Nomenclatura ISO19794-4	Nomenclatura STASM
Contorno do rosto:		
Ponta do queixo	2.1	6
Mandíbula Esquerda	7.0	3
Mandíbula Direita	7.1	9
Queixo Esquerda	2.11	7

Queixo Direita	2.12	5
Costeleta Esquerda	10.9	1
Costeleta Direita	10.10	11
Topo da Cabeça	11.4	N/A
Linha do Cabelo Meio	11.1	14
Linha do Cabelo Direita	11.2	15
Linha do Cabelo Esquerda	11.3	13
Topo do Cabelo	11.5	N/A
Orelhas		
Topo Orelha Esquerda	10.1	N/A
Topo Orelha Direita	10.2	N/A
Meio Orelha Esquerda	10.3	N/A
Meio Orelha Direita	10.4	N/A
Lóbulo Orelha Esquerda	10.5	N/A
Lóbulo Orelha Direita	10.6	N/A
Base da Orelha Esquerda	10.7	2
Base da Orelha Direita	10.8	10
Sobrancelhas:		
Canto Interno S Esquerda	4.1	22
Canto Interno S Direita	4.2	21
Borda Superior S Esquerda	4.3	24
Borda Superior S Direita	4.4	17
Canto Exterior S Esquerda	4.5	25
Canto Exterior S Direita	4.6	18
Olho Direito:		
Iris Borda Superior	3.2	N/A
Iris Borda Inferior	3.4	N/A
Pupila	3.6	38
Canto Interno	3.8	30
Pálpebra Inferior	3.10	36
Canto Externo	3.12	34
Pálpebra Superior	3.14	32
Centro do Olho	12.2	Interseção(30-34,32-36)
Olho Esquerdo:		
Iris Borda Superior	3.1	N/A
Iris Borda Inferior	3.3	N/A
Pupila	3.5	39
Canto Interno	3.11	40
Pálpebra Inferior	3.9	46
Canto Externo	3.7	44
Pálpebra Inferior	3.13	46
Centro do Olho	12.1	Interseção(40-44,42-46)
Nariz:		
Extremidade Narina Esq	9.1	54

Extremidade Narina Dir	9.2	58
Ponta do Nariz	9.3	52
Base Narina Direita	9.4	57
Base Narina Esquerda	9.5	55
Topo do Nariz Esquerda	9.6	N/A
Topo do Nariz Direita	9.7	N/A
Corpo Nasal Centro	9.12	49
Corpo Nasal Esquerda	9.13	48
Corpo Nasal Direita	9.14	50
Ponte Nasal	9.15	56
Narina Esquerda	12.3	53
Narina Direita	12.4	51
Boca:		
Contorno Labial:		
Superior Externo:		
Esquerdo	8.3	65
Esquerdo-Centro	8.5	64
Filtro Esquerdo	8.10	63
Central	8.1	62
Filtro Direito	8.9	61
Direito-Centro	8.6	60
Direito	8.4	59
Inferior Externo:		
Direito	8.8	75
Centro	8.2	74
Esquerdo	8.7	73
Perímetro interno labial:		
Canto Interno da Boca Esq	2.4	59
Canto Interno da Boca Dir	2.5	65
Lábio Superior Interno Cen	2.2	67
Lábio Superior Interno Esq	2.6	66
Lábio Superior Interno Dir	2.7	68
Lábio Inferior Interno Cen	2.3	70
Lábio Inferior Interno Esq	2.8	71
Lábio Inferior Interno Dir	2.9	69

Tabela 3: Tabela de conversão entre os pontos Stasm e o padrão ISO 19794-5

B Log do envio das informações ao cartão

Algumas explicações para a leitura deste log:

- O envio de cada arquivo é definido pela palavra *Enviando* mais o nome do arquivo.
- O envio de cada mensagem é definido pela palavra *Sending* mais o nome que define a mensagem.
- Abaixo da linha da mensagem estará sua resposta, dada pelo cartão. Como escrito no log, 9000 significa mensagem recebida e tratada com sucesso.
- Outras informações estarão comentadas com —–¿ indicando que foi adicionado durante a escrita deste documento

  Log:

```
---------->Zona MRZ como apareceria impressa no cartão
P<BRAFACHINI<CAMPELLI<<LUCA<<<<<<<<<<<<<<<<
1234567897BRA9601204M2810250008919779 88<<<20

---------->Leitoras/Impressoras encontradas pelo sistema
PC/SC terminal SCM Microsystems Inc. SDI010 Smart Card Reader
[Vendor Interface] (21120637203933) 00 00
PC/SC terminal SCM Microsystems Inc. SDI010 Smart Card Reader
[Vendor Interface] (21120637203933) 00 01

Sending BAC
Perso Sending BAC
9000: OK
P<BRAFACHINI<CAMPELLI<<LUCA<<<<<<<<<<<<<<<<
1234567897BRA9601204M2810250008919779 88<<<20

Sending Create File
9000: OK
Enviando arquivo DG1
Sending Write File
9000: OK

---------->Informações da imagem da foto impressas
---------->pela biblioteca JMRTD
out 25, 2018 7:10:39 PM org.jmrtd.lds.iso19794.FaceImageInfo
toMimeType
INFO: DEBUG: Image type: 1

Sending Create File
9000: OK
Enviando arquivo \ac{datagroup}2
Sending Write File
```

```
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
```

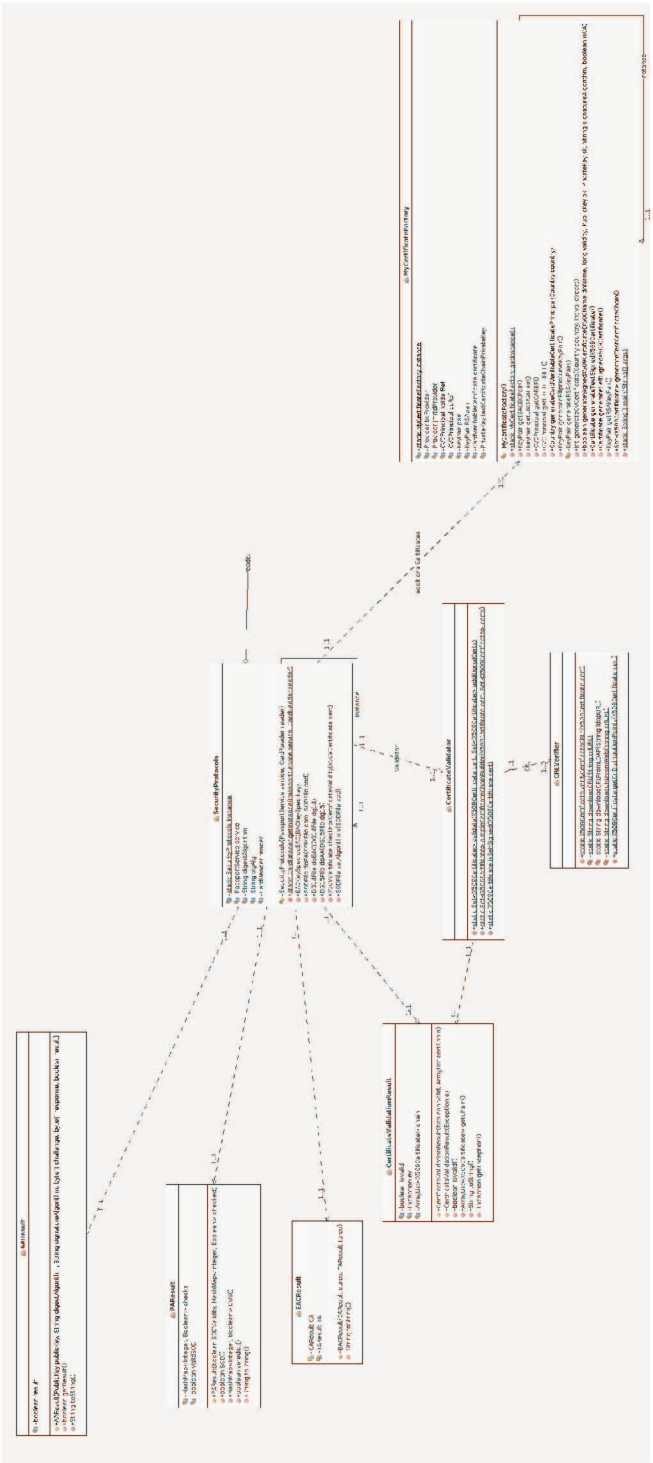
```
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Enviando informações para AA
9000: OK
9000: OK
Sending Create File
9000: OK
Enviando arquivo DG15
Sending Write File
9000: OK
Enviando arquivo COM
Sending Create File
9000: OK
Sending Write File
9000: OK
Certificado existe
Hashes.size 4
Sending Create File
9000: OK
Enviando arquivo SOD
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending Write File
9000: OK
Sending LOCK
```

```
9000: OK

-------->Final do envio, requisitado para remover o cartão.
Remova o cartão atual
```

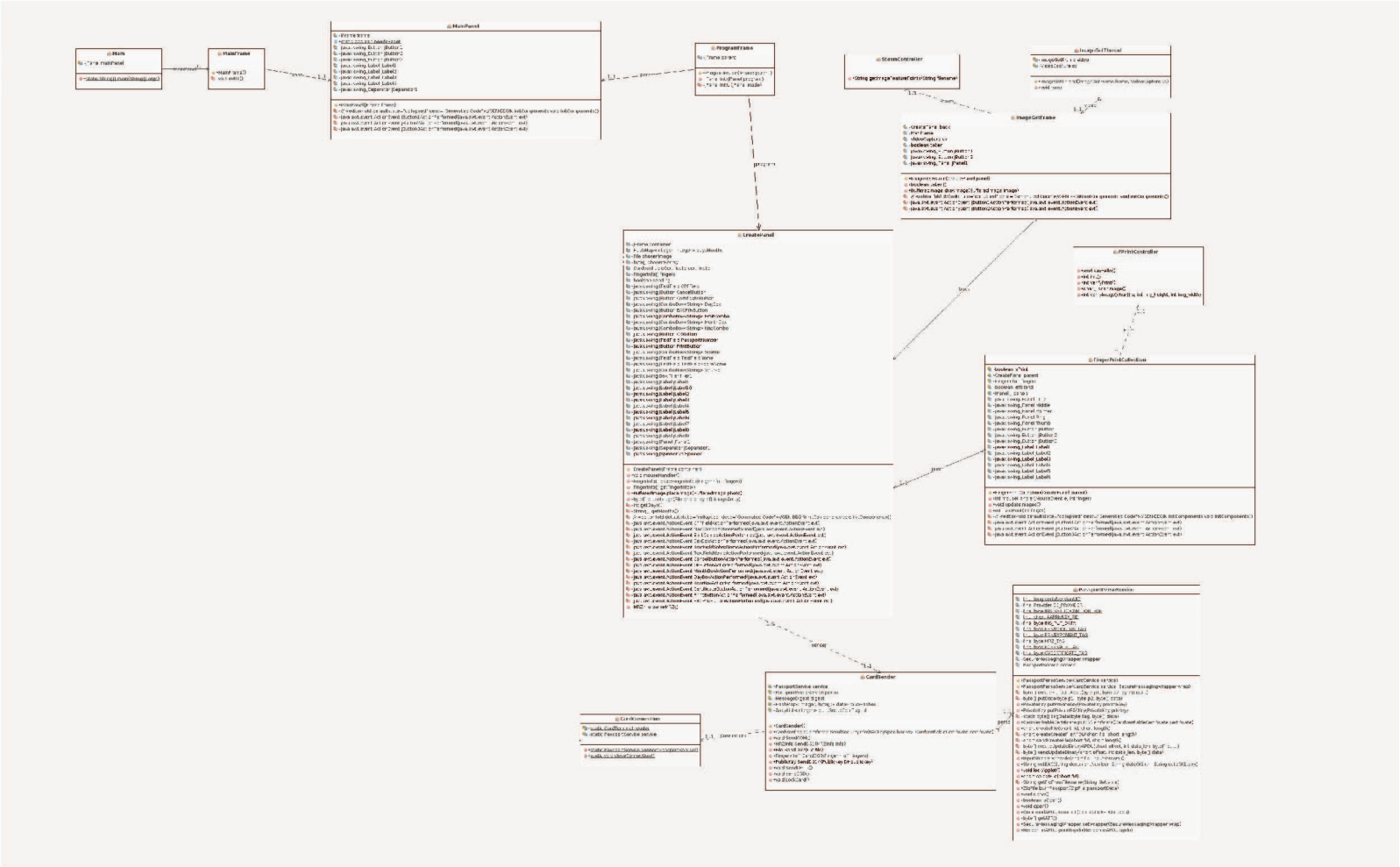
C Diagramas de classes do projeto

Figura 14: Diagrama de classes com foco nas classes de segurança



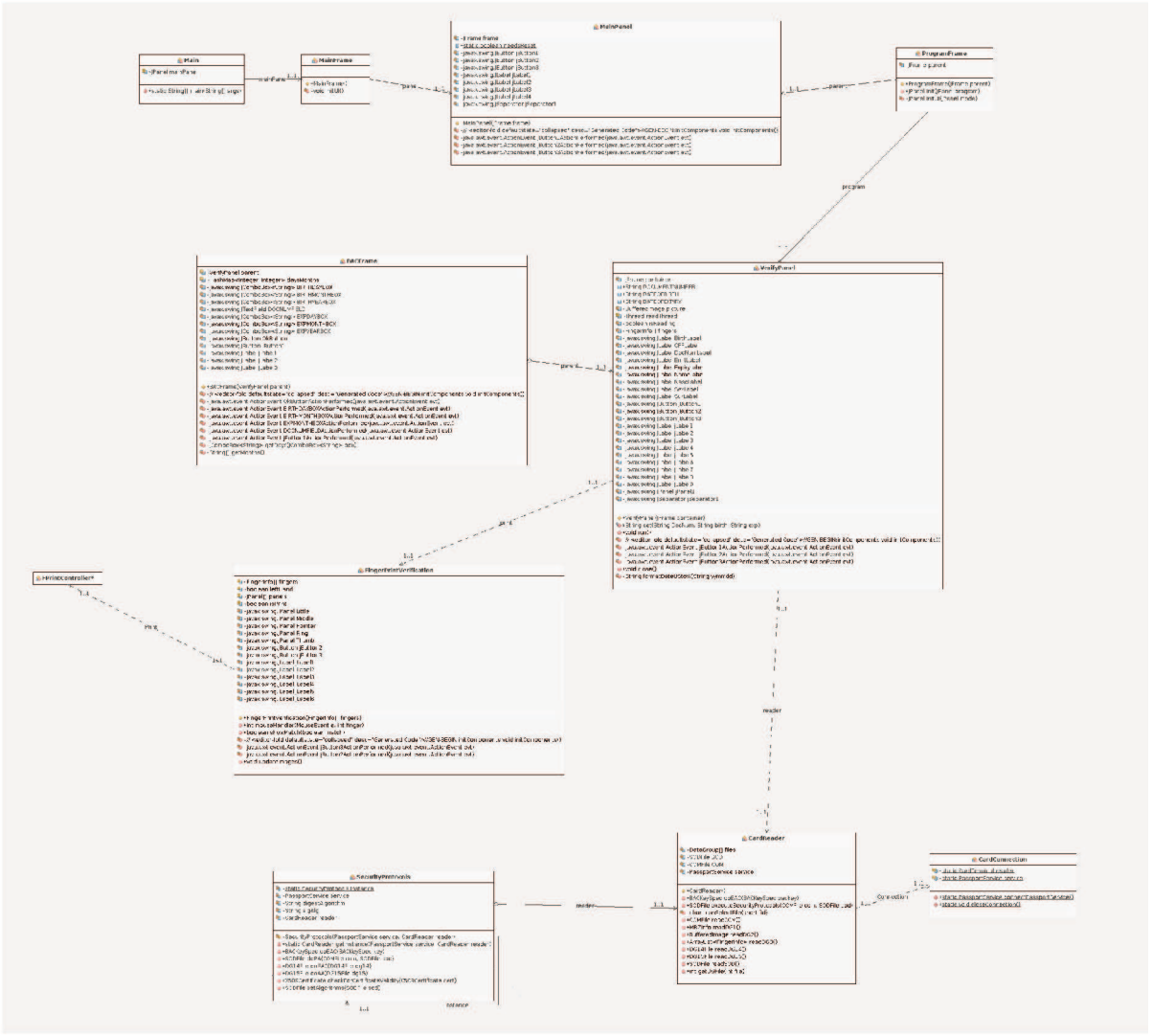
Fonte: Plugin EasyUML da IDE NetBeans

Figura 15: Diagrama de classes do módulo de escrita



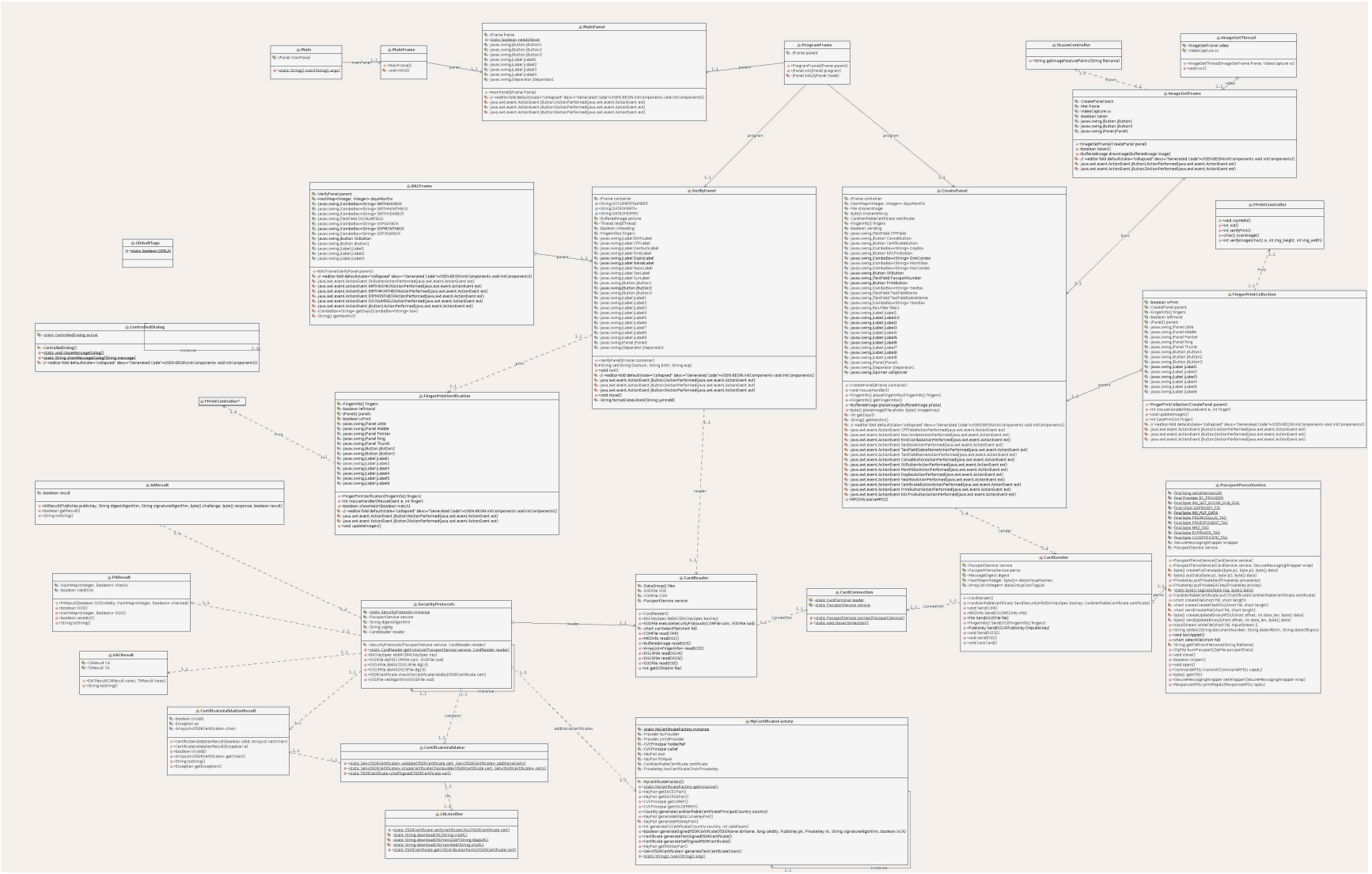
Fonte: Plugin EasyUML da IDE NetBeans

Figura 16: Diagrama de classes do módulo de leitura



Fonte: Plugin EasyUML da IDE NetBeans

Figura 17: Diagrama de classes completo



Fonte: Plugin EasyUML da IDE NetBeans