UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Felipe Menegola Blauth

Framework para Padronização do Acesso a Cartões Inteligentes

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Ricardo Felipe Custódio, Dr.
Orientador

Jeandré Monteiro Sutil, M.Sc.
Co-Orientador

Florianópolis, junho de 2011

Framework para Padronização do Acesso a Cartões Inteligentes

Felipe Menegola Blauth

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovada em sua forma final pelo Departamento de Informática e Estatística da Universidade Federal de Santa Catarina.

Prof. Vitório Bruno Mazzola, Dr.

Coordenador do Curso

Banca Examinadora

Prof. Ricardo Felipe Custódio, Dr.

Jeandré Monteiro Sutil, M.Sc.

Jean Everson Martina, Dr.

Túlio Cícero Salvaro de Souza, M.Sc.

André Bereza Júnior

À minha família, que sempre esteve ao meu lado,
independente das minhas escolhas.

Agradecimentos

Agradeço à minha mãe, minha irmã e minha tia Ivone, por terem formado o tripé de sustentação que permitiu a conclusão deste trabalho, bem como minha graduação. Sem elas nada seria possível.

Agradeço também ao professor Ricardo Felipe Custódio e ao LabSEC, cujos valiosos ensinamentos levarei para toda vida.

Um agradecimento especial ao Jeandré Monteiro Sutil, que jamais mediu esforços para me orientar e guiar desde o início do desenvolvimento deste trabalho.

Sumário

Lista de Figuras

Lista de Siglas

AC	Autoridade Certificadora
API	Application Programming Interface
APDU	Application Protocol Data Unit
ASI-HSM	Advanced Security Initiative - Hardware Security Module
ATR	Answer To Reset
CA	Certificate Autority
CCID	Chip/Smart Card Interface Device
DES	Data Encryption Standard
FIPS	Federal Information Processing Standards
HSM	Hardware Security Module
HTTPS	HyperText Transfer Protocol secure
IC	Integrated Circuit
ICC	Integrated Circuit Card
ICP	Infra-estrutura de Chaves Públicas
ICP-Brasil	Infra-estrutura de Chaves Públicas Brasileira
ICPEDU	Infra-estrutura de Chaves Públicas para Ensino e Pesquisa
IEC	International Electrotechnical Commission
IFD	Interface Device
ISO	International Organization for Standardization
LabSEC	Laboratório de Segurança em Computação - UFSC
MCT-7	Manual de Condutas Técnicas 7
NIST	National Institute of Standards and Technology
OpenSC	Open Source Smart Card Framework
PC/SC	Personal Computer/Smart Card
PIN	Personal Identification Number
PKCS	Cryptographic Token Interface Standard
PKI	Public Key Infrastructure
RNP	Rede Nacional de Ensino e Pesquisa
RSA	Rivest-Shamir-Adleman
SO	Security Officer
SSL	Secure Socket Layer
USB	Universal Serial Bus

Resumo

Tokens Criptográficos, como Cartões Inteligentes (do inglês Smart Cards), são plataformas computacionais intrinsecamente seguras, capazes de fornecer maior segurança e privacidade aos seus usuários. Esses dispositivos manipulam informações de autenticação, como certificados digitais, chaves criptográficas e senhas. Além disso, eles provêm, física e logicamente, maneiras seguras de proteger a informação sensível.

O projeto ASI-HSM [Lab 11], para o qual este trabalho foi desenvolvido, é um Módulo de Segurança Criptográfica (HSM) projetado para prover o gerenciamento seguro de chaves criptográficas, principalmente aquelas utilizadas por Autoridades Certificadoras (ACs). Esse módulo necessita de constante intervenção humana para que seja devidamente administrado, operado e auditado. A fim de garantir que apenas pessoas autorizadas possam manipulá-lo, sua gerência demanda forte autenticação, a qual é feita, principalmente, a partir de Smart Cards.

Atualmente, o ASI-HSM confia toda a interação com Smart Cards a um conjunto de ferramentas disponibilizadas pelo projeto OpenSC [Ope 11]. Essas funcionalidades possuem um grau aceitável de confiabilidade, mas não implementam algumas funcionalidades essenciais, tornam o tratamento de erros complicado e limitam o número de modelos de cartões que o ASI-HSM pode suportar.

Padronizar a maneira como o ASI-HSM interage com os Smart Cards, baseando-se na arquitetura de acesso mais aceita, além de elevar a segurança e a confiabilidade das operações são os alvos deste trabalho.

Abstract

Cryptographic tokens, such as Smart Cards, are intrinsically secure computing platforms, capable of providing enhanced security and privacy to their users. These devices are able to manipulate authentication information such as digital certificates, cryptographic keys and passwords. Moreover, they are capable of providing, physically and logically, secure ways to protect the sensitive information.

The ASI-HSM project [Lab 11], for which this work was developed, is a Hardware Security Module (HSM) designed to provide secure management of cryptographic keys, especially those used by Certificate Authorities (CAs). This system requires constant human intervention to be properly administered, operated and audited. To ensure that only authorized people can manipulate it, its management requires strong authentication, which is mainly made by using Smart Cards.

Currently, the ASI-HSM entrusts all the interaction with the Smart Cards to a set of tools provided by the OpenSC [Ope 11] project. These features have an acceptable degree of reliability, but do not implement some key features, make the error handling complicated and limit the number of card models that the ASI-HSM can support.

Standardize how the ASI-HSM interacts with Smart Cards, based on the most-accepted architecture, and increase safety and reliability of operations are the targets of this work.

Capítulo 1

Introdução

Smart Cards são utilizados constantemente em conjunto com o ASI-HSM, tornando possível a identificação e autenticação de seus usuários. Por esse motivo, é necessário existir um setor lógico [1] específico no equipamento, responsável por implementar o código que realiza essa gerência de maneira segura e eficaz.

No momento, esse setor encontra-se um tanto defasado e melhorias são necessárias. Principalmente no aspecto relativo à sua adaptação com a arquitetura e os padrões de acesso mais aceitos.

1.1 Contextualização

Recentemente, o ASI-HSM foi homologado pela ICP-Brasil com o nível máximo de segurança (nível três) [Imp 11]. Isso significa que esse módulo representa um contêiner seguro e confiável para chaves criptográficas. Além de proteções físicas e lógicas para o material sensível, o ASI-HSM também provê maneiras de realizar a gerência dessas chaves, sua auditoria e sua operação.

Diversas entidades estiveram presentes no desenvolvimento do ASI-HSM. [BER 09] explica: "Uma das únicas soluções nacionais para Módulos de Segurança

[1] Setor lógico, neste contexto, significa um conjunto de arquivos escritos em linguagem de programação com a finalidade de implementar funcionalidades para um contexto específico como, por exemplo, para trabalhar com Cartões Inteligentes.

Criptográfica foi desenvolvida em conjunto pela empresa Kryptus, fabricante de hardwares criptográficos, e pelo LabSEC que foi o responsável pela implementação do software desse HSM, conhecido como ASI-HSM. O ASI-HSM faz parte do projeto ICPEDU, coordenado e financiado pela Rede Nacional de Ensino e Pesquisa (RNP), que é uma empresa pública ligada ao Ministério de Ciência e Tecnologia e sempre está incentivando o avanço tecnológico no Brasil".

É fácil notar que um projeto desse porte deve oferecer o que há de melhor em cada aspecto de seu desenvolvimento. Todas as operações envolvendo autenticação de usuários do ASI-HSM utilizam Cartões Inteligentes. Portanto, um bom suporte a Smart Cards é necessário, constituindo o foco deste trabalho.

1.2 Objetivo

Este trabalho tem como objetivo tornar mais robusta a interação entre o ASI-HSM e os Smart Cards utilizados para autenticação de seus usuários.

1.3 Objetivos Específicos

O objetivo específico deste trabalho consiste em implementar uma solução utilizando modelos e interfaces já consolidados. Especificamente, o trabalho baseia-se na arquitetura de acesso PC/SC [PC/ 05a] e na interface PKCS #11 [RSA 11a]. Os objetivos que se destacam são:

- Tornar possível o reuso de cartões;
- Tornar o ASI-HSM compatível com uma gama possivelmente ilimitada de cartões;
- Tornar o ASI-HSM receptível a novas tecnologias;
- Remover todas os comandos *system* utilizados na interação com os cartões;
- Disponibilizar códigos de erros precisos;

- Elevar os níveis de segurança.

1.4 Justificativa

O ASI-HSM, atualmente, encontra-se amarrado às ferramentas disponibilizadas pelo OpenSC [Ope 11]. Essas ferramentas são, em sua essência, utilitários de interface texto, os quais ele faz uso através de chamadas *system* [htt 11].

O problema com essa abordagem encontra-se no fato de se ficar preso aos serviços disponibilizados por esses utilitários, tornando muito difícil adicionar funcionalidades ausentes e criando uma relação de total dependência.

Desfazer essas amarras e padronizar o acesso tornará o ASI-HSM muito mais flexível e receptivo a novas tecnologias. Além disso, devido a essa flexibilidade, determinadas melhorias na segurança tornam-se possíveis.

1.5 Metodologia

Para a realização do trabalho será estudada a arquitetura de acesso a Smart Cards, descrita no padrão PC/SC [PC/ 05a], a partir de computadores pessoais. Após pleno entendimento, definir-se-á em quais camadas da arquitetura serão feitas implementações. Em seguida, um protótipo será implementado para simular o comportamento e analisar sua viabilidade. Por fim, será feita a implantação no ASI-HSM, gerando uma nova versão.

1.6 Limitações do Trabalho

As limitações deste trabalho envolvem a extensão das funcionalidades às interfaces de gerência. Apesar de a maior parte das implementações não necessitarem quaisquer modificações, algumas de fato precisam. Essa restrição impossibilita o máximo proveito de alguns novos mecanismos.

Outra limitação é a quantidade de modelos diferentes de cartões testados. Foram testados três diferentes modelos e, embora o trabalho baseie-se fortemente em padrões já consolidados, seria interessante testar um número maior de modelos.

Por fim, a principal limitação deste trabalho encontra-se na não avaliação das decisões tomadas de acordo com o Manual de Condutas Técnicas 7 (MCT-7) [Ins 11]. Este trabalho procurou encontrar uma solução apropriada e segura para os problemas vigentes, desconsiderando o *MCT-7*.

Capítulo 2

Segurança e Dispositivos Criptográficos

Nas últimas décadas a informação, que era usualmente transmitida e armazenada em papel, vem se concentrando cada vez mais em discos magnéticos, CDs, DVDs e outras memórias voláteis e não voláteis. Copiar e alterar documentos tornou-se muito mais fácil e técnicas visando garantir a Segurança da Informação tornaram-se necessárias.

Alcançar essa segurança requer o uso de uma gama enorme de técnicas, e uma ciência vem exatamente para fornecê-las: a Criptografia. [MEN 97]

2.1 Criptografia

O dicionário Concise Oxford (2006) define a Criptografia como sendo "a arte de escrever e resolver códigos". Contudo, de acordo com [JK 07] apesar de essa definição estar historicamente correta, ela não captura a essência da Criptografia moderna. [JK 07] define, portanto, a Criptografia moderna como sendo "o estudo de técnicas para proteger a Informações Digital, transações e a computação de forma distribuída". Outra definição, ilustrada em [MEN 97], afirma que a "Criptografia é o estudo de técnicas matemáticas relacionadas com a Segurança da Informação", a fim

de garantir:

- *Confidencialidade*: Propriedade que mantém o conteúdo da Informação escondido de todos, exceto dos indivíduos autorizados a acessá-lo;
- *Integridade*: Propriedade que garante que a informação manipulada mantenha todas as características originais estabelecidas pelo proprietário da informação;
- *Não-Repúdio*: Propriedade que previne uma entidade de negar ações feitas anteriormente;
- *Autenticidade*: Propriedade relacionado com a identificação e a segurança da origem da Informação.

Visando promover essas quatro propriedades básicas, funções matemáticas foram desenvolvidas com o intuito de possibilitar o embaralhamento da informação de maneira tão imprevisível, que a probabilidade de que se consiga desembaralhá-la fosse muito pequena. Esse processo é conhecido como cifragem.

O processo de cifragem de dados utiliza um segredo como entrada, conhecido como chave criptográfica. Somente conhecendo a chave correspondente àquela utilizada para cifragem é possível desfazer o processo de embaralhamento, obtendo novamente o conjunto inicial de dados (decifragem). Essa chave correspondente varia de acordo com o tipo de criptografia empregado, podendo ser a mesma para o caso da Criptografia Simétrica, ou uma correspondente única para o caso da Criptografia Assimétrica.

Além das funções utilizadas para embaralhar a informação, a base da Criptografia conta também com as Funções Resumo. Portanto, podemos classificar três classes de funções matemáticas:

- Funções Resumo;
- Criptografia Simétrica;
- Criptografia Assimétrica.

2.1.1 Funções Resumo

Funções Resumo são funções de compressão significando que, geralmente, a saída é menor que a entrada. Basicamente, essas funções recebem um conjunto de dados de tamanho qualquer e o comprimem, convertendo-o para outro geralmente de tamanho fixo. Funções Resumo são popularmente conhecidas como funções de *Hash*.

Essas funções devem ser resistentes a colisões, ou seja, a probabilidade de que dois conjuntos de dados levem para o mesmo resumo criptográfico deve ser muito baixa. Adicionalmente, deve ser igualmente baixa a probabilidade de que, a partir do resumo, obtenha-se a mensagem original.

Essas propriedades tornam as Funções Resumo ideais para para a verificação da integridade de mensagens. Isso se deve ao fato de que a menor modificação efetuada sobre uma determinada informação provoca alteração do resumo e, consequentemente, ele não mais representa a informação original. [JK 07]

2.1.2 Criptografia Simétrica

Criptografia Simétrica ou criptografia de chave privada é aquela em que ambas as partes envolvidas compartilham o mesmo segredo. Esse segredo, conhecido como chave criptográfica, é utilizado quando as partes desejam se comunicar de maneira secreta.

O processo tem início quando uma das partes utiliza a chave criptográfica para cifrar, antes de enviar, determinada mensagem. Como o receptor possui a mesma chave, ele a utiliza no processo de decifragem, obtendo a mensagem original.

É implícito nesse mecanismo que as partes envolvidas, de alguma forma, negociaram a chave utilizada no processo antes de qualquer comunicação ser feita. [JK 07]

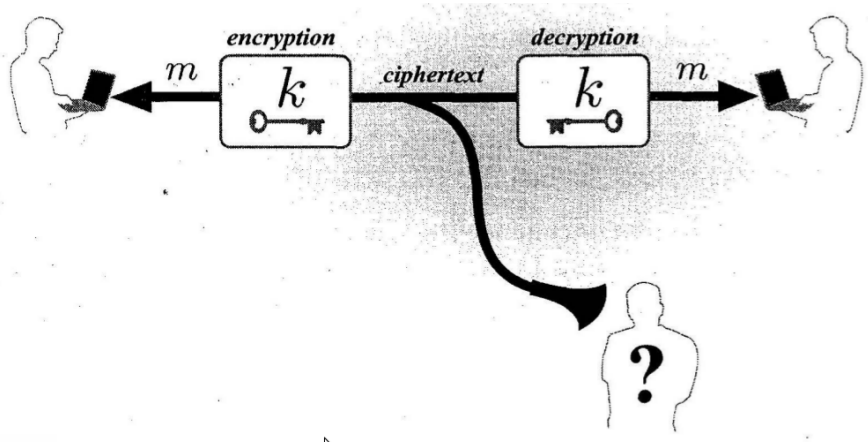

Figura 2.1: Figura representando a Criptografia Simétrica [JK 07].

2.1.3 Criptografia Assimétrica

A Criptografia Simétrica pode ser usada para o estabelecimento de comunicação segura através de um canal inseguro. Aparentemente, ela resolveria o problema primário da criptografia (troca de mensagens em sigilo), mas uma questão prevalece: como as chaves são inicialmente obtidas entre as partes interessadas?

Obviamente elas não podem ser enviadas através de um canal inseguro, uma vez que algum agente malicioso poderia obtê-las. Protocolos, como o acordo de chaves Diffie-Hellman [rfc 11], foram criados com o intuito de solucionar esse problema. Apesar de muito interessante e com diversas aplicações, esses protocolos ainda são suscetíveis a alguns ataques, como o ataque do homem do meio (man-in-the-middle attack) [JK 07]. Somente com o advento da Criptografia Assimétrica foi possível expandir a Criptografia para uma enorme gama de aplicações estando presente, direta ou indiretamente, no cotidiano das pessoas.

A Criptografia Assimétrica, também chamada de criptografia de chave pública, é aquela em que é utilizado uma chave para cifrar e outra, correspondente, para decifrar a informação. Especificamente, uma entidade (o destinatário) gera um par de chaves $\{Kr, Ku\}$ chamadas de *chave privada* e *chave pública*, respectivamente, e envia *Ku* para o remetente. O remetente então utiliza a chave pública para cifrar

a mensagem antes de enviá-la, garantindo que somente o proprietário de *Kr* poderá decifrá-la. A criptografia assimétrica é surpreendente pois duas entidades são capazes de trocar mensagens secretas sem terem acordado qualquer chave.

2.2 Dispositivos Criptográficos

Dispositivos que armazenam informações criptográficas e são capazes de realizar tarefas de criptografia são chamados de Dispositivos Criptográficos. Podem ser implementados como Smart Cards, USB Tokens, HSMs e até mesmo, utilizando técnicas apropriadas, em software. [RSA 11a]

A segurança dos algoritmos de criptografia, tanto simétrica como assimétrica, consiste em manter a chave sensível em sigilo. Caso essa chave seja comprometida, informações secretas poderão ser reveladas e as quatro propriedades citadas na seção 2.1 não serão mais verificadas.

Dentro desse contexto, os Dispositivos Criptográficos possuem um papel fundamental, oferecendo proteções físicas e lógicas ao material sensível.

2.2.1 Smart Card

Existem no mercado diversos tipos de cartões utilizados para autenticação de seus usuários. De maneira geral, esses cartões possuem formato retangular e tem seus atributos padronizados por normas definidas pelos seguintes órgãos:

- *International Organization for Standardization* (ISO) [1];
- *International Electrotechnical Commission* (IEC) [2].

A primeira classificação que podemos fazer diz respeito à necessidade de existir contato com a respectiva leitora. Cartões sem contato (do inglês contactless)

[1] Comumente traduzida como Organização Internacional para Padronização ou Organização Internacional de Normalização

[2] Comumente traduzida como Comissão Eletrotécnica Internacional

e cartões com contato (do inglês contact) estão disponíveis. Cartões sem contato são baseados na tecnologia *MIFARE ISO/IEC 14443* e, de acordo com [NK 07] [NK 08] [Dig 11], não apresentam o mesmo nível de segurança de determinados cartões com contato, especificamente, dos Smart Cards, definidos em seguida.

Cartões com contato foram utilizados para este trabalho e, de acordo com [RAN 97], sua classificação é resumida na figura X.

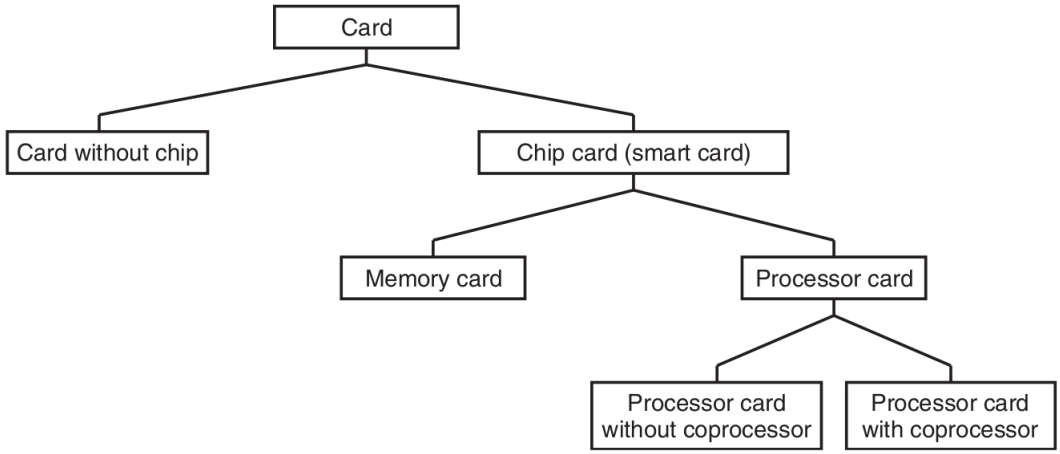

Figura 2.2: Visão geral dos diferentes tipos de Smart Cards [RAN 97].

Os cartões podem ser divididos em cartões sem *chip* e cartões com *chip*. Estes, como mostra a figura, também são chamados de Smart Cards e constituem Dispositivos Criptográficos.

Os *chips* presentes nos Smart Cards podem ser de dois tipos e dão nome aos cartões: *chips* de memória, contidos em cartões de memória e *chips* de microcontroladores, contidos em cartões de processadores. Esses últimos podem ainda ser subdivididos em cartões de processadores com ou sem coprocessadores, coprocessador este responsável pela execução de algoritmos de Criptografia Assimétrica.

Para este trabalho focaremos no uso dos cartões capazes de realizar criptografia de chave pública, pois a autenticação dos usuários do ASI-HSM [Lab 11] é baseada nesse tipo de criptografia. Alguns modelos, como o *Siemens CardOS V4.3*, *G&D STARCOS SPK 2.4* e *Oberthur IAS-ECC* serão utilizados.

Figura 2.3: Cartão Inteligente e leitora.

2.2.2 HSM

Um Hardware Security Module (Módulo de Segurança Criptográfica) ou simplesmente HSM, é um Dispositivos Criptográfico desenvolvido para oferecer a maior proteção possível ao seu material sensível. Sua utilização assemelha-se a dos Smart Cards. Todavia, em escala muito maior.

Em termos de tipos de HSMs, existem basicamente dois:

- *Network Attached HSM*;
- *Embedded HSM*.

Network Attached HSMs são aqueles cuja comunicação é feita através de protocolos de redes de computadores. Obviamente, um túnel seguro é estabelecido entre a aplicação e o dispositivo, visando coibir a ação de agentes maliciosos. Este trabalho desenvolve suporte para um HSM chamado de ASI-HSM [Lab 11], o qual é classificado como Network Attached.

Já os módulos embarcados (Embedded) geralmente fazem uso de um barramento (PCI, por exemplo) para realizar a comunicação com seus usuários. Esse tipo de HSM geralmente é menor, sendo utilizado em contextos mais restritos, quando não existe a necessidade de múltiplos pontos de acesso.

O grande diferencial entre um HSM e uma plataforma computacional capaz de realizar tarefas de criptografia como, por exemplo, um servidor, é sua proteção contra adulteração. Essa proteção, designada em inglês como *tamper protection*, deve detectar toda e qualquer tentativa de adulteração irregular dos dados. Uma vez detectada tais tentativas, medidas devem ser tomadas para proteger o material sensível,

Figura 2.4: Exemplo de Network Attached HSM - ASI-HSM

muitas vezes sendo necessário apagar todo seu conteúdo. Comumente, um HSM é capaz de fornecer as seguintes funcionalidades:

- Gerenciamento de Chaves criptográficas;
- Aceleração criptográfica;
- Proteções físicas e lógicas avançadas.

A exemplo dos Smart Cards, diversas normas internacionais regem o desenvolvimento de HSMs. As mais largamente aceitas tanto entre governos, como entre consumidores de HSMs, são as normas *NIST FIPS 140-2* [FED 11] e *MCT-7* [Ins 11].

Capítulo 3

Arquitetura de Acesso a Smart Cards

A arquitetura mais adotada atualmente para acesso a Smart Cards, a partir de computadores pessoais, é baseada na proposta do PC/SC WorkGroup [1] . Esse grupo de trabalho, formado em 1996, conta com a presença de grandes fabricantes de software e hardware e sua arquitetura é implementada nativamente por sistemas Windows. Ela também está disponível para sistemas Unix, Linux e Mac. Porém, não de forma nativa.

A formação desse grupo de trabalho veio com o intuito de promover uma especificação padrão, a fim de que Smart Cards, Smart Cards Readers (leitoras de Smart Cards) e computadores, feitos por diferentes fabricantes, possam trabalhar conjuntamente através de interfaces de acesso padronizadas. Dessa forma, aplicações que desejam utilizar Smart Cards terão seu desenvolvimento facilitado. [PC/ 05a]

Este trabalho baseou-se em sua arquitetura para planejar e implementar soluções.

[1] http://www.pcscworkgroup.com/

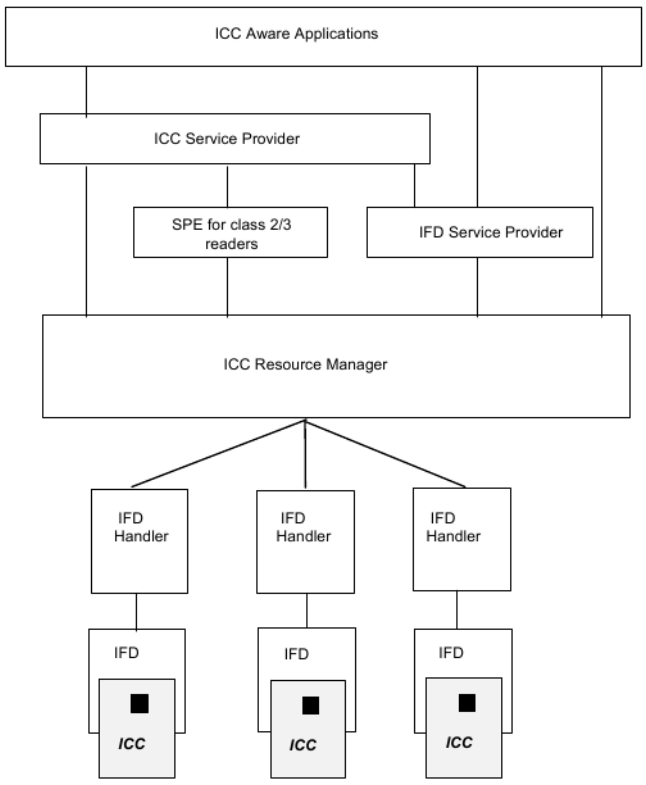

Figura 3.1: Arquitetura PC/SC proposta [PC/ 05a].

3.1 Componentes da Arquitetura

A seguir, as camadas da arquitetura terão suas funções descritas de acordo com a especificação [2]. Além disso, uma vez que existem padrões largamente utilizados que não estão presentes na especificação PC/SC, como a interface PKCS #11 e a estrutura PKCS #15 por exemplo, o autor acredita que esses padrões encontram um local apropriado dentro dela e os apresentará como tal, justificando seu posicionamento.

De fato, outra motivação para englobar as explicações baseando-se no padrão proposto pelo PC/SC Workgroup é a maneira como é feito na prática. A maior parte das implementações da interface PKCS #11, explicada posteriormente na seção

[2] As camadas *SPE for class 2/3 readers* (para leitoras com mecanismos de PIN seguro) e *IFD Service Provider* (para leitoras com funcionalidades extras), fogem do escopo deste trabalho e não serão explicadas.

3.6.1, por exemplo, comunicam-se diretamente com elementos presentes na arquitetura PC/SC.

3.2 ICC

Integrated Circuit Card (ICC), comumente chamado de Smart Card, representa um Cartão Inteligente propriamente dito (fisicamente). Para serem compatíveis com este padrão, os cartões devem estar de acordo com as normas *ISO/IEC 7816-1*, *ISO/IEC 7816-2*, *ISO/IEC 7816-3*, *ISO/IEC 14443* (para cartões de proximidade) e *ISO/IEC 7816-10* (para cartões síncronos). [PC/ 05b]

É interessante, para o escopo do trabalho, destacar as primeiras três normas:

- *ISO/IEC 7816-1*: Especifica as características físicas dos cartões compatíveis, como as dimensões e a localização do Circuito Integrado (IC);

- *ISO/IEC 7816-2*: Define as dimensões e a localização dos contatos metálicos do cartão, assim como seus propósitos e características elétricas;

- *ISO/IEC 7816-3*: Define os sinais elétricos e protocolos de transmissão, são eles:

  - Asynchronous half-duplex character transmission protocol (T=0);
  - Asynchronous half duplex block transmission protocol (T=1).

Assim como definido na especificação PC/SC, o desenvolvimento de Smart Cards é fortemente impulsionado por padrões internacionais. O fato da inexistência de monopólio no setor, desde o inicio do desenvolvimento de cartões, além da grande necessidade de interoperabilidade contribuíram para esse fato. [RAN 97]

Por esse motivo, a maioria do cartões disponíveis no mercado segue as normas *ISO/IEC 7816* e, consequentemente, são PC/SC compatíveis.

3.2.1 PKCS #15

As normas necessárias para que um Smart Card seja compatível com o padrão PC/SC padronizam a maneira como é feita a comunicação entre cartões e leitoras. Contudo, elas não definem como os objetos, chaves e certificados por exemplo, devem ser guardados e posteriormente acessados. Isso cria dificuldades nos níveis mais altos, gerando a necessidade de comandos específicos para cada cartão.

O padrão PKCS #15 veio com o intuito de resolver esse problema. Ele define um sistema de arquivos padronizado, contendo diversos diretórios que, por sua vez, possuem ponteiros para listas de objetos presentes no cartão, como mostrado na figura 3.2. Dessa maneira, uma aplicação esperando um cartão formatado no padrão PKCS #15 sabe exatamente onde procurar por chaves, certificados, PINs, etc.

O objetivo deste padrão, portanto, consiste em promover a portabilidade de credenciais criptográficas através de múltiplas aplicações. Em outras palavras, visa permitir que usuários de Smart Cards sejam capazes de se identificar para múltiplas aplicações em diferentes ambientes.

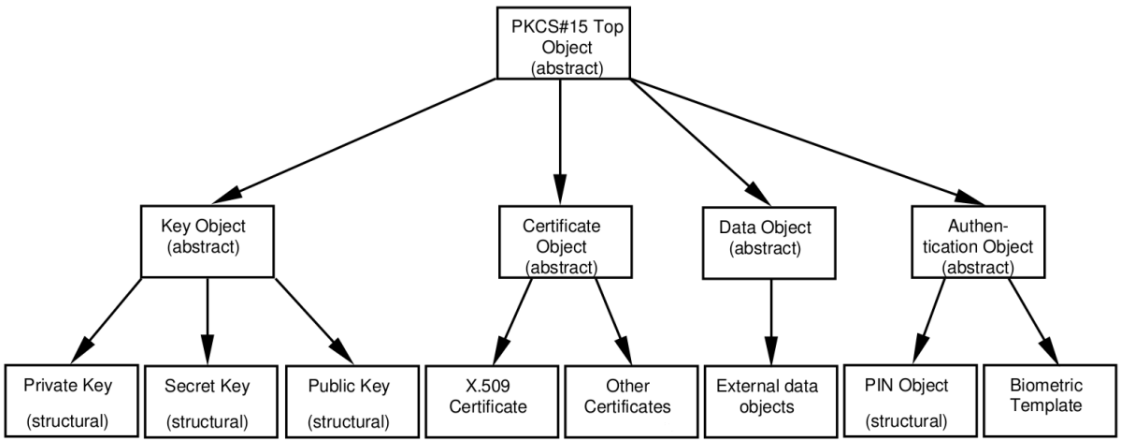

Figura 3.2: Hierarquia de objetos PKCS #15 [RSA 11b].

Desde o início, o padrão PKCS #15 baseou-se nas normas *ISO/IEC 7816-4 ISO/IEC 7816-5* e *ISO/IEC 7816-6*. Em especial, a norma *ISO/IEC 7816-4* define

um conjunto de comandos, denominados de Application Protocol Data Unit (APDU) (explicado em detalhes na seção 3.4.2), básicos necessários para ler, escrever e atualizar objetos no cartão. Por esse motivo, a compatibilidade entre cartões PKCS #15 formatados está garantida somente para esse conjunto exclusivo de APDUs.

De fato, este trabalho verificou que apenas alguns comandos são interoperáveis, geralmente aqueles relativos à listagem de objetos. A indústria, muitas vezes, por necessitar de uma gama maior de comandos, define APDUs proprietários fora da norma *ISO/IEC 7816-4* [3]. Isso culmina com a existência de vários APDUs compatíveis apenas com os cartões para os quais foram criados, dificultando a interoperabilidade. As normas *ISO/IEC 7816-8* (para operações de segurança) e *ISO/IEC 7816-9* (para comandos relacionados à gerência do cartão) definem novos conjuntos de APDUs e poderiam expandir o padrão PKCS #15, possivelmente resolvendo o problema da compatibilidade.

Vale lembrar que a norma *ISO/IEC 7816-15*, desde 2004, definiu um padrão baseado no PKCS #15. Consequentemente, cartões que implementam todas as normas *ISO/IEC 7816* e não definem APDUs proprietários são interoperáveis, pois aceitam os mesmos comandos.

3.2.2 ATR

Answer To Reset (ATR) é a sequência de *bytes* de saída de um Smart Card após ter sido eletricamente resetado por uma leitora. O formato do ATR é definido pela norma *ISO/IEC 7816-3*.

O ATR faz parte dos atributos que um Smart Card deve fornecer e, por esse motivo, está sendo explorado na seção ICC. De acordo com a especificação PC/SC, o ATR deve ser utilizado para identificar diferentes Smart Cards [PC/ 05a].

[3] É importante destacar que APDUs propritários não dizem respeito à forma de comunicação, ou seja, não se referem ao tamanho da unidade, em bytes, de comunicação entre cartões e leitoras. Referem-se, de fato, ao conteúdo dessa unidade de comunicação.

3.3 IFD

Interface Device (IFD), representa uma leitora de Smart Cards. As leitoras devem obedecer aos mesmos padrões de comunicação definidos para os ICC (normas *ISO/IEC 7816-1*, *ISO/IEC 7816-2* e *ISO/IEC 7816-3*). Além disso, elas devem possuir mecanismos de detecção de inserção e remoção de cartões, além de fornecer suporte para as funcionalidades expostas pela interface do IFD Handler, explicado em seguida na seção 3.4. O IFD deve, portanto, incorporar as funcionalidades necessárias para dar suporte à interface exposta. O padrão não impõe restrições quanto ao tipo de canal I/O utilizado, mas recomenda o uso da interface USB.

3.4 IFD Handler

IFD Handler representa o driver que coordena a comunicação, através do canal I/O utilizado pela leitora (usualmente USB), entre o computador e o IFD. O padrão PC/SC define uma interface padronizada para esses drivers, baseada na norma *ISO/IEC 7816-4*, e pode ser consultada em [PC/ 07]. Precisamente, então, um IFD Handler representa o software necessário para mapear as capacidade nativas de uma leitora de Smart Cards para essa interface. [PC/ 07]

3.4.1 CCID Readers

Uma vez que o padrão PC/SC define uma interface padronizada, IFDs que implementam o padrão PC/SC e escolheram o canal USB podem se beneficiar de um IFD Handler genérico, chamado de Chip/Smart Card Interface Device (CCID). Esse *driver* é embarcado em sistemas Windows e acompanha o pacote *libccid*, instalado como dependência do pacote *pcscd* em sistemas Unix, Linux e Mac.

Basicamente, esse driver responde à pergunta “Necessito de um driver para minha leitora?”. Ele, portanto, especifica como a leitora deve ser acessada através do canal USB, tornando o uso de leitoras bastante transparente e criando o conceito de

"leitoras CCID".

3.4.2 APDU

Application Protocol Data Unit (APDU) é a unidade de comunicação entre uma leitora de Smart Cards e um Smart Card. A estrutura dos APDUs é definida pela norma *ISO/IEC 7816-4*, assim como os APDUs básicos para realizar a comunicação.

Existem duas categorias de APDUs: APDUs de comando e APDUs de resposta. APDUs de comando consistem de um cabeçalho (*header*) obrigatório e de um corpo (*body*) opcional. Já os APDUs de resposta são compostos por um corpo opcional e um "*trailer*" obrigatório. [RAN 97]

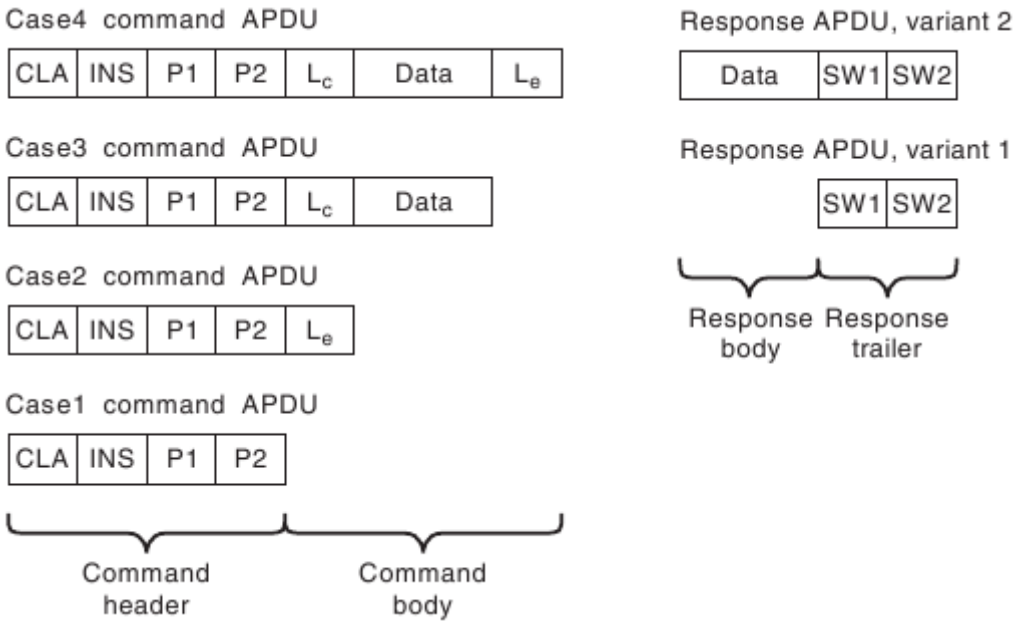

Figura 3.3: Os quatro diferentes tipos de APDUs de comando e as duas variantes de resposta [RAN 97].

Um APDU de comando consiste em quatro *bytes*: *CLA* é a classe (do inglês *class*), *INS* é a instrução (do inglês *instruction*) e *P1* e *P2* são parâmetros. A classe representa um conjunto padrão de comandos, análogo a classes em paradigmas orientado a objetos, enquanto que a instrução representa o comando propriamente dito. Os

parâmetros, por sua vez, fornecem informações adicionais. [RAN 97]

O corpo contém até três elementos de dados. O primeiro, *Lc*, representa o tamanho dos dados contidos no comando, enquanto que o último, *Le*, contém o tamanho dos dados esperados na resposta. *Data*, obviamente, contém os dados (quando houverem) trafegados. [RAN 97]

Os APDUs de resposta são compostos de *SW1* e *SW2*, que representam o status da resposta, e de um conjunto de dados (*Data*) caso exista. [RAN 97]

3.5 ICC Resource Manager

De acordo com a especificação, o Gerenciador de Recursos (do inglês Resource Manager) possui múltiplas funções e desempenha um papel chave dentro da arquitetura. Ele é responsável por fornecer acesso controlado a leitoras e, através delas, a cartões.

Em primeiro lugar, ele é responsável pela identificação e rastreamento de recursos, devendo:

1. Rastrear leitoras disponíveis e torná-las acessíveis para aplicações, associando-as a seus respectivos IFD Handlers;

2. Rastrear cartões conhecidos, assim como seus respectivos Service Providers, e tornar essa informação acessível para outras aplicações;

3. Rastrear a inserção e remoção de cartões e de leitoras, mantendo informações precisas sobre o estado de cada componente.

Para suportar o item 2 acima, é necessário que exista uma pré instalação. Esse passo possibilita que o Resource Manager consiga identificar cada cartão e associá-lo ao respectivo Service Provider. Explicações sobre como o Resource manager identifica cada cartão foi descrito anteriormente, na seção 3.2.2, ATR. Os sistemas Windows, como dito anteriormente, possuem uma implementação fiel desta especificação e, consequentemente, possuem o passo da instalação. Já sistemas Linux e Unix não

implementam o item 2 acima, fazendo com que seja função da aplicação realizar a ligação, estática ou dinâmica, com o Service Provider.

Em segundo lugar, ele é responsável por controlar a alocação de IFDs e recursos através de múltiplas aplicações. Ele faz isso provendo mecanismos para o uso de uma determinada leitora (e consequentemente de um cartão) em modos exclusivos ou compartilhado.

Finalmente, ele deve fornecer primitivas de acesso para os serviços disponíveis em um cartão. O Resource Manager realiza essa tarefa disponibilizando uma User Interface (UI), ou seja, uma Interface de Usuário, para aplicações. A especificação define todos os mecanismos e funções que essa interface deve ter e ela é bem implementada em sistema *Unix-like*. Basicamente, ela fornece às aplicações a possibilidade de detectar a inserção e remoção de cartões e leitoras, detectar o ATR proveniente do cartão e enviar e receber APDUs.

3.6 ICC Service Provider

Esta camada representa um Provedor de Serviços para acesso a Smart Cards. Esses provedores são responsáveis por encapsular funcionalidades presentes nos cartões através de interfaces de alto nível.

O padrão define, basicamente, dois tipos de provedores de serviços relacionados com Smart Cards: ICC Service Provider (ICCSP) e Cryptographic Service Provider (CSP). O primeiro, podendo ser traduzido como um Provedor de Serviços para Cartões Inteligentes, é responsável por fornecer todas as funcionalidades não relacionadas com criptografia, das quais se destacam:

- Localizar arquivos através de seus nomes;
- Criar e abrir arquivos;
- Ler e escrever em arquivos;
- Fechar arquivos;

- Deletar arquivos;
- Gerenciar atributos de arquivos;
- Gerenciar a autenticação de usuários do cartão.

Já o CSP, que pode ser traduzido como um Provedor de Serviços Criptográficos, é responsável por encapsular apenas as funcionalidades relacionadas com criptografia. Dentre elas, podemos enfatizar:

- Criação de chaves;
- Gerenciamento de chaves;
- Assinaturas digitais;
- Resumos Criptográficos (*Hashing*);
- Importação e exportação de chaves.

Essa separação fundamentou-se em questões políticas. Uma vez que diferentes governos exigem diferentes níveis de segurança e, consequentemente, diferentes algoritmos criptográficos, essa separação facilita o desenvolvimento e a portabilidade de cartões entre diferentes países e/ou regiões. [PC/ 11]

3.6.1 PKCS #11 [RSA 11a]

A interface PKCS #11, também chamada de Cryptoki, define uma Application Programming Interface (API) [4] de comunicação com Dispositivos Criptográficos. Ela foi desenvolvida independente do padrão PC/SC e abstrai todas as camadas inferiores, preocupando-se apenas em oferecer, às aplicações, uma interface padronizada de acesso a esses dispositivos. Note que as explicações a seguir focam no uso da Cryptoki para acesso a Smart Cards, mas o padrão pode ser estendido para qualquer Token criptográfico. O padrão define, estruturalmente, dois importantes níveis de abstração:

[4] Podendo ser traduzida como Interface de Programação de Aplicações

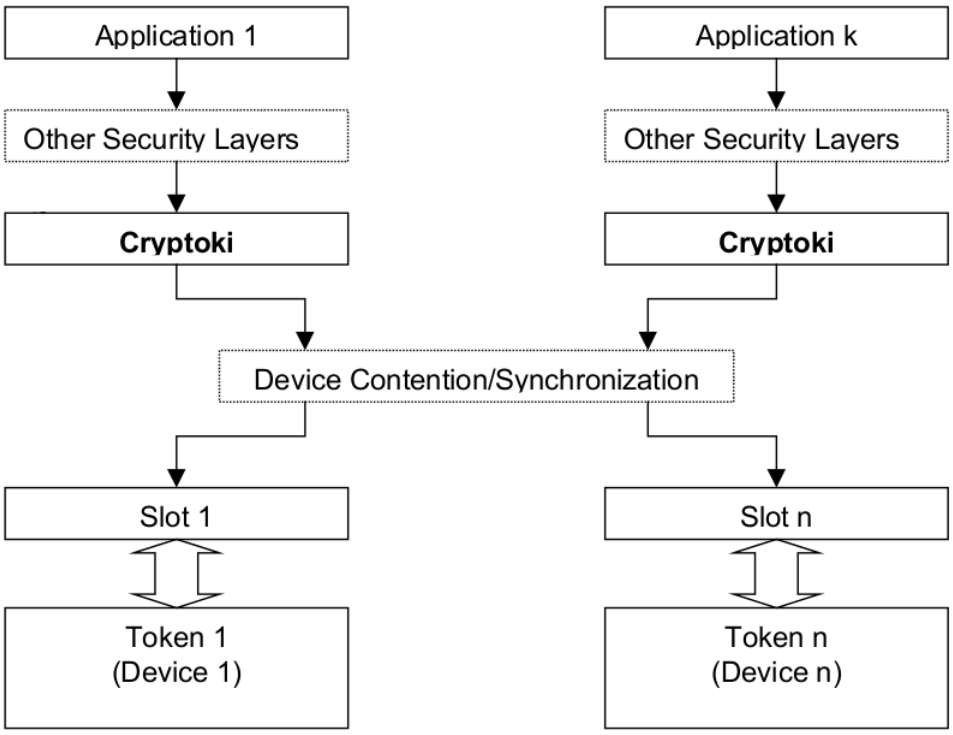

Figura 3.4: Modelo geral Cryptoki [RSA 11a].

- Token: é a visão lógica de um Smart Card (ou qualquer dispositivo criptográfico, como USB Tokens ou HSMs);

- Slot: é a visão lógica de uma leitora, podendo ou não conter um token inserido.

A partir do momento em que um Slot contém um Token, é possível interagir com o cartão e, para essa interação, a interface PKCS #11 define uma estrutura orientada a objetos (vide figura 3.5). A interface PKCS #11 não está definida no padrão PC/SC. porém, o que é feito nos sistemas mais populares atualmente é colocá-la exatamente na camada de um Service Provider, implementando-a para interagir diretamente com a interface de aplicação (UI) do Resource Manager (vide seção 3.5). Por esse motivo, a Cryptoki comporta-se como um Service Provider único, unindo as funcionalidades de um ICCSP e de um CSP.

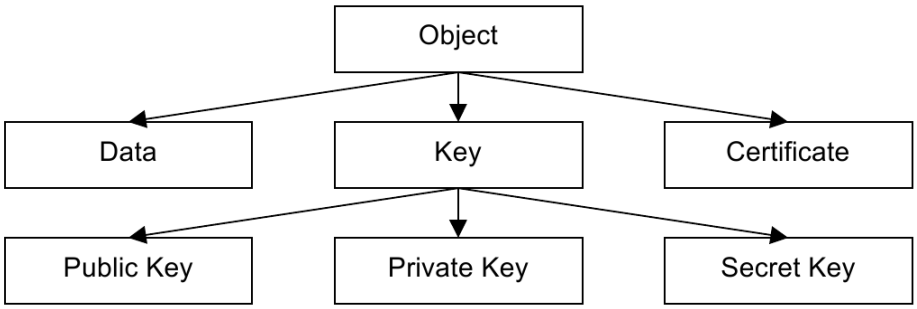

Figura 3.5: Estrutura de Objetos PKCS #11 [RSA 11a]

3.7 ICC Aware Applications

Esta camada representa qualquer aplicação que deseje se comunicar com Cartões Inteligentes. Como mostrado na figura 3.1 uma aplicação pode, basicamente, realizar a comunicação com os Cartões Inteligentes de duas maneiras: através de um Service Provider ou diretamente acessando a UI do Resource Manager. Ambas essas abordagens serão discutidas no Capítulo 4, apontando suas vantagens e desvantagens.

Podemos citar como exemplos de ICC Aware Applications os navegadores, como o Mozzila Firefox e o Internet Explorer.

Capítulo 4

Propostas

Com a arquitetura de acesso explicada, é possível entender a atual relação do ASI-HSM com seus Cartões Inteligentes.

O ASI-HSM é um HSM do tipo Network Attached, conforme explicado na seção 2.2.2, ou seja, toda a comunicação é feita através de protocolos de rede. Obviamente, é necessário que essa comunicação ocorra através de um túnel seguro, utilizando o protocolo TLS/SSL.

Para realizar a gerência, auditoria e operação do ASI-HSM, existem duas interfaces disponíveis:

- Interface gráfica Java;
- Interface texto.

Através das interfaces, os comandos são repassados para a unidade gestora do ASI-HSM onde executa, além do sistema operacional e de outros processos, o *Open-HSMd*.

O *OpenHSMd* é responsável por responder a qualquer comando externo e é a unidade lógica do ASI-HSM. O que nos interessa, para o escopo deste trabalho, é justamente a interação desse componente com a leitora conectada e, consequentemente, com seus Smart Cards.

A figura 4.1 explica o funcionamento simplificado do ASI-HSM no aspecto

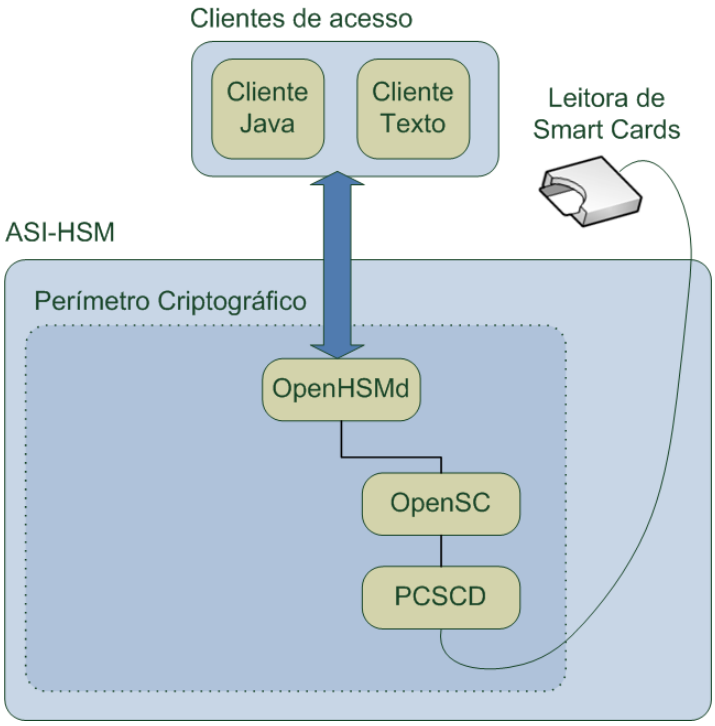

Figura 4.1: Interação entre o ASI-HSM e seus Smart Cards.

relativo à sua interação com os Smart Cards. Como internamente o ASI-HSM roda um Sistema Operacional *FreeBSD*, os seguintes pacotes são instalados para garantir o funcionamento:

- *opensc*;
- *pcscd*;
- *libpcsclite1*;
- *libccid*.

O pacote *pcscd* depende dos pacotes *libccid* e *libpcsclite1*, por isso a figura 4.1 simplificou a modelagem. O pacote *pcscd*, conjuntamente com o pacote *libpcsclite1*, formam o Resource Manager, explicado na seção 3.5. Já o pacote *libccid* implementa um IFD Handler genérico para leitoras USB PC/SC compatíveis, como explicado na seção 3.4.1.

Pode-se notar que não existe código específico para tratar da interação com Smart Cards, deixando essa tarefa a cargo dos utilitários do OpenSC.

4.1 OpenSC

O OpenSC, pacote *opensc*, é um conjunto de utilitários e bibliotecas construídos para trabalhar com Smart Cards. Esse projeto implementa, a nível de Service Provider (vide seção 3.6) , a interface PKCS #11 e a nível de ICC (vide seção 3.2), a estrutura PKCS #15.

Eles focam em suportar múltiplos cartões. Para isso, formatam os cartões suportados utilizando o padrão PKCS #15 e posteriormente os utilizam através de uma única interface PKCS #11, implementada com o nome de *opensc-pkcs11.so* [1].

De acordo com os mantenedores do projeto OpenSC, nem todos os fabricantes implementam inteiramente o padrão PKCS#15 e, mesmo se o fazem, os cartões continuam incompatíveis devido a detalhes internos de implementação [2]. O OpenSC, então, implementa um *driver* específico para cada tipo de cartão suportado, a fim de lidar com as diferenças de implementações e tornar possível o uso de uma interface única (PKCS #11).

Eles escreveram diversos *driver* e muitos cartões são suportados. Uma lista de todos os cartões suportados pode ser encontrada em [Ope 11];

4.2 Os Problemas

O ASI-HSM utiliza o OpenSC para a comunicação com seus Cartões Inteligentes. Precisamente, ele faz uso dos seguintes utilitários:

[1] O uso não se restringe apenas à biblioteca *opensc-pkcs11.so*, existem outras bibliotecas para o acesso, como por exemplo a *libopensc.so*. Porém, essa biblioteca não segue padrão algum, sendo utilizada apenas para projetos relacionados com o OpenSC.

[2] Dados obtidos a partir das listas de discussões oficiais do projeto OpenSC: opensc-user@lists.opensc-project.org. e opensc-devel@lists.opensc-project.org.

- *pkcs15-init*: utilizado para inicializar o cartão e importar chaves e certificados;
- *pkcs15-tool*: utilizado para as operações de criptografia, como assinaturas digitais, cifragem e decifragem de dados.

Essa abordagem gera diversos problemas, os quais são detalhados a seguir.

4.2.1 Primeiro problema: Comandos *system*

O uso dos Smart Cards pelo OpenHSMd é feito através de comandos *system*. Esse tipo de instrução chama o processador de comandos do sistema operacional para executar o comando passado como argumento. Após ter finalizado a execução do comando, o processador devolve o controle para o programa, retornando um valor *int*, cuja interpretação é dependente de sistema. o trecho de código a seguir exemplifica seu uso:

Listing 4.1: Exemplo de comando *system* em C.

```
...
char cmd[] = “pkcs15 −init −C";
if (system(cmd))
{
   // Ocorreu erro na chamada system
   exit(1);
}
...
```

Apesar de esses utilitários serem bem testados e confiáveis, o valor *int* retornado é genérico demais, não sendo possível realizar um tratamento de erros preciso. Uma simples operação de *login* no cartão, por exemplo, pode retornar diversos códigos de erro. Alguns deles estão listados abaixo:

- PIN incorreto;

- PIN bloqueado;
- PIN não inicializado;
- Usuário já logado;
- Sessão não inicializada.

Sem precisão nos códigos de erro, vários problemas surgem:

- O usuário não sabe se ele está digitando o PIN incorreto, levando-o a bloquear seu cartão (três tentativas incorretas bloqueiam o PIN de usuário);
- O usuário não sabe se sua autenticação perante o ASI-HSM falhou devido a mal contato do cartão ou erro em alguma operação de criptografia;
- O ASI-HSM não sabe se o erro ocorreu no cartão ou na leitora (e qual foi esse erro).

A robustez de um sistema envolve tratamentos de erros precisos. Por isso, é fundamental saber exatamente onde o erro ocorreu e tratá-lo da maneira correta.

Além disso, comandos *system* resultam em chamadas de sistema por parte do Sistema Operacional. Cada comando *system* realiza um *exec()*, chamando um processo filho para executar o comando, e devolver o resultado em questão. É feito, portanto um *fork()* nesse processo, ou seja, uma chamada de sistema. De acordo com [Rob 11] chamadas de sistema representam vulnerabilidades de código sendo, portanto, essencial sua eliminação em sistemas de segurança.

4.2.2 Segundo Problema: Reuso de Cartões Inteligentes

O foco do projeto OpenSC sempre foi no uso dos cartões e não na alteração do seu conteúdo. Por isso, ele fornece bom suporte para uma sequencia básica de operações. Essa sequencia consiste nas seguintes etapas:

1. Inicialização;

2. Geração de objetos, como par de chaves e/ou certificados;

3. Uso simplificado, consistindo de assinaturas digitais e/ou cifragem e decifragem de dados.

No item 3, o fluxo termina, não sendo possível, de nenhum ponto, reiniciá-lo. Portanto, a partir do momento em que um cartão é inicializado com o OpenSC, ele não pode mais ser reinicializado. Para agravar o problema, qualquer objeto criado no cartão não pode ser destruído.

Isso conflita com todos os padrões. Tanto o padrão PC/SC, como a interface PKCS #11 preveem a destruição de objetos e a reinicialização de Tokens. Esse problema é grave e causa o desperdício de cartões.

4.2.3 Terceiro Problema: Limitação do Número de Cartões Suportados

Apesar de o OpenSC suportar uma grande quantidade de cartões, muitos não são suportados. Fornecer suporte para uma gama possivelmente ilimitada de cartões é um dos objetivos deste trabalho e o OpenSC não parece ser a solução ideal.

4.2.4 Quarto Problema: Limitação de Funcionalidades Importantes

No contexto do OpenSC, as funcionalidades que cada cartão pode oferecer variam de acordo com as implementações feitas em cada driver (vide seção 4.1). Vários mecanismos, contudo, não são implementados e acabam limitando as funcionalidades presentes nos cartões.

Dentre elas, podemos citar os mecanismos de *Wrap* e *Unwrap*, explicados em detalhes na seção 5.2.2. Nota-se, pela figura 4.1, que o fio ligando a leitora de Smart Cards ao ASI-HSM está fora do perímetro criptográfico. Em geral, isso não gera riscos à segurança. Porém, em algumas situações especificas, como na importação de chaves para dentro do cartão, a segurança poderia ser elevada utilizando esses mecanismos.

Outro aspecto refere-se ao uso de Criptografia de Curvas Elípticas (Eliptic Curve Cryptography). Novos cartões aparecem no mercado constantemente e muitos já suportam não somente chaves RSA, como também chaves de Curvas Elípticas. Não é possível prever quando o OpenSC irá fornecer suporte para esse tipo de criptografia.

4.3 Propostas

Diversas propostas foram levantadas e analisadas a fim de melhorar o modelo atual, tendo sempre em mente a solução dos quatro problemas explicitados anteriormente. As subseções seguintes resumem cada uma delas e apontam a eleita, deixando claro o porquê da escolha.

4.3.1 Proposta 1: Utilizar o Módulo PKCS #11 do OpenSC

O OpenSC fornece um módulo PKCS #11, chamado de *opensc-pkcs11.so*, para interagir com os Cartões Inteligentes. Ao codificar as funcionalidades presentes nos utilitários em chamadas para a interface PKCS #11, seria possível remover boa parte das chamadas *system* e, consequentemente, melhorar o tratamento de erros. Essa abordagem resolveria, em parte, o problema dos comandos *system* (vide seção 4.2.1), mas deixaria todos os outros problemas em aberto, não sendo adequada.

O motivo da afirmação de que o módulo PKCS #11 do OpenSC removeria apenas *parte* dos comandos *system* é devido ao fato de ele não implementar todas as funções definidas na Cryptoki. Esse assunto é explorado em detalhes na seção 5.2.1.

4.3.2 Proposta 2: Trabalhar Diretamente com a UI do Resource Manager

O padrão PC/SC não requer necessariamente o uso de um Service Provider [PC/ 11]. Como mostrado na figura 3.1, é possível realizar a comunicação diretamente entre aplicações e a Interface de Usuário (UI) do Resource Manager. Essa abordagem

parece interessante pois, a princípio, resolveria todos os problemas anteriores, como explicado a seguir.

Trabalhar diretamente com a UI do Resource Manager tornaria possível a detecção de eventos relacionados. Isso possibilitaria, conjuntamente com a funcionalidade da UI de obter o ATR de cada cada cartão, identificar diferentes Smart Cards e trabalhar de maneira distinta com cada um deles. Para isso, bastaria manter um arquivo de configuração semelhante ao do exemplo abaixo [3]:

Listing 4.2: Ex. de arquivo de configuração contendo o mapeamento do ATR para o modelo de cartão.

```
3BF2180002C10A31FE58C80874=cartaoX
3BB71800C03E31FE6553504B3234900025=cartaoY
```

Ao detectar a inserção de um Smart Card, a aplicação consultaria o arquivo de configuração para verificar se o cartão está na lista de cartões suportados. Se sim, utilizaria código específico para ele.

Essa abordagem resolveria todos os problemas, da seguinte forma:

- Problema dos comandos *system* (vide seção 4.2.1): Os utilitários do OpenSC não seriam utilizados e a UI do PC/SC forneceria os código de erros necessários;
- Problema do reuso de cartões (vide seção 4.2.2): A grande maioria dos Smart Cards permite reinicialização e destruição de objetos;
- Problema da limitação de cartões suportados (vide seção 4.2.3): Código específico seria implementado para cada novo cartão suportado.
- Problema da limitação de funcionalidades (vide seção 4.2.4): A UI não impõe restrições às funcionalidades presentes nos cartões.

[3] Por motivos de segurança esse arquivo não pode permanecer em claro no disco, pois algum agente malicioso poderia adulterá-lo. Ele deve, portanto, ser assinado por alguma fonte confiável ou ser mantido *hardcoded*.

Apesar de essa proposta parecer animadora, o esforço para implementá-la e mantê-la seria enorme. A UI do Resource Manager trabalha com primitivas de comunicação, ou seja, diretamente com APDUs (vide seção 3.4.2). Trabalhar dessa forma é custoso, pois não existem abstrações. Deve-se, portanto, interpretar diretamente as sequencias de *bytes* contidas nos APDUs.

Além disso, como explicado na seção 3.2.1, a indústria possui diversos APDUs proprietários e, muitas vezes, os cartões não seguem todas as normas *ISO/IEC 7816*. Isso torna o código necessário para lidar com cada cartão totalmente dependente do mesmo.

Concluímos, então, que o esforço não compensa os benefícios dessa abordagem. Uma vez que um código dependente de cada modelo de cartão é difícil de desenvolver, manter e evoluir, ela está descartada.

4.3.3 Proposta 3: Utilizar Módulos PKCS #11 Proprietários

Os fabricantes de cartões fornecem, aos seus compradores, módulos compatíveis com os padrões de mercado. Uma vez que interface PKCS #11 é a interface *de-facto* para acesso a Tokens criptográficos, vários fabricantes a disponibilizam. Além disso, visto que não é de interesse dos fabricantes não implementar, em seus módulos PKCS #11, todas as funcionalidades presentes nos cartões, esses módulos não são afetados pelos problemas do reuso de cartões (vide seção 4.2.2) e da limitação de funcionalidades (vide seção 4.2.4). O problema dos comandos *system* (vide seção 4.2.1), por sua vez, é implicitamente resolvido pois, como na proposta 1 (vide seção 4.3.1), codificaríamos diretamente utilizando a interface PKCS #11.

Aparentemente essa proposta nos força a utilizar um módulo PKCS #11 exclusivo e, consequentemente, um modelo (ou família de modelos) de cartão único, deixando em aberto o problema da limitação de cartões suportados (vide seção 4.2.3). Todavia, é possível propor uma solução para essa questão. Ela consistiria em manter um repositório de módulos PKCS #11 no interior do ASI-HSM, sendo consultado toda vez que houvesse necessidade de utilização de um cartão. A técnica utilizada seria a

seguinte:

1. Carrega-se o módulo PKCS #11 n [4] da lista;
2. Verifica-se se esse módulo é o correto;
3. Se for ele é utilizado, senão incrementa-se n e retorna para o passo 1;

Essa solução, contudo, não é apropriada. Como explicado na seção 3.2.1, a estrutura PKCS #15 conjuntamente com a família de normas *ISO/IEC 7816* (especialmente as normas *ISO/IEC 7816-4* e *ISO/IEC 7816-15*) vieram com o intuito de padronizar a maneira como os cartões são acessados. Indústrias de Smart Cards costumam implementar alguns ou até mesmo partes desses padrões. Isso pode levar um módulo PKCS #11 a ter certa compatibilidade com determinado cartão para o qual não foi feito.

Para exemplificar esse cenário, suponhamos que o módulo N é feito para o cartão n e existem, no repositório de módulos, além de N, os módulos O e P. A maneira de descobrir se um cartão pertence a um determinado módulo, dentro desta abordagem, consistiria em carregá-lo e chamar uma ou algumas funções da interface PKCS #11. Em seguida, verificar-se-ia a corretude das respostas do cartão, tornando possível decidir se o módulo selecionado foi o correto.

Imaginemos, então, que o cartão p é inserido na leitora. Dentro do nosso algoritmo, vamos supor que seja testado o módulo N para verificar se ele é o correto. O módulo N chamaria, por exemplo, as funções X, Y e Z da interface PKCS #11. Nesse ponto, se não houvesse qualquer compatibilidade entre os cartões n e p, todas as funções falhariam. Porém, devido aos itens citados anteriormente, qualquer combinação poderia ocorrer, sendo possível até que as funções X, Y e Z fossem todas compatíveis. No entanto, mesmo que todas fossem compatíveis, várias outras poderiam não ser, ocasionando a carga do módulo incorreto e erros futuros imprevisíveis.

Para contornar esse problema, os testes para verificar se o módulo correto foi carregado seriam extensos, ou seja, um grande número de funções da interface

[4] n, neste contexto, significa um módulo PKCS #11 proprietário qualquer.

PKCS #11 deveriam ser chamadas. A frequência de *clock* do micro-processador de um Smart Card é muito baixa, geralmente em torno de 5MHz [RAN 97]. É fácil imaginar que essa abordagem teria, portanto, um impacto negativo na eficiência das transações.

4.3.4 Proposta 4: Solução Mista (adotada)

A UI do Resource Manager possui primitivas de baixo nível (APDUs) para realizar a comunicação com o cartão, complicando seu desenvolvimento (vide seção 4.3.2). Porém, através dela, é possível obter todas as informações de eventos relacionados a cartões e leitoras. É possível, portanto, detectar a inserção e remoção dos mesmos, assim como manter uma lista apurada de quantos desses dispositivos estão conectados e quais são seus estados. Somado a isso, a UI provê a habilidade de detecção do ATR de um Smart Card que, de acordo com o padrão PC/SC [PC/ 05a], é o mecanismo padrão de identificação de Cartões Inteligentes.

A proposta adotada baseia-se em um misto das propostas 2 (vide seção 4.3.2) e 3 (vide seção 4.3.3). Manteremos os módulos proprietários, exatamente como explicado na proposta 3 (vide seção 4.3.3). Entretanto, trocaremos a técnica de busca cega pelo módulo correto por um sistema de detecção preciso. Dessa forma, teremos apenas os benefícios daquela proposta.

Para realizar essa tarefa, modificaremos o arquivo de configuração da proposta 2 (vide seção 4.3.2) pelo seguinte [5]:

Listing 4.3: Ex. de arquivo de configuração contendo o mapeamento do ATR para módulo PKCS #11.

```
3BF2180002C10A31FE58C80874=moduloX
3BB71800C03E31FE6553504B3234900025=moduloY
```

Assim, a cada remoção e inserção de cartões, verificaremos se o ATR apresentado possui um módulo correspondente. Se sim, ele é carregado, senão uma mensagem de

[5] Da mesma forma que o arquivo anterior, este também deve tomar as medidas de segurança cabíveis assinado.

cartão não suportado é lançada. Com o módulo carregado, o código PKCS #11 será utilizado, abstraindo todas as peculiaridades e APDUs distintas de cada cartão para uma interface única.

Além de todas as vantagens dessa abordagem, ela ainda resume o suporte a novos cartões a duas únicas ações: adicionar uma nova linha no arquivo de configuração e incluir o módulo PKCS #11 correspondente no sistema de arquivos.

Capítulo 5

Implementações

Este capítulo descreve as implementações feitas para atender à proposta adotada na seção 4.3.4. Uma vez que o ASI-HSM é implementado inteiramente utilizando a linguagem de programação *C*, as implementações deste trabalham também a utilizam. As explicações seguirão o fluxo padrão de utilização de um Smart Card, visando facilitar seu entendimento.

A figura 5.1 detalha o funcionamento do sistema. Explicações detalhadas sobre cada passo da figura serão dadas nas duas seções seguintes. Cada uma detalhará os métodos definidos nos dois *header files* implementados por este trabalho: *p11_module_manager.h* e *p11_util.h*.

5.1 Implementações Utilizando a UI do Resource Manager

A UI do Resource Manager é especificada de forma genérica no padrão PC/SC. Os sistemas Windows a implementaram, dando-lhe o nome de *winscard.h*. Esse *header file*, com algumas diferenças [1], também está disponível para o sistema operacional do ASI-HSM, o *FreeBSD*. O pacote *libpcsclite1* o fornece, juntamente

[1] Existem funções adicionais definidas na interface *winscard.h* para Windows. Porém, elas oferecem apenas facilidades extras, não sendo obrigatórias

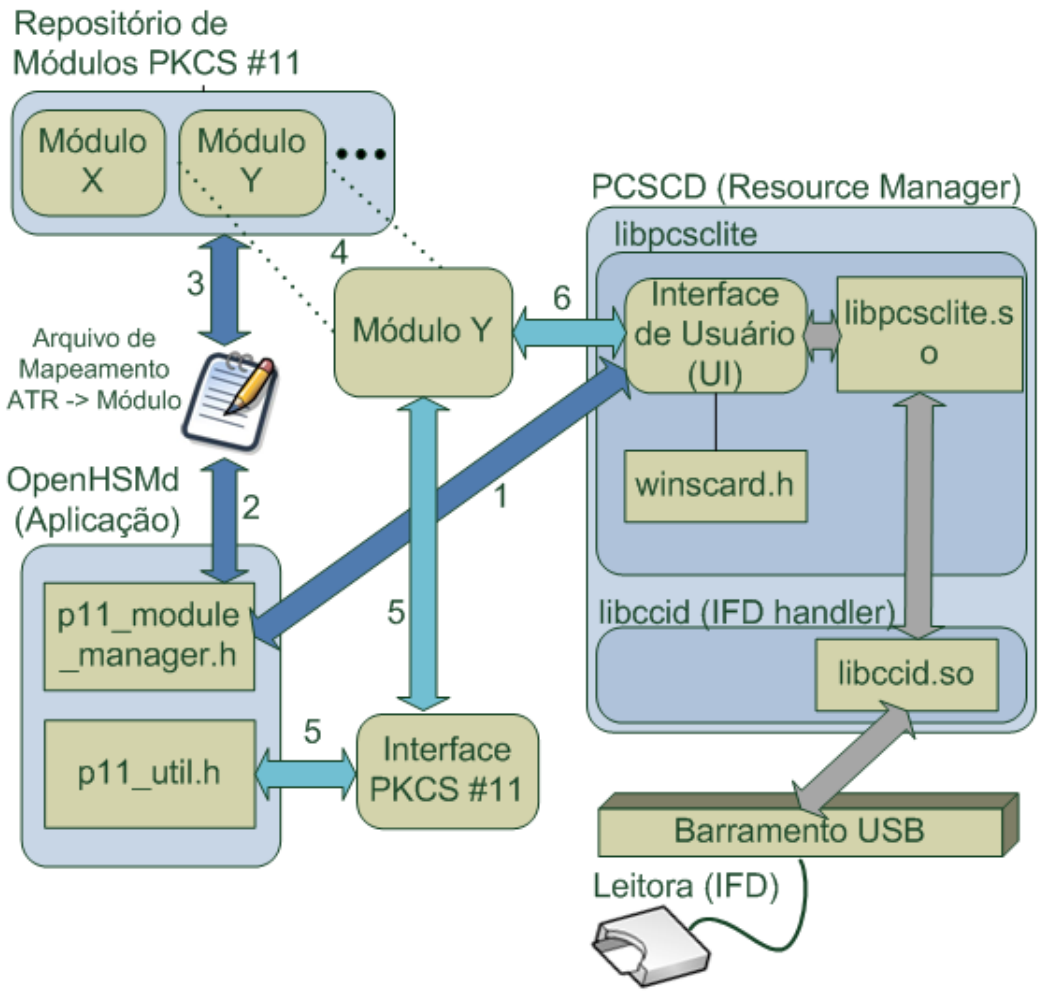

Figura 5.1: Esquema representando a solução implementada

com sua implementação, chamada de *libpcsclite.so*.

O arquivo *p11_module_manager.h* inclui o arquivo *winscard.h*, utilizando este último para as tarefas relacionadas à detecção de eventos (passo 1 da figura 5.1). Além disso, a arquivo *p11_module_manager.h* também é responsável por extrair o ATR do cartão conectado. Com o ATR em mãos, ele verifica, no arquivo de configuração, se existe módulo relacionado, ou seja, se o ASI-HSM o suporta (passo 2 da figura 5.1). Em caso afirmativo, é feita a carga dinâmica do módulo PKCS #11 correto, tornando-o disponível para uso futuro (passos 3 e 4 da figura 5.1). Os seguintes métodos foram definidos para suportar essa tarefa.

- *void P11_MM_manage_slot_events(void *ptr)*;
- *int P11_MM_check_ATR(char *ATR)*;
- *int P11_MM_load_pkcs11_module(char *module_path, int module_size)*;
- *int P11_MM_unload_pkcs11_module()*;

Seguindo o fluxo de execução, a primeira tarefa é criar uma *thread* específica para detectar eventos relacionados a cartões e leitoras. Essa *thread* chama a função *P11_MM_manage_slot_events*, a qual permanece em *loop* infinito, utilizando as funções de *callback* da UI para detectar eventos.

O arquivo *winscard.h* define diversos métodos e boa parte deles foram utilizados. Com o objetivo de tornar as explicações objetivas, contudo, apenas as funções diretamente relacionadas serão analisadas.

A função *SCardGetStatusChange* da UI é responsável pela notificação de mudança de estado do sistema. De maneira simplificada, ela bloqueia a execução por *n* segundos até que ocorra alguma mudança no estado de uma leitora (inserção ou remoção de Smart Cards ou da própria leitora). O valor de *n* é um parâmetro da função, podendo ser utilizado uma constante especial para bloquear por tempo indeterminado (*INFINITE*).

A função *P11_MM_manage_slot_events*, executada durante todo o funcionamento do sistema, possui o seguinte comportamento:

1. Espera por um evento de leitora, chamando a função *SCardGetStatusChange*;
2. Ao ser detectado a inserção de uma leitora, aloca-se a estrutura *SCARD_READER-STATE*. Essa estrutura contém dados sobre o estado da leitora, tais como seu nome, o ATR (em caso de haver cartão inserido), o estado atual da leitora, o tipo do último evento ocorrido, etc;
3. Chama-se novamente a função *SCardGetStatusChange* agora passando a recém

alocada estrutura, a fim de trabalhar somente com essa leitora [2]. Essa função irá bloquear a execução até que algum dos atributos da estrutura passada seja modificado, indicando a ocorrência de algum evento (e qual foi esse evento).

4. Ao ser detectado um evento de cartão, é verificado se ocorreu uma inserção ou remoção. Neste caso, volta-se para o passo 3. Naquele, por outro lado, chama-se a função *P11_MM_check_ATR*, responsável por extrair o ATR, realizar seu *parsing* e consultá-lo no arquivo de configuração. Caso a consulta mapeie para um módulo válido, verificar se o módulo já não está carregado. Se não estiver, chama-se a função *P11_MM_load_pkcs11_module* para carregá-lo (descarregando possíveis módulos previamente carregados, através da função *P11_MM_unload_pkcs11_module)*. Ao final da operação, retornar para o passo 3, esperando por novo evento.

Os items 1 e 2 ocorrem no início da execução do *OpenHSMd* não sendo necessário executá-los novamente, uma vez que o ASI-HSM possui uma leitora fixa.

5.2 Implementações Utilizando a Interface PKCS #11

As implementações utilizando a interface PKCS #11 beneficiam-se diretamente do módulo carregado anteriormente. Sob a ótica de sua implementação, ocorre uma abstração total do modelo de cartão inserido na leitora. Implementações nesse nível fornecem, portanto, uma camada capaz de trabalhar com todos os cartões suportados da mesma maneira.

Para ser utilizado pelo OpenHSMd, um cartão é primeiramente formatado e inicializado. Em seguida, seu perfil é criado e suas credenciais criptográficas são geradas. Por fim, ele é utilizado repetidas vezes para autenticação. Essas três etapas serão dividias em três subseções, com o intuito de melhor explicar cada fase da processo do ciclo de vida de um cartão, assim como apontar os benefícios dessa implementação.

[2] É possível trabalhar com várias leitoras simultaneamente, desde que cada uma possua sua própria estrutura *SCARD_READERSTATE*

5.2.1 Inicialização dos Cartões e Criação de Perfis

A inicialização de um cartão consiste em formatá-lo e torná-lo pronto para uso. A interface PKCS #11 fornece os métodos *C_InitToken* e *C_InitPin* para essa tarefa.

O método *C_InitToken* pode ser utilizado tanto para inicializar, como para reinicializar um cartão. Um dos parâmetros dessa função deve ser alimentado com uma sequência de caracteres, conhecida como Personal Identification Number (PIN). Se essa função estiver sendo chamada pela primeira vez, como ocorre com cartões novos de fábrica, o PIN fornecido tornar-se-á o Security Officer (SO) PIN [3]. Chamadas subsequentes, caracterizando a reinicialização do Token, deverão alimentar o SO PIN correto ou um erro é lançado.

A partir do momento em que o token é inicializado (ou reinicializado), o cartão está pronto para receber um perfil de usuário. No contexto do ASI-HSM um cartão, e consequentemente o perfil do usuário, é identificado através de seu número de série [4]. Antes de executar qualquer operação, portanto, é feita uma busca pelo número de série com o objetivo de identificar o perfil no banco de dados. A versão anterior a esse trabalho, por não utilizar a interface PKCS #11 e pelos utilitários do OpenSC não fornecerem uma maneira de obter o número de série real do cartão, utilizavam números de série virtuais [5], definidos pelo usuário na inicialização de cada cartão. Isso gerava problemas de ambiguidade, pois nada impedia o usuário de definir o mesmo número de série para cartões com funções diferentes, resultando em falhas de autenticação.

A criação do perfil, portanto, é feita em duas etapas. Primeiro, é chamada a função *C_InitPin* para inicializar o PIN de usuário. Em seguida, a função *C_GetTokenInfo*

[3] A interface PKCS #11 define dois tipos de PINs: O SO PIN e o PIN de usuário. O SO pin é utilizado em funções de administração do Token como, por exemplo, sua reinicialização. Já o PIN de usuário (User PIN) é utilizado para uso das funcionalidades do cartão.

[4] O número de série não deve ser confundido com o ATR. Este, identifica um modelo ou uma família de cartões. Aquele, por sua vez, representa um identificador único.

[5] O OpenSC exige que os usuários declarem um número de série em sua inicialização, caso contrário nenhum é utilizado.

é chamada para obter informações sobre o Token em questão (o Smart Card). Dentre essas informações, é possível capturar o número de série e associá-lo ao perfil sendo criado.

Nenhuma nova função foi definida para esse processo. As implementações foram incorporadas a funções já existentes do OpenHSMd.

5.2.2 Geração de Credenciais Criptográficas

A geração de credenciais criptográficas consiste em criar o material criptográfico necessário para que determinado usuário possa se autenticar de maneira segura perante o ASI-HSM.

Antes de iniciar sua explicação, é necessário entender um pouco melhor como a interface PKCS #11 trabalha. Apesar de ser escrita em linguagem *C*, uma linguagem estruturada, a Cryptoki ataca os problemas de maneira orientada a objetos. Para isso, cada tipo de credencial criptográfica, como chaves e certificados, possuem um identificador de classe e um identificador de tipo. Além disso, cada objeto possui diversos atributos, que podem ser utilizados para determinar o comportamento do mesmo.

A maneira que a interface PKCS #11 encontrou para simular um comportamento orientado a objetos foi através de *templates*. *templates* nada mais são que conjuntos de dados definidos em uma *struct*, semelhante a uma classe. O exemplo de código a seguir mostra todos os passos necessários para a geração de um par de chaves RSA, utilizando seus respectivos *templates*.

Listing 5.1: Exemplo de código fonte PKCS #11: Geração de par de chaves.

```
...
CK_RV rv;
CK_SLOT_ID first_slot = ...;
CK_UTF8CHAR user_pin = ...;
CK_SESSION_HANDLE session_handle;
CK_RV rv;
```

```
CK_MECHANISM mechanism = {CKM_RSA_PKCS_KEY_PAIR_GEN, NULL_PTR, 0};
CK_ULONG modulusBits = 1024;
CK_BBOOL true = CK_TRUE;
CK_BBOOL false = CK_FALSE;
CK_OBJECT_HANDLE pub_key_handle;
CK_OBJECT_HANDLE priv_key_handle;
CK_OBJECT_CLASS priv_key_class = CKO_PRIVATE_KEY;
CK_OBJECT_CLASS pub_key_class = CKO_PUBLIC_KEY;
CK_KEY_TYPE key_type = CKK_RSA;

// Template para chave pública.
CK_ATTRIBUTE publicKeyTemplate[] = {
   {CKA_CLASS, &pub_key_class, sizeof(pub_key_class)},
   {CKA_KEY_TYPE, &key_type, sizeof(key_type)},
   {CKA_LABEL, pub_key_label, strlen((char *)pub_key_label)},
   {CKA_TOKEN, &true, sizeof(true)},
   {CKA_ENCRYPT, &true, sizeof(true)},
   {CKA_VERIFY, &true, sizeof(true)},
   {CKA_WRAP, &true, sizeof(true)},
   {CKA_MODULUS_BITS, &modulusBits, sizeof(modulusBits)},
   {CKA_EXTRACTABLE, &true, sizeof(true)}
};

// Template para chave privada.
CK_ATTRIBUTE privateKeyTemplate[] = {
   {CKA_CLASS, &priv_key_class, sizeof(priv_key_class)},
   {CKA_KEY_TYPE, &key_type, sizeof(key_type)},
   {CKA_LABEL, priv_key_label, strlen((char *)priv_key_label)},
   {CKA_TOKEN, &true, sizeof(true)},
   {CKA_PRIVATE, &true, sizeof(true)},
   {CKA_SENSITIVE, &true, sizeof(true)},
```

```
    {CKA_DECRYPT, &true, sizeof(true)},
    {CKA_SIGN, &true, sizeof(true)},
    {CKA_UNWRAP, &true, sizeof(true)},
    {CKA_EXTRACTABLE, &false, sizeof(false)}
};

// Abrindo sessão de usuário.
if (( rv = C_OpenSession(first_slot,
                                    CKF_SERIAL_SESSION | CKF_RW_SESSION,
                                    NULL_PTR, NULL_PTR,
                                    &session_handle)) != CKR_OK){
    do_error_handling(rv); // erro
}

// Logando no token.
if ((rv = C_Login(session_handle, CKU_USER, user_pin,
                            strlen((const char *)userPin))) != CKR_OK){
    do_error_handling(rv); // erro
}

// Gerando o par de chaves
if ((rv = C_GenerateKeyPair(session_handle, &mechanism, publicKeyTemplate, 9,
                                        privateKeyTemplate, 10, &pub_key_handle,
                                        &priv_key_handle)) != CKR_OK){
    do_error_handling(rv); // erro
}

// Deslogando do token.
if ((rv = C_Logout(session_handle)) != CKR_OK){
    do_error_handling(rv); // erro
}
```

```
// Fechando sessão de usuário
if ((rv = C_CloseSession(session_handle)) != CKR_OK){
   do_error_handling(rv); // erro
}
...
```

Vamos tomar o exemplo do *template* para chaves privadas, denominado de *privateKeyTemplate[]* e explicar alguns de seus campos:

- *CKA_CLASS*: Representa a classe do objeto. O atributo *CKO_PRIVATE_KEY* significa que é uma classe de chaves privadas;
- *CKA_KEY_TYPE*: Representa o tipo da chave. O atributo *CKK_RSA* significa uma chave do tipo RSA;
- *CKA_TOKEN*: Atributo responsável pela persistência do objeto. Se definido como *CK_TRUE*, será gravado na memória não volátil do cartão. Em caso de receber o valor *CK_FALSE*, existirá somente enquanto a sessão permanecer aberta;
- *CKA_DECRYPT*: Define se o objeto pode ser utilizado para decifrar alguma informação;
- *CKA_PRIVATE*: Define se o objeto é privado. Objetos privados só podem ser acessados após o usuário ter logado com sucesso no Token;
- *CKA_SIGN*: Define se o objeto pode ser utilizado para realizar assinaturas digitais;
- *CKA_UNWRAP*: Define se o objeto pode ser utilizado para realizar unwrap (decifragem) de outra chaves;
- *CKA_EXTRACTABLE*: Define se o objeto é exportável do Token.

Esse exemplo também introduz o conceito de sessão. É sempre necessário abrir uma sessão para trabalhar via PKCS #11. Essa sessão pode ser de usuário, como

mostrado no exemplo anterior, ou de Security Officer (SO), para realizar tarefas administrativas (vide exemplo 5.1).

Após abrir uma sessão, é possível logar no token. Caso a sessão seja de usuário, deve-se logar utilizando o PIN de usuário. Se a sessão for de SO, o PIN de SO deve ser utilizado. Após logar no Token com sucesso, é possível ter acesso a objetos privados, como chaves privadas e/ou qualquer objeto que tenha o atributo *CKA_PRIVATE* definido como verdadeiro (*CK_TRUE*).

Sempre é necessário fechar uma sessão após ter finalizado as operação. Quando isso ocorre, todos os objetos criados com o atributo *CKA_TOKEN* com valor *CK_FALSE* são automaticamente destruídos. Essa propriedade nos permite chamar esses objetos de *objetos de sessão*, pois seu ciclo de vida é finalizado com o encerramento da sessão.

Com uma idéia melhor de como a interface PKCS #11 trabalha, é possível explicar as funções mais importantes da interface *p11_util.h*, implementada para este trabalho. Omitiremos os parâmetros e valores de retorno ao explicar as funções, com o objetivo de tornar as explicações mais agradáveis sem, contudo, prejudicar seu entendimento.

5.2.2.1 Função *P11_UTIL_store_private_key_secure*

Smart Cards são capazes de gerar números aleatórios, garantindo-lhes a habilidade de criar chaves criptográficas. Porém, pelo fato de o gerador de números aleatórios do ASI-HSM possuir um grau de confiança elevado, o par de chaves utilizado para autenticação deve ser gerado no interior do ASI-HSM e, em seguida, exportado para o cartão. Essa é uma tarefa delicada pois, como nota-se pela figura 4.1, o fio que liga a leitora de Smart Cards ao ASI-HSM não se encontra no interior do perímetro criptográfico. Um agente malicioso, escutando a comunicação entre as duas entidades, poderia ler a chave privada, comprometendo a segurança.

O objetivo deste método, portanto, é implementar um protocolo capaz de minimizar os riscos dessa operação. O protocolo baseia-se nas seguintes funções da

interface PKCS #11.

- *C_Wrap*: Processo em que uma chave é cifrada com outra chave. Não existem restrições sobre as possíveis combinações de Criptografia Simétrica e/ou Assimétrica. Por exemplo, é possível cifrar uma chave simétrica DES com um chave assimétrica RSA. Existe, contudo, uma exceção à regra: chaves públicas RSA só podem ser utilizadas para cifrar (*Wrap*) outras chaves, enquanto que chaves privadas RSA podem apenas ser utilizadas para decifrar (*Unwrap*).

- *C_Unwrap*: Processo em que um chave, previamente cifrada (*Wrapped*), é decifrada.

Tanto o processo de *Wrapping* como de *Unwrapping* ocorrem no interior do Token criptográfico, garantindo a proteção do material sensível, uma vez que ele nunca é exposto. A implementações desse protocolo, assim como diversas outras funções implementadas, utilizaram também a biblioteca de código aberto *OpenSSL* [The 11], com o objetivo de simular os processos de *Wrapping* e *Unwrapping* no interior do ASI-HSM, assim como realizar qualquer tarefa relacionada à criptografia [6].

Os objetos de sessão possuem um papel fundamental no desenvolvimento deste protocolo pois, uma vez que não são escritos na memória não volátil do cartão, eles são criados e destruídos rapidamente.

A figura 5.2 ilustra o protocolo desenvolvido. Após gerar, no interior do ASI-HSM, o par de chaves $\{Kr^h, Ku^h\}$ que será utilizado para autenticar o usuário, o protocolo tem início. Primeiramente, é solicitado ao cartão que gere um par de chaves de sessão $\{Kr^c, Ku^c\}$, ou seja, um par de chaves com seu atributo *CKA_TOKEN* = *CK_FALSE* (passos 1 e 2 da figura 5.2).

Em seguida, o cartão exporta a chave pública Ku^c recém gerada. Essa exportação é feita obtendo-se o módulo e o expoente da chave pública, através da função *C_GetAttributeValue*, da interface PKCS #11. Com o módulo e o expoente em mãos, funções do OpenSSL são utilizadas para montar uma estrutura representando

[6] O ASI-HSM já utiliza largamente o *OpenSSL*, não havendo necessidade de incluir essa biblioteca.

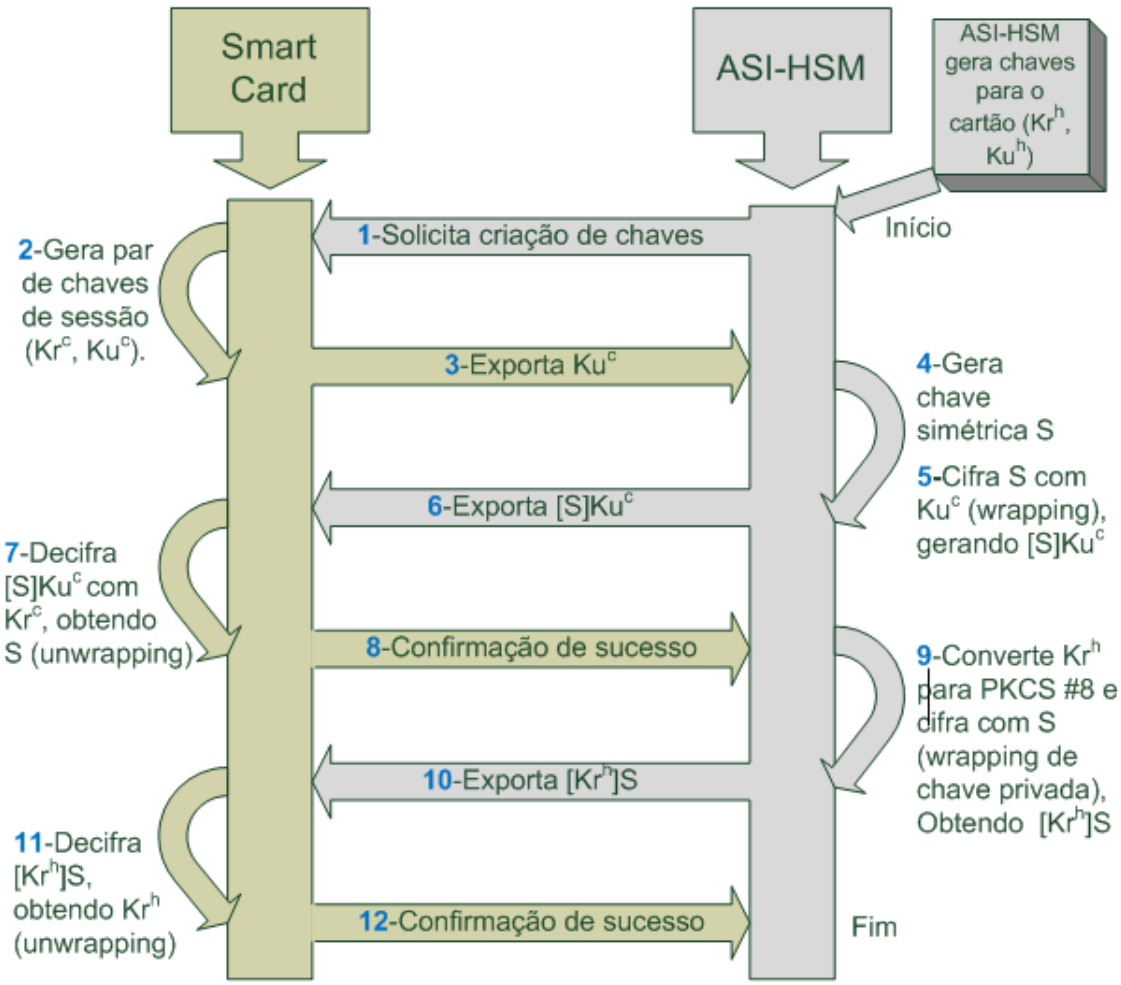

Figura 5.2: Protocolo de importação de chaves privadas para Smart Cards.

essa chave e, posteriormente, utilizá-la para cifrar (*Wrapping*) uma chave simétrica *S* também gerada no ASI-HSM (passos 3, 4 e 5 da figura 5.2).

A chave simétrica cifrada, denotada como $[S]Ku^c$, é então enviada para o cartão. Utilizando a função *C_Unwrap* e a chave privada Kr^c gerada anteriormente, o cartão é capaz de obter a chave simétrica sem que, em nenhum momento, essa chave tenha trafegado em claro. A partir desse momento, tanto o cartão como o ASI-HSM compartilham a mesma chave *S* (passos 6, 7 e 8 da figura 5.2).

Ao receber a notificação de sucesso do cartão, o ASI-HSM pode iniciar o processo de *Wrapping* de chaves privadas. Para o sucesso desse processo, é necessário

converter a chave privada em um formato específico, denominado PKCS #8 [RSA 11c]. Funções do OpenSSL foram utilizadas para essa conversão. Tendo sido convertida para o formato esperado, a chave Kr^h é cifrada com a chave de sessão S, gerando $[Kr^h]S$ (passo 9 da figura 5.2).

Por fim o cartão, fazendo uso da função *C_Unwrap*, é capaz de decifrar $[Kr^h]S$ utilizando S obtendo, finalmente, Kr^h. O protocolo termina com o cartão enviando a mensagem de sucesso ao decifrar (*Unwrap*) a chave privada Kr^h.

Como tanto o par de chaves $\{Kr^c, Ku^c\}$ como a chave simétrica S são criados, no Smart Card, com seus atributos *CKA_TOKEN* definidos como *CK_FALSE*, eles são automaticamente destruídos ao final da operação. Essa destruição ocorre pois a sessão é fechada.

5.2.2.2 Função *P11_UTIL_store_public_key*

Essa operação é relativamente mais simples que a anterior, uma vez que a chave pública pode ser enviada em claro. A implementação dessa função, portanto, utiliza a biblioteca OpenSSL para decodificar a chave pública, obtendo seu módulo e expoente e, então, utiliza a função *C_CreateObject* para criar o novo objeto no cartão. O *template* para a criação dessa chave assemelha-se ao do exemplo de código mostrado no exemplo 5.1. Temos agora, no entanto, que fornecer os valores do módulo e do expoente, através dos atributos *CKA_MODULUS* e *CKA_PUBLIC_EXPONENT*.

5.2.2.3 Função *P11_UTIL_store_certificate*

Ocorre de maneira semelhante à exportação da chave pública. Entretanto, desta vez é necessário decodificar o Nome do Titular (Subject), o Nome do Emitente (Issuer) e o Número Serial (Serial Number). Em termos de atributos PKCS #11, eles correspondem ao *CKA_SUBJECT*, *CKA_ISSUER* e *CKA_SERIAL_NUMBER*, respectivamente.

5.2.2.4 Função *P11_UTIL_key_pair_gen*

Esta função solicita ao cartão a criação de um par de chaves. Seu conteúdo é semelhante ao mostrado no exemplo 5.1.

5.2.3 Utilização do Cartão

A utilização do cartão é definida como o momento em que um usuário se autentica perante o ASI-HSM, a fim de realizar determinadas tarefas cabíveis a ele. Primeiramente é necessário obter a chave privada previamente importada utilizando o protocolo descrito na seção 5.2.2.1. Essa tarefa é implementada na função *P11_UTIL_get_private_key* (vide seção 5.2.4.4).

Com um identificador da chave privada em mãos [7], é possível utilizá-la para a prova de posse, proposta pelo ASI-HSM.

5.2.3.1 Função *P11_UTIL_decipher*

Função responsável por realizar toda e qualquer desafio proposto pelo ASI-HSM, respondendo de forma apropriada.

5.2.4 Outras Funções Importantes

Algumas funções são inerentes a todas as fases (inicialização, geração de credenciais criptográficas e utilização) do uso de um Smart Card. Outras, por sua vez, encapsulam tarefas específicas de determinadas operações. As mais importantes são citadas a seguir.

5.2.4.1 Função *P11_UTIL_negotiate_session_key*

Função utilizada no protocolo de importação de chaves, com o intuito de encapsular os detalhes da negociação da chave simétrica.

[7] É importante ressaltar que todas as funções que buscam por objetos privados retornam apenas um identificador para o mesmo. O objeto jamais deixa a segurança de seu contêiner.

5.2.4.2 Função *P11_UTIL_priv_key2pkcs8*

Também utilizada no protocolo de importação de chaves, ela encapsula a conversão de uma chave privada para o formato PKCS #8.

5.2.4.3 Função *P11_UTIL_get_error_code*

Um extenso código de erros é definido na interface PKCS #11. Essa função recebe um argumento do tipo CK_RV [8] e retorna um *string* contendo a descrição de seu valor. Geralmente essa função é chamada quando algum erro ocorre, para informar ao usuário a natureza do erro ou para escrevê-lo em um arquivo de *logs*.

5.2.4.4 Função *P11_UTIL_get_private_key*

Essa função busca por determinada chave privada em algum Smart Card e retorna um identificador para a mesma. É utilizada múltiplas chamadas para a função *C_GetAttributeValue*, da interface PKCS #11, com a classe de objetos *CKA_CLASS* definida como *CKO_PRIVATE_KEY* (chaves privadas).

5.3 Compatibilidade Retroativa

A compatibilidade retroativa foi implementada. Cartões já inicializados com o OpenSC podem agora se beneficiar do uso do módulo PKCS #11 disponibilizado pelo OpenSC (*opensc-pkcs11.so*). Dessa forma, mesmo esses cartões se beneficiarão de códigos de erros precisos. Além disso ocorrerá a remoção de boa parte dos comandos *system* eliminando, portanto, possíveis vulnerabilidades de código.

Apesar de poderem ser utilizados da mesma forma, cartões que, por algum motivo, não possuirem um módulo PKCS #11 próprio e necessitarem ser inicializados pelo OpenSC, terão sua inicialização feita da maneira antiga, através de comandos *system*. Para definir um modelo de cartão suportado apenas pelo OpenSC, basta especi-

[8] RV significa return value, ou seja, representa o valor do retorno de determinada operação.

ficar, no arquivo de configuração, a seguinte entrada (substituindo ATR_CARTAO_OPENSC pelo ATR desejado):

Listing 5.2: Ex. de mapeamento de um módulo suportado apenas pelo OpenSC

```
ATR_CARTAO_OPENSC=opensc-pkcs11.so
```

5.4 Resultados Obtidos

Todos os resultados citados abaixo foram testados e mostraram-se funcionais. Será resumido como sua implementação foi feita.

Basicamente, o esquema adotado forneceu suporte para resolver todos os objetivos propostos na seção 1.3. Utilizando um repositório de módulos PKCS #11, juntamente com um sistema de detecção de modelos de cartões preciso, foi possível utilizar o mesmo código para qualquer cartão sendo utilizado. Para isso, bastou realizar a carga dinâmica do módulo correto e utilizá-lo através de código PKCS #11 nativo. Essa abordagem resolve implicitamente o problema de múltiplos cartões suportados, exposto na seção 4.2.3.

Já a interface PKCS #11, por sua vez, fornece diversos mecanismos para trabalhar com Smart Cards, possibilitando a implementação de soluções para os problemas expostos na seção 4.2. Em especial, o protocolo de importação de chaves seguro foi desenvolvido utilizando seus mecanismos.

Capítulo 6

Considerações Finais

Em um contexto geral trabalhar profundamente com Smart Cards, entendendo todo o caminho percorrido desde uma aplicação até a interpretação do comando pelo microprocessador do Smart Card não é uma tarefa trivial. Existe pouca documentação sobre o assunto, muitas vezes restando apenas a especificação técnica dos padrões vigentes que, apesar de serem os documentos mais completos e ricos sobre o conteúdo, possuem uma curva elevada de aprendizado, pelo seu conteúdo altamente técnico. Além disso, diferentes padrões surgiram em diferentes contextos como, por exemplo, a interface PKCS #11 e o padrão PC/SC, mas suas implementações acabaram por coexistir. Tendo isso em mente, este trabalho é útil como um guia para quem deseja trabalhar com Smart Cards, pois buscou mostrar como todos os padrões foram concebidos, implementados e, posteriormente, como acabaram por interagir.

Em um contexto específico, o trabalho desenvolveu soluções para diversos problemas relativos ao uso de Smart Cards. O ASI-HSM agora possui uma solução genérica, baseada em padrões bem estabelecidos e livre de qualquer contexto. Portanto, ele é beneficiado diretamente por tudo que esses padrões tem a oferecer, evitando sua amarração a determinada tecnologia. Adicionalmente, qualquer cartão que forneça um módulo PKCS #11 compatível pode, agora, ser incorporado facilmente, tornando o ASI-HSM livre para escolher que cartões deseja suportar.

6.1 Trabalhos Futuros

Atualmente, a autenticação de um grupo de usuários no ASI-HSM é feita utilizando segredo compartilhado. Cada pedaço desse segredo é cifrado com o material criptográfico contido em um determinado Smart Card. Durante a criação do grupo de usuários, é definido quantos usuários *m* de um total de *n* são necessários para juntar novamente o segredo.

Dentro desse contexto, um problema surge: o que acontece se um membro desse grupo esquecer ou bloquear seu PIN? A resposta trivial para esse problema seria regerar todos os pedaços do segredo compartilhado e cifrá-los novamente com todos os Smart Cards necessários. Obviamente essa solução é problemática uma vez que exige a presença de todos os membros do grupo. Além disso, tarefas críticas como criação de grupo de usuários devem ser evitadas ao máximo, pois possuem custos envolvidos [1].

A solução para esse problema já é possível com o modelo atual do ASI-HSM [2]. Porém, utilizando o modelo adotado para este trabalho é possível simplificar a solução.

Com o advento do uso da interface PKCS #11, o conceito de SO PIN e de User PIN ficaram claros. O SO PIN é utilizado, dentre outras tarefas, para desbloquear e/ou reinicializar o PIN de usuário [3]. Utilizando esses conceitos, poderíamos propor uma solução em que seria utilizado o mesmo SO PIN para todos os membros do grupo. O SO PIN seria gerado aleatoriamente no interior do ASI-HSM e cifrado com o mesmo segredo compartilhado utilizado na autenticação. Dessa forma, o usuário estaria automaticamente vinculado ao ASI-HSM e, caso perdesse acesso ao seu User PIN, os próprios membros do grupo poderiam aprovar o desbloqueio do cartão.

Essa solução também permite desvincular o usuário do ASI-HSM com a aprovação dos demais. Bastaria modificar o SO PIN para um de escolha do usuário e

[1] Esses custos envolvem, dentre outros fatores, cerimônias, auditorias e viagens

[2] O problema do desperdício de cartões, contudo, permanece.

[3] Por esse motivo, o SO PIN muitas vezes é referido como PUK

cifrar o pedaço do segredo compartilhado com o Smart Card de um novo membro.

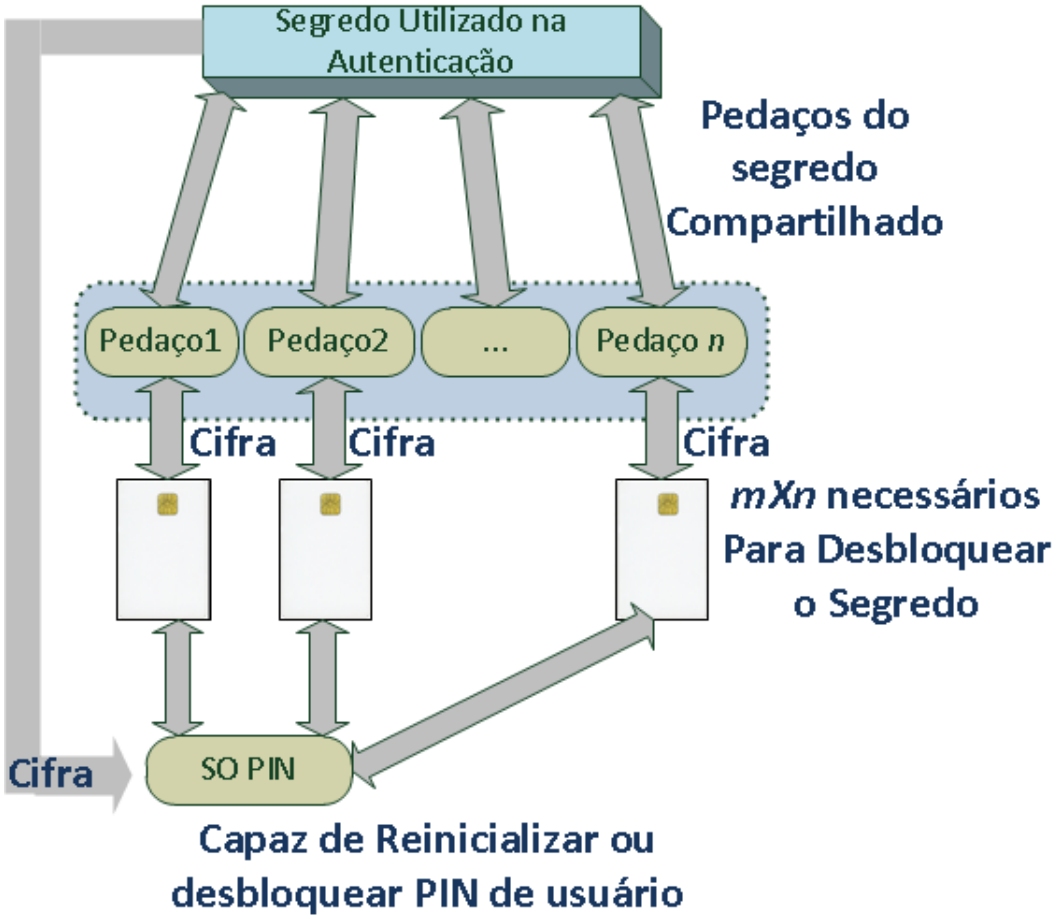

Figura 6.1: Esquema de vinculação de usuários ao ASI-HSM

Outro trabalho futuro é avaliar a adequação do esquema adotado com o *MCT-7*. Essa avaliação está fora do escopo deste trabalho, mas é importante na medida em que os módulos PKCS #11 propritários estarão dentro do ASI-HSM e, portanto, sujeitos às implicações dessa norma.

Referências

[BER 09] BEREZA, J. A. Aprimoramento de um hsm para homologação na icp-brasil. Universidade Federal de Santa Catarina, 2009. Trabalho de conclusão de curso (graduação em ciências da computação).

[Dig 11] Digital Security Group, Radboud University Nijmegen. **Security Flaw in Mifare Classic**. Disponível em <http://www.ru.nl/ds/research/rfid/>. Acesso em: 23 de maio de 2011.

[FED 11] FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. **SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES**. Disponível em <http://csrc.nist.gov/publications/fips/fips1402/fips1402.pdf>. Acesso em: 23 de maio de 2011.

[htt 11] http://www.cplusplus.com. **system call**. Disponível em <http://www.cplusplus.com/reference/clibrary/cstdlib/system/>. Acesso em: 23 de maio de 2011.

[Imp 11] Imprensa Nacional. **Diário Oficial da União**. Disponível em <http://www.in.gov.br/imprensa/visualiza/index.jsp?jornal=1&pagina=2&data=03/05/2011>. Acesso em: 23 de maio de 2011.

[Ins 11] Instituto Nacional de Tecnologia da Informação. **Manual de Condutas Técnicas 7**. www.iti.gov.br/twiki/pub/Homologacao/Documentos/MCT7__Vol.I.pdf.

[JK 07] JONATHAN KATZ, Y. L. **Introduction to Modern Cryptography: Principles and Protocols**. Chapman & Hall/CRC, 2007.

[Lab 11] LabSEC. **ASI-HSM**. https://projetos.labsec.ufsc.br/openhsmd.

[MEN 97] MENEZES, A. J. **Handbook of Applied Cryptography (Discrete Mathematics and Its Applications)**. CRC Press, 1997.

[NK 07] NOHL KARSTEN, H. P. **Mifare Little Security, Despite Obscurity**. Disponível em <http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html>. Acesso em: 4 de dezembro de 2009.

[NK 08] NOHL KARSTEN, H. P. **Reverse-Engineering a Cryptographic RFID Tag**. Disponível em <http://www.usenix.org/events/sec08/tech/nohl.html>. Acesso em: 23 de maio de 2011.

[Ope 11] OpenSC project. **opensc-project.org Home of open source smart card solutions**. Disponível em <http://www.opensc-project.org/>. Acesso em: 23 de maio de 2011.

[PC/ 05a] PC/SC Workgroup. **Interoperability Specification for ICCs and Personal Computer Systems, part 1: Introduction and Architecture Overview**. Disponível em <http://www.pcscworkgroup.com/specifications/files/pcsc1_v2.01.01.pdf>. Acesso em: 23 de maio de 2011.

[PC/ 05b] PC/SC Workgroup. **Interoperability Specification for ICCs and Personal Computer Systems, part 2: Interface Requirements for Compatible IC Cards and Readers**. Disponível em <http://www.pcscworkgroup.com/specifications/files/pcsc2_v2.01.01.pdf>. Acesso em: 23 de maio de 2011.

[PC/ 07] PC/SC Workgroup. **Interoperability Specification for ICCs and Personal Computer Systems Part 3: Requirements for PC-Connected Interface Devices**. Disponível em <http://www.pcscworkgroup.com/specifications/files/pcsc3_v2.01.09.pdf>. Acesso em: 23 de maio de 2011.

[PC/ 11] PC/SC Workgroup. **Introduction and Architecture Overview**. Disponível em <http://www.pcscworkgroup.com/specifications/files/pcsc6_v2.01.01.pdf>. Acesso em: 23 de maio de 2011.

[RAN 97] RANKL, W. **Smart Card Applications: Design models for using and programming smart cards**. Wiley, 1997.

[rfc 11] rfc2631. **Diffie-Hellman Key Agreement Method**. Disponível em <http://tools.ietf.org/html/rfc2631>. Acesso em: 23 de maio de 2011.

[Rob 11] Robert N. M. Watson. **Exploiting Concurrency Vulnerabilities in System Call Wrappers7**. Disponível em <http://www.watson.org/ robert/2007woot/2007usenixwootexploitingconcurrency.pdf>. Acesso em: 23 de maio de 2011.

[RSA 11a] RSA Laboratories. **PKCS #11 v2.20: Cryptographic Token Interface Standard**. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf.

[RSA 11b] RSA Laboratories. **PKCS #11 v2.20: Cryptographic Token Interface Standard**. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-15/pkcs-15v1_1.pdf.

[RSA 11c] RSA Laboratories. **PKCS #8: PrivateKey Information Syntax Standard**. ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs8.asc.

[The 11] The OpenSSL Project. **OpenSSL: The Open Source toolkit for SSL/TLS**. Disponível em <http://www.openssl.org/>. Acesso em: 23 de maio de 2011.