Universidade de Brasília
IE – Departamento de Ciência da Computação
Estudos em Codificação e Criptografia – 2º/2006
Orientador: Professor Pedro Antônio Dourado de Rezende

Smartcards – Operando em Baixo Nível

Leonardo Pignataro

Motivação

Este documento é fruto de uma pesquisa feita sobre smartcards com o objetivo de desenvolver aplicações rodando em dispositivos portáteis de pagamento, ou POS (*point-of-sale*), que operem com smartcards com funcionalidades criptográficas. Um exemplo típico de POS é o Nurit Lipman 8320, mostrado ao lado.

Os recursos computacionais em tais dispositivos são consideravelmente limitados, em especial no que se refere à memória disponível (em geral da ordem de 1MB). Por isso, o uso de bibliotecas prontas para operar com smartcards fica praticamente impossibilitado, pois o uso de bibliotecas aumenta muito o tamanho do código compilado, que ocupa a memória. Além disso, haveria o esforço de portar tais bibliotecas para o dispositivo, trabalho nada trivial, principalmente se considerarmos que o software sendo desenvolvido deveria ser portável entre dispositivos com arquiteturas bem diferentes. Ou seja, não bastaria portar a biblioteca para um dispositivo específico, seria necessário modificá-la de forma a torná-la facilmente portável entre dispositivos.

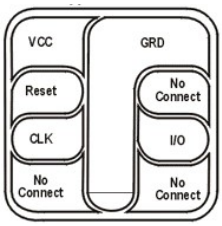
Por essas razões, fez-se necessário uma pesquisa sobre como operar com smartcards diretamente, em baixo nível. É importante esclarecer que nível é este, pois por "baixo nível" pode-se entender envio de sinais elétricos. Todos os dispositivos com os quais pudemos trabalhar disponibilizam uma API para o programador que o permite abstrair-se dos detalhes de comunicação com o smartcard, podendo trabalhar enviando e recebendo bytes – mais especificamente APDUs, como será esclarecido mais adiante.

Introdução

Um smartcard, ou ICC (*integrated circuits card*), pode ser definido simplesmente como um cartão portátil com circuitos integrados imbutidos. Entretanto, no contexto deste documento, um smartcard é um cartão, tipicamente do tamanho de um cartão de crédito, que possui um chip imbutido com capacidaedes criptográficas.

Um exemplo típico é o e-CPF, mostrado ao lado. Este smartcard tipicamente possui um par de chaves associado a uma pessoa física, com o qual pode-se assinar digitalmente documentos, tendo tal assinatura fé pública conferida pela medida provisória 2.200-2, de 4 de agosto de 2001, desde que observadas as premissas envolvidas (essa questão foge ao escopo deste documento).

Embora o formato dos chips pareçam variar de cartão para cartão, o importante nos chips são os 8 pontos de contato elétrico, cada um com uma localização específica e uma função específica (um para corrente, um para "terra", um para E/S, um para fornecer o clock segundo o qual o processador do chip irá operar, um para enviar o sinal de "reset", e três reservados para uso futuro).



A operação de um smartcard é baseada em camadas de abstração. No nível mais baixo estão os sinais elétricos transmitidos por esses conectores. Logo acima, definem-se protocolos de transmissão de dados, e, utilizando estes protocolos, comandos para operar o cartão. Existe também uma organização lógica da memória do cartão. Todas essas camadas são padronizadas pelo padrão ISO 7816. Este padrão é público, porém é preciso pagar para ter acesso ao mesmo. Alternativamente, é possível obter os *drafts* (rascunhos) que circularam pela internet enquanto o padrão está em desenvolvimento. Entretanto, esses documentos não são facilmente encontrados.

Caso se deseje operar com assinaturas digitais, é preciso acessar os certificados armazenados no smartcard. Para isso, recorre-se ao padrão PKCS#15, que define uma forma de armazenar certificados em smartcards. Esse padrão, desenvolvido pela RSA Security, é público e gratuito, podendo ser baixado no site da empresa.

Falaremos em seguida sobre esses 2 padrões, e então daremos um exemplo prático de operação do smartcard, passando então à situação de padronização e interoperabilidade em que se encontra a ICP Brasil.

ISO 7816-3

A parte 3 do padrão ISO 7816 cobre os sinais eletrônicos e protocolos de transmissão, assuntos que de uma forma geral estão em um nível mais baixo do que o que se pretende atingir nesta pesquisa. Entretanto, o documento trata de 2 assuntos muito importantes para o nosso contexto: a especificação da string ATR e definição de pares comando-resposta (APDUs).

Answer To Reset

A operação de um smartcard se inicia com o envio de um sinal de *reset* para o cartão. O cartão então responde com um stream de bytes, o ATR (*answer-to-reset*), que tem no máximo 33 bytes. Esta string contém parâmetros que especificam como a comunicação deve ocorrer, a nível físico (protocolo de transmissão, velocidade de transmissão, etc). Além disso, o ATR contém uma string denominada *history bytes*, que identifica o modelo do smartcard, bem como algumas de suas capacidades.

Com essas informações, é possível tomar decisões sobre quais mecanismos utilizar para operar com o cartão. A biblioteca OpenSC, por exemplo, possui um arquivo .c para cada modelo de smartcard. Cada um desses arquivos contém funções que fazem as mesmas operações, porém de formas específicas para cada cartão.

APDUs

Um passo na comunicação entre a aplicação e o cartão consiste no envio de um comando, processamento no cartão, e envio de uma resposta do cartão para a aplicação. O padrão ISO 7816 especifica que para cada comando deve haver uma resposta, e cada par comando-resposta deve ser finalizado antes de se iniciar o próximo par. Um APDU (*application protocol data unit*) é uma mensagem que representa um comando, ou uma resposta do cartão.

Um APDU de commando consiste de 2 partes: um header obrigatório de 4 bytes, contendo os bytes Class, Instruction, P1 e P2 (CLA, INS, P1, P2), e um campo opcional denominado *command body,* que pode conter o campo Lc, que denota o tamanho do campo de dados do comando, o campo de dados do comando, e o campo Le, que denota o tamanho máximo esperado da resposta.

CLA – INS – P1 – P2	[Lc] – [dados] – [Le]

O byte Class traz informações sobre como o APDU deve ser enviado. O byte Instruction codifica uma instrução, um comando. Os bytes P1 e P2 geralmente são parâmetros do comando, mas podem ser usados para especificar um subcomando.

Um APDU de resposta consiste também de 2 partes: um campo de dados opcional, de tamanho menor ou igual a Le, e dois bytes, SW1 e SW2, que denotam o resultado do comando.

[dados]	SW1 – SW2

Existe também o campo Lr, que denota o tamanho real do campo de dados da resposta.

Como os campos de dados tanto no comando quanto na resposta são opcionais, existem 4 casos: (1) comando e resposta sem campo de dados; (2) comando sem campo de dados, resposta com; (3) comando com campo de dados, resposta sem; (4) comando e resposta com campo de dados. A codificação do APDU do comando para cada um dos casos está ilustrada na figura abaixo.

caso 1

CLA – INS – P1 – P2

caso 2

CLA – INS – P1 – P2	Le

caso 3

CLA – INS – P1 – P2	Lc	Dados

caso 4

CLA – INS – P1 – P2	Lc	Dados	Le

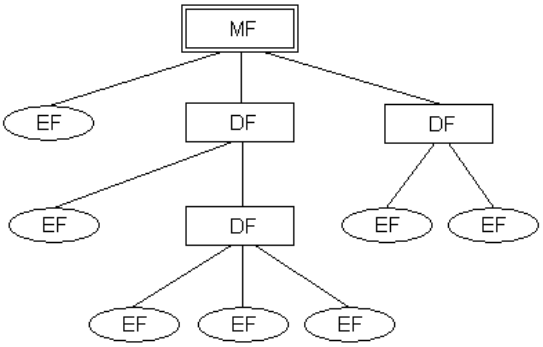
O padrão ISO 7816-3 explica em detalhes como deve ser a codificação e decodificação dos APDUs. Contudo, não entraremos aqui nesses detalhes.

ISO 7816-4

A parte 4 do padrão ISO 7816 define, entre outros, uma estruturação lógica da memória do smartcard e um conjunto de comandos para manipular esta estrutura. É importante ressaltar que a estruturação definida pelo padrão é apenas uma abstração, é como a aplicação vê a memória do cartão quando operando com APDUs. A questão de como a memória está organizada fisicamente está fora do escopo do padrão.

Estruturação lógica da memória

A memória é estruturada como um sistema de arquivos, onde existem 2 tipos de arquivos: os *dedicated files* (DFs) e os *elementary files* (EFs), sendo que EFs contém dados e DFs contém EFs. Fazendo um paralelo com um sistema de arquivos convencionais, os DFs seriam diretórios, e os EFs seriam arquivos. Os EFs contidos em um DF seriam seus "filhos", e o DF seria o "pai" dos EFs. Existe um DF especial, chamado *master file* (MF), que é a raiz do sistema, e sua presença é obrigatória.



Existem 2 tipos de EFs, os internos (*internal* EFs) e os de trabalho (*working* EFs). Os internos são utilizados pelo próprio cartão para armazenar dados internos, por exemplo estruturas de controle. Os de trabalho são EFs disponíveis para o mundo exterior, e não são interpretados pelo cartão.

Existe o conceito de canal lógico, que estabelece uma ligação com um DF, definindo aquele DF como o "DF atual" ou "DF selecionado". Um canal lógico pode ter uma ligação adicional a um EF do DF selecionado, definido-o como o "EF atual" ou "EF selecionado". Após o reset do cartão, implicitamente cria-se um canal lógico no qual o MF está selecionado. Podem existir múltiplos canais lógicos, porém para fins de simplificação, consideraremos apenas um canal lógico.

O padrão especifica que arquivos devem poder ser referenciados por pelo menos um dos seguintes métodos:

- Referência por ID de arquivo. Cada arquivo possui um ID, codificado em 2 bytes, e um ID referencia um arquivo sob o DF atual. Nesse esquema, todos os EFs e DFs filhos de um mesmo DF devem ter IDs diferentes para que possam ser referenciados sem ambigüidade. Os valores 3F00, 3FFF e FFFF são reservados (o primeiro é o ID do MF, o segundo é utilizado na referência por caminho, e o terceiro é reservado para uso futuro);
- Referência por caminho (*path*). Um caminho começa com o ID do MF ou do DF atual, e acaba com o ID do arquivo referenciado, tendo os IDs dos DFs intermediários no meio. Caso o ID do DF atual não seja conhecido, o valor 3FFF pode ser usado (assim como o "." em prompts de commando convencionais);
- Referência de EF *short identifier*. Um *short identifier* consiste em um número de 1 a 30 que referencia um EF dentro do DF atual. O número 0 referencia o EF atual;
- Referência de DF por nome. Cada DF pode ser referenciado por um nome, uma string de 1 a 16 bytes. Nesse esquema, para que cada DF possa ser referenciado sem ambigüidade, cada DF do cartão deve ter um nome único.

Assim, existem diferentes métodos para selecionar por exemplo o DF do PKCS#15:

- Selecionar o MF, depois selecionar o ID do arquivo;
- Selecionar o arquivo pelo seu nome;
- Selecionar o caminho 3F:00:XX:YY (onde XX:YY é o ID do arquivo).

Os EFs podem ter 2 estruturações:

- Transparente (transparent structure): o EF é visto como uma seqüência de unidades de dados, ou bytes;
- Em registros (*record structure*): o EF é visto como uma seqüência de registros acessíveis separadamente.

Os EFs do PKCS#15, por exemplo, são todos transparentes. EFs estruturados em registros podem ter seus registros de tamanho fixo ou variável, e os registros podem estar dispostos de forma linear ou cíclica.

Comandos

O padrão ISO 7816-4 define uma gama de comandos para manipular o sistema lógico de arquivos do cartão. Demonstraremos 2 comandos: o de seleção de arquivos (SELECT) e o de leitura de arquivos transparentes (READ BINARY).

Comando SELECT

Este comando altera o canal lógico, modificando o DF e/ou o EF selecionado. O comando é especificado pelo byte Instruction = 0xA4, e o byte P1 especifica como o arquivo será referenciado:

b8	***b7***	***b6***	***b5***	***b4***	***b3***	***b2***	***b1***	***significado***	***valor do campo de dados***
0	0	0	0	0	0	x	x	Referência por ID	
0	0	0	0	0	0	0	0	- selecionar MF, DF ou EF	ID, ou nulo
0	0	0	0	0	0	0	1	- selecionar DF filho do DF atual	ID do DF
0	0	0	0	0	0	1	0	- selecionar um EF filho do DF atual	ID do EF
0	0	0	0	0	0	1	1	- selecionar DF pai do DF atual	nulo
0	0	0	0	0	1	x	x	Referência por nome de DF	
0	0	0	0	0	1	0	0	- refência direta por nome	nome do DF
0	0	0	0	1	0	x	x	Referência por caminho	
0	0	0	0	1	0	0	0	- caminho a partir do MF	caminho (sem o ID do MF)
0	0	0	0	1	0	0	1	- caminho a partir do DF atual	Caminho (sem o ID do DF atual)

No caso de P1 = 0x00, o ID especificado deve ser único nos 2 ambientes seguintes: filhos do DF atual, e DFs "filhos" do DF "pai" do DF atual. Em outras palavras, com P1=0x00 pode-se implicitamente selecionar um EF filho do DF atual ou um DF "irmão" do DF atual (ou o próprio DF atual, não muito útil).

O padrão define uma estrutura chamada *file control information* (FCI), que é a resposta de um comando SELECT. Existem 3 formatos para essa estrutura, denominados FCP, FMD e FCI (*file*

control parameters, *file management data*, e o último é uma união dos 2 anteriores). O FCP contém informações como:

- Tamanho do arquivo
- ID do arquivo
- Tipo do arquivo
- Nome do arquivo, se for um DF
- Estrutura do arquivo, se for um EF

O byte P2 do comando SELECT especifica qual dos formatos de FCI retornar, além de especificar o comporatento no caso de a referência ao arquivo a ser selecionado ser ambígua:

b8	***b7***	***b6***	***b5***	***b4***	***b3***	***b2***	***b1***	***significado***
0	0	0	0	-	-	x	x	Qual arquivo selecionar em caso de ambigüidade
0	0	0	0	-	-	0	0	- Primeira ou única ocorrência
0	0	0	0	-	-	0	1	- Última ocorrência
0	0	0	0	-	-	1	0	- Próxima ocorrência
0	0	0	0	-	-	1	1	- Ocorrência anterior
0	0	0	0	x	x	-	-	Estrutura do FCI
0	0	0	0	0	0	-	-	- FCI (FCP e/ou FMD)
0	0	0	0	0	1	-	-	- FCP
0	0	0	0	1	0	-	-	- FMD
0	0	0	0	1	1	-	-	- Formato proprietário

O campo Lc deve codificar o tamanho do campo de dados, de acordo com o caso, e não deve haver campo Le (caso 3 de APDU).

Comando READ BINARY

Este comando lê bytes de EFs transparentes, e não deve ser aplicado a outros tipos de EFs. O comando é especificado pelo byte Instruction = 0xB0. Os bytes P1 e P2 são codificados da seguinte maneira:

- Se o bit 8 de P1 for zero, os 15 bits restantes de P1:P2 codificam um offset, de 0 a 32767, a partir do qual a leitura será feita;
- Se o bit 8 de P1 for 1, então os bits 7 e 6 de P1 são 00 (bits reservados para uso futuro), bits 5 a 1 de P1 codificam um ID curto de EF, e P2 codifica um offset de 0 a 255.

Caso o ID curto de EF seja válido, o camando altera o canal lógico antes de fazer a leitura. Caso não seja especificado um ID curto de EF, deve haver um EF já selecionado no canal lógico.

Não deve haver campo de dados nem Lc, e o campo Le informa o númeo de bytes que devem ser lidos (caso 2 de APDU).

ISO 7816-8

A parte 8 do pardrão ISO 7816 define comandos para operações criptográficas: geração de par de chaves, cifragem e decifragem, assinatura e verificação de assinatura, cálculo de hash. A maioria dos comandos são na verdade subcomandos do comando PERFORM SECURITY OPERATION, sendo diferenciados pelos bytes P1 e P2. As operações em geral são feitas com uma seqüência de comandos, iniciada pelo comando MANAGE SECURITY ENVIRONMENT.

Por exemplo, para assinar digitalmente um documento, primeiro apresentamos o PIN com o comando VERIFY; depois fazemos um MANAGE SECURITY ENVIRONMENT, especificando o algoritmo de hash a ser utilizado, o padding e a chave privada a ser utilizada; em seguida, enviamos um comando HASH, ou enviando o documento para o cartão fazer o hash (comando COMPUTE HASH) ou enviando o hash pré-calculado (comando PUT HASH), e só então enviamos um comando COMPUTE SIGNATURE, que irá retornar o valor da assinatura.

PKCS #15

O padrão PKCS#15, elaborado pela RSA Security, define uma organização baseada na estrutura lógica de arquivos definida pelo padrão ISO 7816, com a qual são armazenados certificados e informações correlatas. Todas as informações estão armazenadas em EFs filhos de um mesmo DF, o PKCS#15 DF, que pode ser selecionado a partir de seu nome, A0:00:00:00:63 || "PKCS-15". Todos os EFs têm estrutura transparente, e contém valores em ASN.1 codificados em BER/DER. Dentro os EFs presentes no PKCS#15, podemos citar:

ODF - *Object Directory File*

Arquivo com ID definido pelo padrão (50:31), contém os IDs dos demais EFs (cujos IDs não são fixados pelo padrão).

PrKDFs - *Private Keys Directory Files*

Contém informações sobre as chaves privadas, como rótulos, uso pretendido, etc. Possui um ID denominado “auth ID”, que é uma referência cruzada para o ID do PIN que autoriza o uso da chave. Existe também um identificador denominado apenas “ID”, que permite relacionar a chave privada à correspondente chave pública, e também possivelmente a um certificado. O “keyReference” (referência de chave) é um byte que identifica a chave privada apenas dentro do contexto do próprio cartão. Este byte é informado no comando MANAGE SECURITY ENVIRONMENT para especificar qual chave utilizar.

CDFs - *Certificate Directory Files*

Contém referências para os EFs onde estão armazenados os certificados. Estas referências são obrigatórias, mas o CDF pode opcionalmente conter os certificados. Entretanto, geralmente eles não contém o certificado completo, pois isso constitui duplicação de informação no cartão, ou seja, desperdício de espaço, já que os certificados obrigatoriamente estão presentes completos em outros EFs. Contém também o ID que permite fazer referência cruzada com um par de chaves associado ao certificado.

Um exemplo operacional

Para demonstrar os conceitos expostos neste documento sendo colocados em prática, mostraremos como assinar digitalmente um documento com um smartcard Starcos 2.3. Assume-se que já se tem calculado o hash do documento a ser assinado. Os comandos baseiam-se tanto nos padrões ISO 7816 e PKCS #15 quanto no manual de APDUs específico do smartcard.

Todos os números nesta seção que aparecem na forma "XX:YY:ZZ:.." são hexadecimais.

1. Buscar os certificados no cartão

Para assinar digitalmente um documento, precisamos primeiro recuperar o certificado do assinante, bem como os certificados que compõem sua cadeia de validação, e validar o certificado. Essas ações decompõem-se nos seguintes passos:

1.1. Acessar o DF do PKCS #15

APDU: 00:A4:04:0C:0C:A0:00:00:00:63:50:4B:43:53:2D:31:35

Class = 0x00
Instruction = 0xA4 (comando SELECT FILE, ISO 7816-4)
P1 = 0x04 (selecionar DF por nome)
P2 = 0x0C (selecionar a primeira/única ocorrência do arquivo, e não retornar FCI)
Lc = 0x0C
Data = A0:00:00:00:63:50:4B:43:53:2D:31:35
(A0 00 00 00 63 || "PKCS-15")

O cartão Starcos só suporta P2 = 0x00 ou 0x0C, significando retornar FCI ou não retornar nada. Note que o padrão ISO 7816-4 especifica que P2 = 0x0C significa retornar FMD.

1.2. Acessar o ODF

APDU: 00:A4:00:0C:02:50:31

Class = 0x00
Instruction = 0xA4 (comando SELECT FILE, ISO 7816-4)
P1 = 0x00 (selecionar MF, DF ou EF)
P2 = 0x00 (retornar FCI)
Lc = 0x02
Data = 50:31 (id do ODF)
Resposta = FCI

Fazendo o parse do FCI, descobre-se o tamanho do ODF. Suponha que esse tamanho seja 48 bytes (0x30).

1.3. Ler o EF corrente (ODF)

APDU: 00:B0:00:00:00

Class = 0x00

Instruction = 0xB0 (comando READ BINARY, ISO 7816-4)
P1:P2 = 00:00 (ler a partir do começo do arquivo)
Le = 0x30 (retornar 48 bytes)

Fazendo o parse do ODF, descobre-se o id do CDF (Certificate Directory Files) e do PrKDFs (Private Keys Directory Files). Suponha que esses ids são, respectivamente, 44:04 e 44:00.

1.4. Acessar e ler o CDF

Os comandos abaixo são muito similares aos anteriores, e por isso sua explicação será abreviada.

APDU: 00:A4:00:00:02:44:04 (seleciona o CDF, obtendo como resposta o FCI)

APDU: 00:B0:00:00:XX (lê o CDF todo)
onde XX é o tamanho do CDF, obtido do FCI recuperado no comando anterior.

Fazendo o parse do CDF, obtem-se todos os certificados do arquivo*.

* Na verdade, é preciso buscar o certificado em um outro EF, cujo ID está contido no CDF. Seria portanto necessário acessar, ler e parsear esse outro arquivo. Esses passos foram omitidos para fins de simplificação.

1.5. Acessar e ler o PrKDFs

APDU: 00:A4:00:00:02:44:00 (seleciona o PrKDFs, obtendo como resposta o FCI)

APDU: 00:B0:00:00:XX
onde XX é o tamanho do PrKDFs, obtido do FCI recuperado no comando anterior.

A informação que buscamos no PrKDFs é o keyReference de cada chave privada presente no cartão, além do ID que permite fazer referência cruzada com o certificado correpondente à chave e o “auth ID”.

Assim, após esse passo, temos todos os certificados presentes no cartão, e para cada certificado que tem uma chave privada correspondente no cartão, temos o keyReference dessa chave e o ID do PIN que autoriza seu uso. Os próximos passos são escolher qual das chaves privadas usar para assinar o cartão e então verificar a cadeia de validação do certificado correspondente, além de checar listas de revogação. Estas operações fogem ao escopo deste documento.

2. Assinar o documento

De posse do keyReference da chave privada com o qual se deseja assinar o documento, e assumindo verificada a validade do certificado correspondente, passamos então à fase de assinatura do documento.

2.1. Apresentar o PIN

APDU: 00:20:00:82:08:(PID)

Class = 0x00
Instruction = 0x20 (comando VERIFY, ISO 7816-4)
P1 = 0x00 (valor fixado pelo padrão)
P2 = 0x82 (auth ID, obtido do PrKDFs)
Lc = 0x08
Data = PIN do usuário, em ASCII, padded com zeros para completar 8 bytes
(por exemplo, se o PIN é 1234, então Data = 31:32:33:34:00:00:00:00)

Caso o PIN informado esteja errado, o comando retorna SW1:SW2 = 63:CX, onde X é o número de tentativas restantes antes que o cartão seja bloqueado.

2.2. Criar um contexto para operação de segurança

APDU: 00:22:41:B6:06:84:01:84:80:01:12

Class = 0x00
Instruction = 0x22 (comando MANAGE SECURITY ENVIRONMENT, ISO 7816-4)
P1:P2 = 41:B6 (compute signature / verify signature)
Lc = 0x06
Data = 84:01:84:80:01:12
84:01:84 – especifica chave privada, keyReference = 0x84
80:01:12 – especifica algoritmo SHA-1 com padding PKCS#1

A rigor, P1:P2 = 41:B6 significa "verify signature" e P1:P2 = 81:B6 significa "compute signature", porém o cartão Starcos não diferencia os valores 0x41 e 0x81 no P1. Este comando inicia o comando composto de assinatura.

2.3. Setar o hash do documento

APDU: 00:2A:90:81:14:(HASH)

Class = 0x00
Instruction = 0x2A (comando PERFORM SECURITY OPERATION, ISO 7816-8)
P1 = 0x90 (subcomando HASH)
P2 = 0x81 (valor obtido do manual do smartcard, especifica subcomando PUT HASH)
Lc = 0x14
Data = hash (sha-1) do documento, 20 bytes

O comando PERFORM SECURITY OPERATION é especificado pelo byte Instruction = 0x2A. Entretanto, existem diversos "subcomandos", um deles sendo o HASH, que é especificado por P1 = 0x90. O subcomando HASH, por sua vez, possui as ramificações COMPUTE HASH, SET HASH e PUT HASH, que respectivamente servem para requisitar ao cartão o cálculo completo do hash, requisitar ao cartão apenas a parte final do cálculo do hash, e informar ao cartão um valor de hash pré-calculado. No caso deste exemplo, utilizamos o comando PUT HASH.

2.4. Calcular a assinatura

APDU: 00:2A:9E:9A:80

Class = 0x00
Instruction = 0x2A (comando PERFORM SECURITY OPERATION, ISO 7816-8)
P1 = 0x9E (subcomando COMPUTE SIGNATURE)

P2 = 0x9A
Le = 0x80 (128 bytes (1024 bits) esperados, tamanho de uma assinatura RSA)
Response = valor da assinatura

Este comando encerra o comando composto de assinatura, iniciado no passo 2.2.

Ferramentas

OpenSC - www.opensc-project.org/opensc

OpenSC é um projeto de software livre que provê um conjunto de bibliotecas e ferramentars para acessar smartcards, com foco em smartcards que suportam operações criptográficas. Como um projeto de software livre, o OpenSC tem como principal distribuição o código fonte das ferramentas, para que o usuário as compile.

A compilação em ambiente Linux é automatizada com makefiles, e se mostrou bem simples. Entretanto, fazer o mesmo em um ambiente Windows pode ser muito complexo. É possível baixar as ferramentas pré-compiladas (binários) para Windows, em um pacote chamado "Smart Card Bundle", porém o pacote disponível quando da elaboração deste documento continha versões desatualizadas das ferramentas. De todo modo, tais versões proveram funcionalidades satisfatórias. Segue uma breve descrição das ferramentas que se provaram mais úteis:

opensc-tool: lê e interpreta o ATR, lista leitores de smartcards disponíveis, e envia APDUs não-formatados (seqüência de bytes, cabe ao usuário formatar o APDU corretamente segundo o padrão ISO 7816-3). Este formato de envio de APDUs é interessante por permitir um maior controle sobre o processo de envio.

opensc-explorer: permite ao usuário "navegar" pela estrutura de diretórios e arquivos do smartcard. Possui uma sintaxe similar a um shell unix (comandos cd, ls, cat, etc).

pkcs15-tool: lê do cartão informações relacionadas ao PKCS#15, como certificados e chaves, permite mudança do PIN.

pkcs15-crypt: assina documentos digitalmente.

pkcs15-init: inicializa cartões em branco com estrutura de PKCS#15, gera pares de chaves (no próprio cartão, se possível), grava certificados, etc.

p15dump: lista todas as informações continadas no diretório PKCS#15 do cartão.

pkcs11-tool: lista certificados, assina documentos, faz hash de documentos (no cartão), gera pares de chaves (no cartão), etc.

Smart Card ToolSet PRO - www.scardsoft.com

Esta ferramenta fornece uma interface mais amigável para operar com APDUs. Permite obter informações sobre o cartão, enviar APDUs formatados (preenche-se os campos Class, Instruction, etc), fazer *batch files* de APDUs, interpreta os bytes de reposta para os APDUs enviados, permite fazer um scan de APDUs (fixa-se por exemplo o Class, P1 e P2, e varia-se o

Instruction de 0x00 a 0xFF para ver quais são suportados pelo cartão). A ferramenta é paga, porém sua versão shareware possui quase todas as funcionalides da versão plena, com o inconveniente que a ferramenta precisa ser reiniciada após o envio de 10 APDUs.

OpenSSL - www.openssl.org

O OpenSSL pode ser útil nesse contexto para fazer o parse das estruturas em notação ASN.1 codificadas em DER recuperadas do cartão. O comando do OpenSSL para tal operação é o "asn1parse".

SafeSign Token Administration

Ferramenta obtida da certificadora CertiSign (www.certisign.com.br), oferece uma interface simples para visualizar os certificados e IDs digitais no cartão, bem como mudar o PIN, importar IDs digitais para o cartão, etc.

A Padronização na ICP Brasil

Um dos objetivos das aplicações sendo desenvolvidas durante esta pesquisa é trabalhar com assinaturas digitais no contexto da infraestrutura de chaves públicas brasileira, ou seja, operar com e-CPFs. Assim sendo, pesquisou-se então sobre que padronizações a ICP Brasil impõe sobre os cartões e sobre conteúdo dos mesmos.

O resultado foi desanimador: a ICP Brasil atem-se a padronizar o formato dos certificados presentes no cartão. Não encontramos nada que tratasse do modo como os certificados devem ser armazenados no cartão, tampouco a que padrões os cartões devem aderir.

O que acontece nesse meio é que as interoperabilidade funciona à base de boas práticas. Por exemplo, todos os e-CPFs com os quais pudemos ter contato tinham seus certificados armazenados segundo o padrão PKCS#15. Isso significa que as certificadoras que gravaram tais cartões t6em aderido ao padrão PKCS#15, seja por bom senso ou por simples acaso, por usarem aplicações que gravam certificados nesse formato. Entretanto, não há (ou pelo menos não fomos capazes de encontrar) nenhum mecanismo legislativo que garanta que os certificados serão gravados dessa forma. Uma certificadora poderia gravar certificados de outra forma nos cartões, e ainda estar em conformidade com a ICP Brasil.

O caso do conteúdo do cartão, de como encontrar e ler os certificados, tem se mostrado menos problemático do que a questão da operação com o cartão. O padrão ISO 7816 é muito extenso, de forma que a maioria dos cartões implementa apenas uma pequena parte do mesmo. Geralmente partes que são colocadas como opcionais pelo padrão não são implementadas, mas o problema está quando há incoformidades com o padrão. Dessa maneira, é impraticável desenvolver uma aplicação que se comunique com smartcards de maneira genérica. O que se faz é ter trechos de código – ou módulos inteiros – específicos para cada tipo de cartão. Mas e quando novos modelos de cartões são inseridos no mercado?

Shroud of Secrecy

Implementar código específico para cada modelo de cartão implica na necessidade de documentação específica para os cartões. Pode-se pensar que uma equipe de desenvolvedores fazendo uma aplicação para operar com smartcards teria poucos problemas em obter tais manuais, já que seria de interesse dos próprios fabricantes que seus cartões funcionem com o maior número de aplicações possível. Entretanto, isso nem sempre é verdade.

Muitos fabricantes do ramo parecem preferir se cobrir em um *shroud of secrecy* ("manto de segredo"), tratando tais documentações como segredos industriais. É o caso da Siemens, que só disponibiliza os manuais do seu sistema operacional de smartcards mediante assinatura de um NDA (*non disclosure agreement*).

Outro caso é a Giesecke & Devrient, ou simplesmente G&D, fabriacante dos cartões Starcos, atualmente comuns em Brasília devido à venda pela certificadora CertiSign. Até a versão 2.3 do cartão, a empresa disponibilizava os manuais livremente. Entretanto, mudou sua política, tirando do ar os manuais para os cartões antigos e passando a exigir assinatura de NDA para obtenção dos manuais da versão mais nova, 3.0.

Felizmente, existem alguns fabricantes que adotam uma postura oposta – disponibilizam livremente a documantação para operação de seus cartões. É o caso da Axalto (antiga Gemalto), que disponibiliza os manuais dos cartões Cryptoflex no seu site:

http://www.cryptoflex.com/Support/documentation.html

Conclusões

Como pode ser visto com o exemplo dos padrões ISO 7816 e PKCS#15, há um grande esforço de padronização em âmbito internacional no sentido de viabilizar a interoperabilidade de smartcards. Entretanto, a interoperabilidade é dificultada pela falta de adesão aos padrões, seja por falta de normatizações que forcem isso, ou pelo fato dos padrões serem demasiadamente complexos. O fato é que garantir a interoperabilidade nesse meio parece tarefa impossível, e atualizações em aplicações para tratar mudanças no meio que desestabilizem a interoperabilidade incerta que existe parecem inevitáveis.

Agradecimentos

Gostaria de agradecer ao co-pesquisador e co-desenvolvedor Caline Dias de Alencar Ribeiro por sua parceria na pesquisa que deu origem a este documento; ao professor Pedro Rezende, pelos conhecimentos transmitidos na área de segurança e pela orientação nesta pesquisa; e à empresa VipSign, por ter proporcionado a oportunidade de realizar a pesquisa.

Bibliografia

"PKCS #15 v1.1: Cryptographic Token Information Syntax Standard", RSA Laboratories, 6 de Junho de 2000

ISO/IEC 7816-3 (*request for review*) – "Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols" – Amendment 2: "Structures and transmission of APDU messages", 4 de Janeiro de 2002

ISO/IEC 7816-4 (*request for review*) – "Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 4: Interindustry commands for interchange", 4 de Janeiro de 2002

ISO/IEC 7816-8 (*request for review*) – "Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 8: Interindustry commands for a cryptographic toolbox", 4 de Janeiro de 2002

"The ISO 7816 Smart Card Standard: Overview"
http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816.aspx

"Reference Manual – Smart Card Operating System – STARCOS® S 2.1", Giesecke & Devrient, 2001

"Reference Manual – Smart Card Operating System – STARCOS® SPK 2.3 – Supplement to the STARCOS S 2.1 Reference Manual", Giesecke & Devrient, 2001

Wikipedia: "Smart card"
http://en.wikipedia.org/wiki/Smart_cards

OpenSC project
http://www.opensc-project.org/opensc/