Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

APLIKACJA BIBLIOTECZNA JELIB

Jakub Tomczak

Promotor
prof. dr hab. inż. Joanna Józefowska

Opiekun
mgr inż. Marek Gosławski

Poznań, 2009 r.

Autor pragnie w tym miejscu podziękować rodzicom,
bratu oraz narzeczonej, którzy w czasie prac nad projektem
byli nieocenionym wsparciem.

Spis treści

Rozdział 1

Wstęp

1.1 Elektroniczna Legitymacja Studencka

Polskie uczelnie weszły w XXI wiek. Zgodnie z rozporządzeniem Ministra Nauki i Szkolnictwa Wyższego z dnia 2 listopada 2006 w sprawie dokumentacji przebiegu studiów [1] od 01 stycznia 2007 zaistniała możliwość wykorzystania karty elektronicznej jako legitymacji studenckiej. W tym momencie otworzyły się drzwi do świata możliwości jakie oferuje elektroniczny identyfikator żaka. Politechnika Poznańska jest jedną z nielicznych instytucji w kraju, która podjęła się wprowadzenia w życie owego rozporządzenia oraz zorganizowania przetargu na zakup blankietów ELS dla 69 innych uczelni.

Według założeń przetargu ELS miała służyć nie tylko identyfikacji właściciela, lecz również umożliwiać np. elektroniczne podpisywanie dokumentów czy zastąpienie biletów miesięcznych komunikacji miejskiej. W ogólności: instalowanie na niej aplikacji uczelnianych i nie tylko, spełniających inne cele niż wyłącznie poświadczenie tożsamości. Takie wymagania zostały spełnione. Z tej też przyczyny postanowiono, by jednym z możliwych zastosowań było wykorzystanie jej jako karty bibliotecznej. Dzięki temu użytkownik nie musiałby posiadać dedykowanej karty (na dodatek innej dla każdej biblioteki), jak i również mógłby w przyszłości, przy pomocy specjalnych terminali, odczytać z niej informacje o ostatnich wypożyczeniach, lub czy nie przekroczył już terminu oddania książek. Takie rozwiązanie usprawniłoby obsługę petenta w bibliotece, jak i uprościło formalności związane z wydawaniem nowych kart czytelnika.

Wraz z wprowadzeniem ELS zdecydowano się, by umieścić na jej rewersie kod kreskowy, który identyfikowałby właściciela legitymacji w bibliotece. Trzeba przyznać, że jest jedno z najprostszych rozwiązań, zarazem mało użyteczne, gdyż kod ten jest ważny tylko w bibliotekach Poznańskiej Fundacji Bibliotek Naukowych. Stąd też powstał pomysł, by przenieść identyfikator biblioteczny z powierzchni karty do chipu znajdującego się na niej. Pamięć oferowana przez przeciętną kartę pozwala na zapisanie zdecydowanie więcej informacji niż na samej karcie, można by więc przechowywać w niej identyfikator nie tylko do jednej biblioteki, ale do wielu, i więcej, niż tylko identyfikator. To wymaga stworzenia

odpowiedniej aplikacji, apletu, który po zainstalowaniu na karcie oferowałby potrzebną funkcjonalność.

1.2 Cel i zakres pracy

Celem pracy jest zaprojektowanie oraz wykonanie apletu na kartę procesorową, który umożliwiałby identyfikację posiadacza karty w systemie bibliotecznym Horizon oraz wykonanie na niej operacji takich jak odczyt ostatnich zdarzeń, dodanie nowego zdarzenia, ustawienia i zdjęcia blokady konta posiadacza karty w określonej bibliotece. Do zadań należy opracowanie poleceń APDU odpowiadających działaniom na danych zawartych na karcie w postaci plików, projekt i implementacja aplikacji jElib oraz opcjolane opracowanie ogólnej metody umożliwiającej przeniesienie aplikacji w wersji plikowej korzystającej z ASN.1 lub nie, do apletu JavaCard. W ramach zakresu dodatkowego jest też implementacja "wtyczek"dla systemu obsługi ELS (SELS) Politechniki Poznańskiej.

Praca składa się z sześciu rozdziałów. W rozdziale 2 omówione są podstawy dotyczące kart inteligentnych. Rozdział 3 dotyczy technologi związanych bezpośrednio z kartami lub na nich implementowanych. W kolejnym rozdziale, 4-tym, przedstawiony jest opis projektu jElib, jego implementacji oraz użytych narzędzi. Rozdział 5 zawiera wnioski autora oraz sugerowane drogi rozwijania projektu. Na ostatni rozdział składa się opis stworzonego API, które aplikacja jElib implementuje.

Rozdział 2

Klasyfikacja i standardy kart elektronicznych

2.1 Karty Smart Card i ich klasyfikacja

Podrozdział ten został opracowany na podstawie [13].

2.1.1 Klasyfikacja ze względu na budowę

Rozmiary kart elektronicznych określone są przez normę ISO/IEC 7810. Norma ta pod względem rozmiaru i umieszczenia układu elektronicznego na karcie wyróżnia ich 4 rodzaje. Są to karty typu:

ID-1 85,60 x 53,98 mm - najpopularniejszy rozmiar (np. karta płatnicza, dowód osobisty),

ID-2 105 x 74 mm - rozmiar karty używany w np. niemieckich dowodach tożsamości,

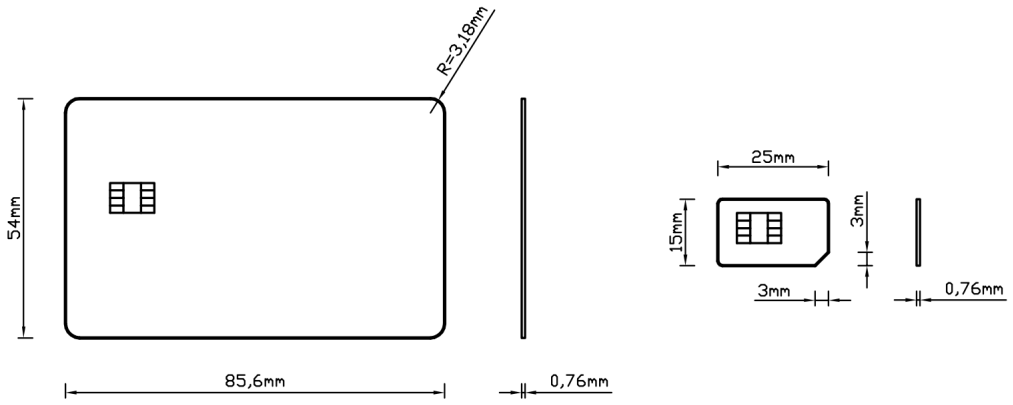
ID-3 125 x 88 mm - rozmiar paszportów niektórych krajów, bądź wiz,

ID-000 25 x 15 mm - najmniejsze karty, stosowane często w telefonach komórkowych (np. karta SIM).

Ze względu na budowę wewnętrzną karty elektroniczne można podzielić na karty pamięciowe oraz karty mikroprocesorowe. Najważniejsze cechy obu typów przedstawione są poniżej.

Karta pamięciowa - wyposażona w pamięć tylko do odczytu typu ROM oraz zapisywalną pamięć typu EEPROM.

Karta mikroprocesorowa - wyposażona w procesor, opcjonalny koprocesor, pamięć tylko do odczytu typu ROM, gdzie znajduje się system operacyjny, pamięć operacyjną typu RAM oraz zapisywalną pamięć typu EEPROM, w której przechowywane są kod i dane aplikacji.



Rysunek 2.1: Karta typu *ID-1* (na lewo) oraz *ID-000* (na prawo)

2.1.2 Klasyfikacja ze względu na interfejsy wymiany danych

W celu ułatwienia sposobu korzystania oraz zwiększenia ilości zastosowań producenci kart opracowali dwa sposoby komunikacji z kartą elektroniczną.

Pierwszy, i najbardziej rozpowszechniony, oparty jest na *metodzie stykowej*. Wymiana danych odbywa się poprzez styki znajdujące się na powierzchni mikroprocesora karty. W momencie wsunięcia karty do czytnika terminala wypustki odpowiadające ułożeniu punktów styku na karcie są przykładane do powierzchni mikroprocesora i poprzez regulowanie napięcia elektrycznego następuje wymiana danych pomiędzy kartą a terminalem. Najczęściej spotykane zastosowanie tej formy komunikacji to bankomaty i systemy logowania.

Drugą metodą wymiany danych z kartą jest *metoda bezstykowa* stosowana w tzw. kartach zbliżeniowych. Jej idea polega na umieszczeniu karty w zmiennym polu elektromagnetycznym (w niewielkiej odległości od terminala), które wzbudza prąd zmienny w antenie/cewce znajdującej się w karcie. Dzięki temu do karty zostaje doprowadzone zasilanie i umożliwiona zostaje komunikacja pomiędzy nią a terminalem bezstykowym. Takie rozwiązanie można spotkać często w systemach kontroli dostępu (otwieranie drzwi), rejestracji czasu pracy, czy ostatnio również przy terminalach płatniczych w sklepie.

Producenci oferują również *karty hybrydowe* lub *dualne*, łączące obie te technologie w jednym blankiecie.

2.1.3 Klasyfikacja ze względu na system operacyjny

Karty mikroprocesorowe mają tą przewagę nad pamięciowymi, że umożliwiają uruchomienie na nich systemu operacyjnego. Systemy te odpowiadają za zarządzanie pamięcią RAM jak i EEPROM, zarządzanie systemem plików (dokładnie omówiony w 2.2.5), operacje I/O oraz dają możliwość wgrania do pamięci karty dodatkowych aplikacji. Poniżej

omówiono niektóre z nich.

Natywne Aplikacje tworzone na potrzeby kart wyposażonych w systemy natywne są zazwyczaj pisane w języku C bądź Assembler. Są to karty, w których to producent odpowiada za zestaw funkcjonalności, jaką karta będzie oferować, gdyż operacja wgrywania oprogramowania ma miejsce tylko podczas procesu wytwarzania karty.

MPCOS System realizujący funkcjonalność elektronicznej portmonetki. Dodatkowo obsługuje komendy zgodne z m.in. *ISO-7816-4* i *ISO-7816-9*.

Java Card W związku z rozwojem obiektowych języków programowania znalazły one też swoje zastosowanie w kartach elektronicznych. Spośród ich szerokiej gamy została wybrana Java. Są to teraz najbardziej rozpowszechnione karty w środowisku programistów. Język używany do tworzenia aplikacji na ten system jest okrojonym podzbiorem Java Standard Edition, stąd też bardzo łatwo jest zacząć w nim programować.

MULTOS Jest to system, w którym możliwe jest uruchomienie aplikacji z systemów natywnych, pisanych w języku C, jak i tych dla Java Card.

2.2 Standardy ISO

2.2.1 ISO 7816-1

Standard *ISO 7816-1* [9] definiuje ogólne charakterystyki materiału z jakiego zbudowane są karty. Mowa w nim m.in. o odporności na promieniowanie ultrafioletowe, promienie Roentgena, zginanie, wstrząsy oraz określa temperaturę pracy czy opór punktów stykowych układu mikroprocesorowego karty.

2.2.2 ISO 7816-2

Standard ten określa dokładne umiejscowienie punktów styku mikroprocesora karty z czytnikiem na karcie, jak i minimalne wymiary tych styków. [10]

2.2.3 ISO 7816-3

Standard *ISO 7816-3* opisuje zasady komunikacji pomiędzy czytnikiem a kartą na poziomie elektrycznym. Definiuje kolejność sekwencji sygnałów pomiędzy nimi. [8]

2.2.4 ISO 7816-4

Standard *ISO 7816-4* [12] jest największym spośród serii ISO 7816. Opracowany w celu określenia form przechowywania danych na kartach, jak również dostępu do nich i sposobu manipulowania nimi.

APDU

Pierwszą rzeczą, którą ten dokument definiuje jest postać komend, poprzez które następuje komunikacja pomiędzy kartą, a systemem znajdującym się poza nią. Są to komendy *APDU* (Application Protocol Data Unit) i wyróżnia się ich dwa rodzaje: komenda żądania i komenda odpowiedzi. Komenda żądania wysyłana jest do karty, natomiast karta odpowiada systemowi zewnętrznemu komendą odpowiedzi. Poniższa tabela prezentuje format tych dwóch struktur.

Pole	Opis	Długość w bajtach
Nagłówek	Bajt klasy CLA	1
	Bajt instrukcji INS	1
	Bajty parametrów P1-P2	2
Pole L_c	Brak, jeśli $N_c = 0$, obecne, jeśli $N_c > 0$	0, 1 lub 3
Pole danych	Brak, jeśli $N_c = 0$, obecne jako ciąg N_c bajtów, gdy $N_c > 0$	N_c
Pole L_e	Brak, jeśli $N_e = 0$, obecne, jeśli $N_e > 0$	0, 1, 2 lub 3

Pole odpowiedzi	Brak, jeśli $N_c = 0$, obecne jako ciąg N_c bajtów, gdy $N_c > 0$	N_r, maks. N_e
Pole statusu	Bajty statusu SW1-SW2	2

Tablica 2.2: Tabela formatu komend APDU do i z karty.[12]

Najbardziej popularnymi rozmiarami pól danych wysyłanych, jak i zwracanych jest 256 bajtów (0xFF heksadecymalnie). Standard jednak przewiduje również możliwość użycia pól o długości do 65 535, po odpowiednim spreparowaniu pól L_c i L_e, nie jest to jednak rozwiązanie obsługiwane przez wszystkie czytniki/karty.

Pewne elementy wchodzące w skład komendy APDU są również opisane w standardzie, mowa tutaj o polach *CLA*, *INS* oraz częściowo *P1*, *P2* i *SW1*, *SW1*.

CLA Pole *CLA* określa klasę komendy. Klasa może wskazywać do jakiej aplikacji się odnosi, użycie szyfrowanego połączenia czy łączenie komend w łańcuch. Klasą standardu jest '00', a w przypadku połączenia szyfrowanego '04'.

INS Pole *INS* wskazuje konkretną komendę apletu. Jest to odpowiednik odwołania się do nazwy metody, którą chcemy wykonać. Standard ten, jak i kilka innych definiują wartości INS, które wraz z klasą standardu *ISO 7816-4* tworzą pary zarezerwowane.

P1 i P2 Wartości P1 i P2 są traktowane jako parametry metody. W tym wypadku muszą być one krótkie, bądź pełnić rolę flagi, gdyż mogą zajmować maksymalnie 2 bajty (łącznie). Większe parametry muszą być przekazane w polu danych komendy. Standard w przypadku części operacji definiuje jakie parametry P1 i P2 operacja powinna przyjmować (i jak w tym przypadku powinna się zachowywać).

SW1 i SW2 Są to wartości zwracane wraz z rezultatem komendy. SW oznacza w języku angielskim *status word*, stąd też obie wartości mówią o stanie zakończenia danej operacji. SW1 oznacza kategorię stanu (standard definiuje podział na ostrzeżenia i błędy, przy czym wartości oznaczających tą samą kategorię jest więcej niż 1), SW2 natomiast wskazuje konkretny stan. Większość stanów ma postać 0x6xxx hexadecymalnie, stan 0x9000 oznacza pomyślne zakończenie operacji.

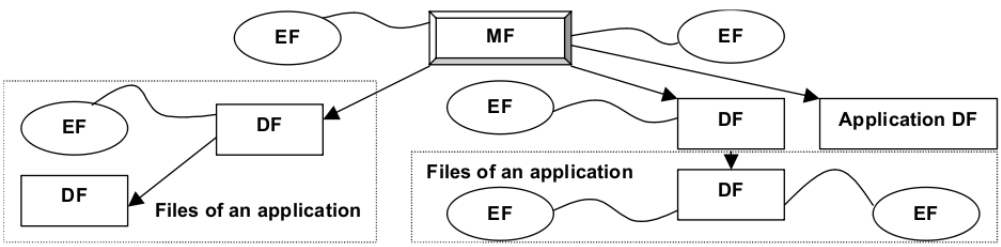
System plików

Standard definiuje organizację danych składowanych na karcie. Mowa tutaj o systemie plików. Wyróżniane są dwa typy struktury:

plik EF (ang. *elementary file*) struktura zawierająca dane, to w niej dane są zapisywane i z niej odczytywane,

plik DF (ang. *dedicated file*) struktura przechowująca aplikacje, pliki EF, jak i pliki DF, oraz inne obiekty.

Pliki DF pełnią rolę zbliżoną do normalnych katalogów, pliki EF natomiast zwykłych plików. Każdy plik ma swój identyfikator, unikalny w ramach jednego pliku DF (dla obu typów plików jest to 2-bajtowy identyfikator pliku, natomiast pliki DF mogą posiadać jeszcze nazwę, która może mieć więcej bajtów). System plików karty posiada jeden katalog specjalny, *MF*, o identyfikatorze *0x3f00*. Pełni on rolę korzenia struktury systemu plików. Poniżej zamieszczony schemat 2.2 przedstawia przykładową strukturę zbudowaną z plików EF i DF.



Rysunek 2.2: Przykładowa struktura zbudowana z plików EF i DF.[12]

Wyboru pliku można dokonać w następujący sposób:

poprzez nazwę DF ,

poprzez identyfikator pliku ,

poprzez ścieżkę ciąg bajtów, w którym każde dwa kolejne bajty oznaczają identyfikator kolejnego pliku na ścieżce pomiędzy MF a wybieranym plikiem,

poprzez krótki identyfikator plik opcjonalnie może posiadać również krótki, 5cio bitowy identyfikator.

Każdy plik przechowuje również informację o swoim typie, strukturze, rozmiarze, polityce dostępu itp. w specjalnej strukturze FCI (*File control information*). Dane te są przechowywane w strukturach *TLV* opisanych dokładniej w 3.3.

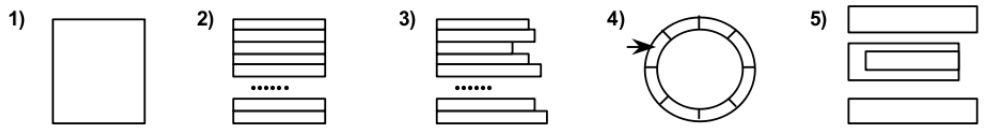
W standardzie znajdziemy również określone podstawowe struktury danych w plikach. Są to struktury:

prosta dane przechowywane są jako ciąg bajtów,

rekordowa dane przechowywane są w postaci identyfikowalnych struktur o stałem lub zmiennej długości,

TLV dane przechowywane są w postaci struktur *TLV*, które mogą się w sobie zagnieżdżać.

Poniższy schemat 2.3 przedstawia reprezentację poszczególnych podejść:



Rysunek 2.3: Rodzaje reprezentacji pliku EF.[12]

1. Struktura przezroczysta,
2. Liniowa struktura rekordów o stałej długości,
3. Liniowa struktura rekordów o zmiennej długości,
4. Cykliczna struktura rekordów o stałej długości (strzałka wskazuje ostatnio zapisany rekord),
5. Struktura TLV.

Architektura bezpieczeństwa

Standard *ISO 7816-4* definiuje również mechanizmy bezpieczeństwa, które można stosować w celu zwiększenia integralności danych, ich szyfrowania oraz zdefiniowania kontroli dostępu. Te mechanizmy to:

status bezpieczeństwa pozwala z regulowaną granularnością (na poziome globalnym, aplikacji, pliku czy komendy) określić aktualny stan obiektu osiągnięty po jednym, bądź serii poleceń związanych z bezpieczeństwem,

atrybuty bezpieczeństwa mają na celu zdefiniowanie ograniczeń i praw dostępu do danych obiektów w zależności od ich statusu bezpieczeństwa, mogą mówić również o operacjach potrzebnych, by konkretny status osiągnąć,

mechanizmy bezpieczeństwa w szczególności:

- uwierzytelnienie za pomocą hasła,
- uwierzytelnienie przy pomocy klucza,
- weryfikacja danych,
- szyfrowanie danych.

2.2.5 ISO 7816-9

Standard ten opisuje cykl życia plików oraz ich zarządzanie w systemie plików karty [11]. Wyróżnia on następujące stany pliku:

1. Plik nieistniejący (stan abstrakcyjny),
2. Stan utworzenia - plik istnieje, lecz mechanizmy bezpieczeństwa z nim skojarzone nie są aktywne,
3. Stan inicjalizacji - plik istnieje i niektóre mechanizmy bezpieczeństwa z nim skojarzone są aktywne,
4. Stan operacyjny aktywny - plik istnieje i wszystkie mechanizmy bezpieczeństwa z nim skojarzone są aktywne,
5. Stan operacyjny nieaktywny - plik istnieje i wszystkie mechanizmy bezpieczeństwa z nim skojarzone są aktywne, lecz można na nim użyć tylko operacji SELECT, ACTIVATE FILE, DELETE FILE i TERMINATE DF/EF,
6. Stan zakończenia - plik istnieje, ale jego wartość nie może zostać zmieniona, aczkolwiek wszystkie mechanizmy bezpieczeństwa z nim skojarzone są aktywne,
7. Stan zakończenia użytkowania karty - karta zostaje zablokowana i nie obsługuje już operacji SELECT,
8. Plik usunięty (stan abstrakcyjny).

Tranzycje pomiędzy stanami od 1 do 4 są jednokierunkowe (zgodnie z rosnącym indeksem stanu) i mogą się odbywać w dowolnej sekwencji przy czym każda sekwencja tranzycji zaczyna się na tranzycji 1 i kończy na 4 oraz przejścia następują ze stanu o indeksie

niższym do stanu o indeksie wyższym.
Tranzycja pomiędzy stanem 4 i 5 jest tranzycją odwracalną.
Tranzycja do stanu 6 jest tranzycją nieodwracalną i możliwą tylko ze stanu 5.
Tranzycje pomiędzy stanami 4/5, a stanami 7 i 8 są tranzycjami nieodwracalnymi i mogą odbywać się z pominięciem stanu 6 (zakończenia).

Standard określa również jakie operacje na plikach są możliwe. Te operacje to:

CREATE FILE tworzy plik EF, bądź katalog DF,

DELETE FILE usuwa plik EF, bądź katalog DF,

DEACTIVATE FILE dezaktywuje plik,

ACTIVATE FILE aktywuje plik,

TERMINATE DF zmienia stan katalogu DF na 'zakończony',

TERMINATE EF zmienia stan pliku EF na 'zakończony',

TERMINATE CARD USAGE blokuje kartę.

Rozdział 3

Technologie kart Java Card

3.1 Składniki Java Card OS

Jak już wcześniej powiedziano, większość kart wyposażona jest w system operacyjny, który odciąża programistę i zajmuje się obsługą pamięci jak i interfejsów I/O karty. Karty, dla których przeznaczony jest aplet jElib wykorzystują system Java Card. W ogólności można go podzielić na wirtualną maszynę oraz API, które połączone dają całość.

3.1.1 Java Card VM

Podrozdział ten został opracowany na podstawie [17].

Maszyna wirtualna Javy wykorzystana w kartach inteligentnych nie odbiega zbytnio pod względem zachowania od jej wersji przeznaczonej na urządzenia, których zasoby nie są tak bardzo ograniczone jak w przypadku kart (np. komputery typu desktop), jednakże stanowi jej bardzo zawężony podzbiór. Z racji małej ilości dostępnej pamięci (średnio ok. 1,2KB RAM, 32KB EEPROM do zapisu, oraz 32-48KB ROM) oraz architektury 16-to bitowej niewykonalnym lub nieopłacalnym jest zaimplementowanie takich mechanizmów jak arytmetyka zmiennoprzecinkowa czy zarządzanie wątkami. Poniższa lista prezentuje mechanizmy Java Card VM i te, które się w niej nie znajdują (a istnieją w normalnej maszynie wirtualnej), oraz obsługiwane i nieobsługiwane typy zmiennych.

Mechanizmy zaimplementowane w Java Card VM to:

- pakiety,
- dynamiczne tworzenie obiektów,
- zwalnianie obiektów (tylko w wersji Java Card 2.2 lub wyższej),
- metody wirtualne,
- wyjątki,
- struktury generyczne.

Mechanizmy niedostępne w Java Card VM to:

- dynamiczne ładowanie klas,
- Security Manager,
- finalizacja,
- wielowątkowość,
- klonowanie obiektów,
- typy *enum*,
- asercje,
- annotacje.

Typy zmiennych dostępne w Java Card VM to:

- byte,
- short,
- boolean,
- int (opcjonalnie),
- tablice jednowymiarowe,
- obiekty.

Mechanizmy niedostępne w Java Card VM to:

- double,
- float,
- long,
- tablice wielowymiarowe.

Java Card VM udostępnia również mały zbiór klas z pakietu `java.lang`.

Dodatkowo Java Card VM ma ograniczenia tyczące się maksymalnej liczby implementowanych przez klasę interface'ów, czy maksymalnej liczby metod w klasie. Są to jednak dość spore wielkości i jest mało prawdopodobnym, by w rzeczywistym zastosowaniu wartości te zostały przekroczone. Dokładne informacje na ten temat można uzyskać w [17].

Inną istotną różnicą pomiędzy wersją desktopową VM a wersją Java Card jest okres życia maszyny wirtualnej Java Card. Działa ona cały czas, gdy karta jest w użyciu. W momencie włożenia karty do czytnika, przykładane jest do niej napięcie i maszyna wirtualna

zaczyna działać. Wtedy możliwe jest zapisywanie obiektów w pamięci. Po odłączeniu napięcia (wyjęciu karty z czytnika), maszyna wirtualna przechodzi w stan wstrzymania, jednakże obiekty zapisane podczas zakończonej sesji nie są tracone. Po ponownym włożeniu karty do czytnika możliwe jest ich odczytanie.

3.1.2 Java Card RE

Podrozdział ten został opracowany na podstawie [16].

Każdy aplet Java Card jest uruchamiany wewnątrz środowiska uruchomieniowego Java Card RE. Nadzoruje ono cykl życia apletu, udostępnia mu funkcjonalność, z której może korzystać, oraz zapewnia mechanizmy, które czuwają nad poprawnością wykonywanych operacji. Java Card RE jest integralną częścią karty, stąd też każda karta jest w niego wyposażona.

Cykl życia apletu można ograniczyć do jego 5 metod. Dzięki nim możliwe jest tworzenie/usuwanie apletu, uczynienie go aktywnym/nieaktywnym oraz wywołanie jego metod.

install powoduje utworzenie nowej instancji apletu w pamięci karty. Jest to również najlepszy moment na inicjalizację pamięci trwałej potrzebnej do pracy apletu.

select powoduje wybranie apletu, co umożliwia kierowanie do niego poleceń. Tutaj ma miejsce przygotowanie danych potrzebnych w trakcie sesji użytkowania.

process powoduje przekazanie sterowania do apletu i umożliwia mu wykonanie jego funkcjonalności.

deselect umożliwia obsługę zakończenia sesji użytkowania apletu.

uninstall umożliwia obsługę usunięcia apletu.

Podczas działania metody `install` ma miejsce również wpisanie apletu do rejestru aplikacji karty, pod kluczem będącym *AID* apletu. Jest to później potrzebne, przy wybieraniu apletu jako aktywnego.

Obsługa mechanizmów związanych z wybieraniem apletu (metoda `select`) jest w Java Card RE dość rozbudowana. Pozwala ona m.in. na domyślne wybieranie wskazanego apletu przy wybraniu jednego z kilku kanałów logicznych karty, czy też dostęp do apletu poprzez kilka kanałów jednocześnie.

Inną przydatną funkcją dostępną w Java Card RE są obiekty *ulotne* (ang. transient). System Java Card traktuje domyślnie pola klas jako obiekty trwałe (zapisywane w wolnej pamięci EEPROM) a zmienne lokalne w szybkiej pamięci RAM. Obiekty ulotne dają nam możliwość operowania nimi poza zasięgiem metody, jako polem obiektu, ale w rzeczywistości rezydują one w pamięci RAM. Tymi obiektami są tablice typów prostych `byte` i `short`. Obiekty te są czyszczone przy wyjęci karty z czytnika, bądź przy zmianie aktywnego apletu. Jest to bardzo przydatny mechanizm wykorzystywany np. do tworzenia buforów danych tymczasowych.

Środowisko uruchomieniowe dba również o to, by aplety nie mogły mieć bezpośredniego dostępu do danych innych apletów. Jest to rozwiązane za pomocą mechanizmu kontekstu. Każdy aplet pracuje w ramach swojego kontekstu. W momencie próby dostania się do danych innego apletu (innego kontekstu), firewall Java Card odrzuca takie żądanie. Jest jednak możliwość współdzielenia danych przy pomocy interface'u `Shareable`. Aplet implementujący ten interface może udostępnić innym apletom obiekt, który będzie oferował dane, bądź funkcje, którego mogą być współdzielone. Aplet ma dodatkowo przy tym możliwość zarządzania dostępem do tych danych, gdyż może odrzucić prośbę o udostępnienie współdzielonego obiektu apletom, które nie powinny mieć do niego dostępu.

Kolejnym bardzo istotnym mechanizmem jaki oferuje Java Card RE jest mechanizm atomowości. Pozwala na zapewnienie, że operacja przebiegła niezakłócona w całości. Jest bardzo przydatny podczas operowania na danych trwałych karty, gdyż ogranicza wystąpienie stanu niespójnego w momencie przerwania operacji w wyniku błędu lub interwencji użytkownika karty (np. wyjęcie karty z czytnika).

Najnowsze wersje Java Card RE (od wersji 2.2.2) oferują również komunikację poprzez *RMI* (Remote Method Invocation). Umożliwia w łatwiejsze, niż poprzez komendy APDU, wykorzystanie funkcjonalności apletu.

3.2 Open Platform/Global Platform

Podrozdział ten został opracowany na podstawie [15].

W dzisiejszych czasach możliwości związane w wykorzystaniem kart inteligentnych są na tyle szerokie, że niemożliwym jest, by każdy twórca systemu, z którym karta będzie miała kontakt wgrywał na nią aplikacje służące wyłącznie do komunikacji z jednym systemem. Byłoby to marnotrawstwo pracy programistów jak i ograniczonych zasobów karty, gdyż w tym momencie funkcjonalność poszczególnych aplikacji mogłaby się dublować. W celu uspójnienia interface'u komunikacji oraz zapewnienia dostępności na karcie najczęściej wykorzystywanych mechanizmów powstało Global Platform. Global Platform (GP, kiedyś Open Platform) jest rozwiązaniem pozwalającym wydzielić na karcie aplikacje oferujące współdzieloną funkcjonalność, jak i umożliwić podział odpowiedzialności i praw, jakie przysługują instytucjom związanym z wytworzeniem, personalizacją i użytkowaniem karty.

Architektura GP definiuje dwa specyficzne typy aplikacji: *Security Domain* (SD) oraz *Global Services Application*. Aplikacja pierwszego odpowiada za realizację mechanizmów bezpieczeństwa w grupie innych aplikacji, które są do niej przypisane. Jest to przede wszystkim: zarządzanie kluczami, szyfrowanie, deszyfrowanie, generowanie sygnatury oraz jej weryfikacja. Aplikacje drugiego typu świadczą usługi współdzielone przez inne aplety, bez względu na to, do jakiego SD one należą (np. *Cardholder Verification Method* CVM - realizuje globalny PIN). SD jest apletem, którego funkcjonalność jest ściśle określona, jednakże implementacja pozostawiona instytucji, która chce kartą zarządzać. Są to za-

zwyczaj: wydawca karty, dostawca aplikacji oraz instytucja kontrolująca (określa politykę bezpieczeństwa aplikacji na karcie). Aplet, nie będący SD, w momencie tworzenia ma określone, z usług którego SD ma korzystać. W sytuacji, gdy przychodzi do niego polecenie, za które odpowiedzialność ponosi SD, przekazuje on polecenie do swojego SD i to ono realizuje dalsze obliczenia.

Komunikacja pomiędzy aplikacjami a SD odbywa się poprzez *Trusted Framework*. Jest to rozszerzenie standardowego środowiska uruchomieniowego karty, które zapewnia swoje mechanizmy bezpieczeństwa, realizuje połączenie pomiędzy apletem a przynależnym mu SD.

Inną częścią architektury GP jest Global Platform Environment (OPEN). Jest to opcjonalny komponent, który realizuje mechanizmy, które powinny być implementowane przez Java RE karty. Odpowiedzialny jest on głównie za wybór aktywnej aplikacji, obsługę kanałów logicznych, zarządzanie zawartością karty (m.in. wgrywanie i tworzenie apletów).

Ostatnim elementem GP wartym krótkiego opisu jest Card Manager. Jest to aplet, który realizuje najważniejsze funkcje GP, dlatego też pełni funkcje Global Platform Environment (OPEN), Security Domain wydawcy karty oraz usługa CVM.

3.3 ASN.1 - standard i zastosowanie

ASN.1 (*Abstract Syntax Notation One*) jest standardem opisu struktur danych. Zawiera również zbiór formalnych reguł opisu tych danych. Przyjęty w 1984 roku jako część standardu CCITT X.409:1984, a następnie jako odrębny standard X.208 i X.680. Jest powszechnie stosowany przede wszystkim w telekomunikacji, jak i również w szeroko pojętej informatyce, umożliwiając opisanie struktur składowanych i przesyłanych danych w sposób jednoznaczny. W notacji ASN.1 każda wartość ma swój określony typ. Typy te mogą być proste lub złożone oraz mogą po sobie dziedziczyć. Każdy typ ma swój określony numer *tag*.

Notacja ASN.1 posiada kilka metod kodowania, umożliwiając zapisanie tych samych danych w różnej formie. Są to między innymi:

BER Basic Encoding Rules,

DER Distinguished Encoding Rules,

XER XML Encoding Rules,

PER Packed Encoding Rules.

Metoda BER polega na zapisaniu wartości przy pomocy par bajtów zpisanych w kodowaniu binarnym w postaci: tagu typu danej wartości, jej długości oraz samej wartości (tzw. TLV, *Tag-Length-Value*). Istotną rzeczą jest to, że identyczne dane w kodowaniu BER można zapisać na 3 sposoby. Pierwszy polega na złożeniu tagu, długości w postaci jednego oktetu (jeśli długość jest mniejsza od 128), lub wielu, gdzie pierwszy oznacza ilość

oktetów poświęconych na zapisanie długości a resztą tę długość określa oraz samej wartości. Druga polega na zróżnicowaniu zapisu samej wartości w zależności od jej typu (zapis tagu, jak i długości pozostaje taki jak w pierwszym przypadku). Trzecia natomiast składa się z tagu, oktetu o wartości '80' oraz wartości zakończonej dwoma oktetami '00 00'.

Kodowanie DER jest bardzo podobne do BER i zawiera wszystkie jego reguły, jednakże pozwala jednoznacznie zapisać daną wartość. Określa ono, który ze sposobów zapisu kodowania BER powinien zostać użyty dla wartości określonego typu, bądź długości.

Kodowanie XER określa sposób konwersji pomiędzy danymi w ASN.1 a formatem XML. Dane zakodowane tym sposobem przypominają dane w formacie XML.

Mechanizmy kodowania PER wykorzystują dodatkowe informacje związane z domeną typu, jakiego wartości są kodowane. Pozwala to na zmniejszenie ilość danych potrzebnych do opisania danej struktury. Powoduje to jednak, że strona dekodująca informacje zapisane w kodowaniu PER musi posiadać pełne informacje co do struktury danych, które odtwarza.

Rozdział 4

Projekt jElib

Poniższy rozdział ma na celu omówienie struktury samej aplikacji, jak i przedstawienie procesu implementacji apletu jElib.

4.1 Wymagania funkcjonalne

Wymagania funkcjonalne określają funkcjonalność, którą aplet zobligowany jest udostępniać. Z racji, że aplet nie udostępnia bezpośrednio użytkownikowi żadnej funkcjonalności na poziomie GUI, omówione zostanie tutaj głównie API apletu, które jest udostępniane poprzez komendy APDU.

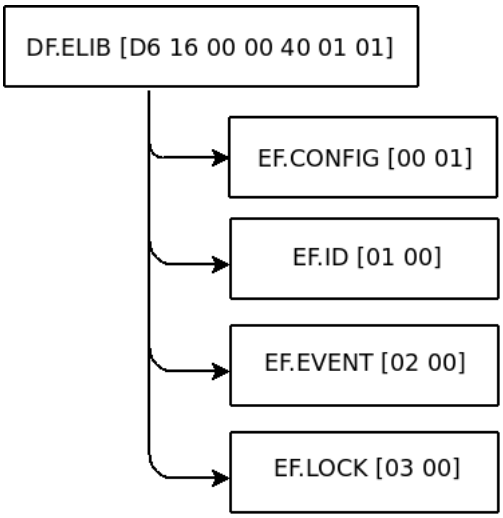
4.1.1 Opis struktury plików jElib

Aby lepiej zrozumieć zdefiniowane API i powiązać konkretne metody z funkcjonalnością apletu należy wpierw przedstawić dane, na których metody te operują.

Projekt jElib jest w założeniu kontynuacją projektu Elib, który w momencie rozpoczęcia realizacji ninejszej pracy magisterskiej nie został jeszcze wdrożony. Początki projektu Elib miały miejsce w pierwszej połowie 2008 roku w Politechnice Poznańskiej. Wtedy to postanowiono wykorzystać ELS jako nośnik informacji studenta z systemów bibliotecznych. Rozwiązanie to miało być realizowane poprzez aplet implementujący obsługę systemu plików zgodną z *ISO 7816-4*. Na kartach, które są wykorzystywane jako ELS producenci dostarczają już takie aplety. Są to ID-One dla kart firmy Oberthur oraz MPCOS dla kart Gemalto. Dane miały być przechowywane w czterech plikach i dostęp do nich miał się odbywać poprzez API zdefiniowane dla dostępu do plików w *ISO 7816-4*.

Z tej też racji, na początku realizacji projektu jElib postanowiono zachować zgodność struktury danych przechowywanych przez aplet z wersją poprzednią, dlatego też jElib dane są składowane również w czterech plikach. W przypadku projektu Elib jak i jElib pliki te znajdują się w katalogu o nazwie *D6160000400101*.

Diagram 4.1 przedstawia strukturę plików, w której są przechowywane dane, zarówno w Elib jak i w jElib.



Rysunek 4.1: Układ plików projektu Elib/jElib.

Charakterystyka plików jElib

Wszystkie pliki wchodzące w skład projektu Elib są plikami o strukturze *TLV* [18]. Stąd też ich zawartość zdefiniowana jest w notacji *ASN.1* (3.3).

EF.CONFIG Plik EF.CONFIG o krótkim identyfikatorze 00 01 zawiera parametry konfiguracyjne aplikacji takie jak rodzaj aplikacji (FILE lub APPLET) oraz jej wersja. Założono, że plik może zawierać do dziesięciu parametrów konfiguracyjnych zapisanych jako para składająca się z nazwy parametru i jego wartości. Maksymalny założony rozmiar pliku to 262 bajty.

```
EF.CONFIG :: = SET SIZE (1..10) OF Param
Param ::= SEQUENCE {
      param_name PrintableString (SIZE (1..10)),
      param_value PrintableString (SIZE (1..10))
}
```

EF.ID Plik EF.ID o krótkim identyfikatorze 01 00 zawiera identyfikatory czytelnika w kolejnych bibliotekach. Przy założeniu, że czytelnik będzie korzystał z 10 bibliotek rozmiar pliku nie przekroczy 412 bajty.

```
EF.ID :: = SET OF Id
Id ::= SEQUENCE {
        library      PrintableString (SIZE (1..15)),
        id           PrintableString (SIZE (1..20))
}
```

Dla studentów korzystających z zasobów bibliotek zrzeszonych w Poznańskiej Fundacji Bibliotek Naukowych wspólny identyfikator biblioteki przyjmuje wartość „PFBN”

EF.EVENT Plik EF.EVENT o identyfikatorze 02 00 zawiera historię wypożyczeń i zwrotów dokonanych w poszczególnych bibliotekach. Założono, że zawarte w pliku informacje dotyczą ostatnich dziesięciu zdarzeń. Maksymalny założony rozmiar pliku to 2152 bajty.

```
EF.EVENT :: = SET SIZE (1..10) OF Event
Event ::= SEQUENCE {
        library      PrintableString (SIZE (1..15)),
        date         UTCTime,
        book         Book,
        event        INTEGER
}
Book ::= SEQUENCE    {
        book         PrintableString (SIZE (1..20)),
        book_title   UTF8String (SIZE (1..50)),
        book_author  UTF8String (SIZE (1..25))
}
```

EF.LOCK Plik EF.LOCK o identyfikatorze 03.00 zawiera informacje przydatne przy obsłudze czytelnika. W strukturze pliku występuje niezależna od biblioteki informacja o założonych blokadach działająca na zasadzie semafora. Biblioteka zakładająca określona blokadę zwiększa wartość licznika na określonej pozycji o jeden, biblioteka zdejmująca blokadę zmniejsza wartość tego licznika o jeden. Plik zawiera również szczegółową informację pochodzącą z danej biblioteki: datę ważności konta, datę pojawienia się czytelnika w bibliotece, szczegółowe informacje o założonych grupach blokad oraz liczbę wypożyczonych książek. Przy korzystaniu z nie więcej niż dziesięciu bibliotek rozmiar pliku nie przekroczy 2387 bajtów.

```
EF.LOCK :: = SEQUENCE {
        lock_vector SEQUENCE SIZE (1..7) OF INTEGER,
        SET SIZE (1..10) OF Lock
}
Lock ::= SEQUENCE {
        library      PrintableString (SIZE (1..15)),
        expire_date  UTCTime,
        use_date     UTCTime,
        lock_info    SET SIZE (1..7) OF LockInfo,
        books_count  INTEGER
}
LockInfo ::= SEQUENCE {
        lock_type    LockType,
        lock_id      INTEGER,
```

```
        lock_date    UTCTime
}
LockType ::= ENUMERATED     {
        nocko (1),          -{ blokada główna („ogólna")
        od (2),             -{ przeterminowanie
        l (3),              -{ (lost) zagubienie materiałów
        damage (4),         -{ zwrócił uszkodzone materiały
        suspens (5),        -{ zawieszenie
        fee (6),            -{ nieuregulowana opłata
        fine (7)            -{ nieuregulowana kara
}
```

4.1.2 API zgodne ze standardami ISO 7816-4 i ISO 7816-9

Bazując na założeniu, że aplet jElib ma być kompatybilny wstecz z rozwiązaniem Elib postanowiono w zakres oferowanej funkcjonalności apletu włączyć metody do zarządzania plikami i ich treścią zdefiniowane w *ISO 7816-4* oraz *ISO 7816-9*. Umożliwiłoby to podmienienie realizacji Elib apletem jElib. W trakcie rozwoju projektu pojawiły się komplikacje, które zmieniły podejście do realizowania przez aplet funkcjonalności z zakresu standardów *ISO 7816-4* i *ISO 7816-9*. Okazało się, że nie można zainstalować apletu jElib na kartach Gemalto. W związku z tym ustalono wraz z twórcami systemu Horizon, że część implementowana po stronie biblioteki będzie przystosowany zarówno do obsługi kart, gdzie funkcjonalność będzie realizowana przez aplety producenta karty, jak i kart z zainstalowanym apletem jElib z dedykowanymi metodami.

Nie zmienia to faktu, że jednak API zgodne ze standardami powstało i może, lecz nie musi być włączone do ostatecznej wersji apletu. Jego krótka charakterystyka przedstawiona jest w tabeli 4.1.

Bajt *CLA*	Bajt *INS*	Nazwa metody	Nr standardu
0x00	0xA4	SELECT	*ISO 7816-4*
	0x0F	ERASE BINARY	
	0xA1	SEARCH BINARY	
	0xB1	READ BINARY	
	0xD1	WRITE BINARY	
	0xD7	UPDATE BINARY	
	0xE0	CREATE FILE	*ISO 7816-9*
	0xE4	DELETE FILE	

Tablica 4.1: Implementowane komendy, zgodne ze standardem *ISO 7816-4* lub *ISO 7816-9*

Dokładny opis komend znajduje się w załączniku (6.1 i 6.2).

4.1.3 API dedykowane dla jElib

Stworzenie apletu pozwoliło lepiej wykorzystać możliwości drzemiące w kartach mikroprocesorowych. Poza zwykłymi funkcjami zapisu i odczytu można teraz przetwarzać dane bezpośrednio na karcie. Są to operacje o małej złożoności, ze względu na ograniczoną ilość zasobów i wydajność mikroprocesora, pozwalają jednak np. poprzez wstępną obróbkę danych ograniczyć liczbę operacji I/O i tym samym przyspieszyć komunikację karta-czytnik. W przypadku jElib, dla pracowników bibliotek, którzy w ciągu dnia będą stykali się z kartą kilkadziesiąt lub więcej razy daje to duże oszczędności czasu. Ponadto można już na poziomie karty wprowadzić mechanizmy weryfikacji danych dostarczanych do nośnika. Z tych właśnie powodów powstało dedykowane API. Przeniosło wiele operacji na kartę, między innymi zarządzanie rozmieszczeniem danych w pliku. Dzięki temu użytkownik nie musi się zastanawiać, w którym dokładnie miejscu ma rozpocząć zapisywanie pliku, by zachować jego logiczną strukturę.

Z powyższych przesłanek powstał zestaw funkcji, których opracowanie rozpoczęło się od zdefiniowania funkcjonalności potrzebnej bibliotekom. Ponadto, w związku z planami poszerzenia funkcjonalności kart w Politechnice Poznańskiej, poszerzono ten zbiór o wiele dodatkowych. Tym samym, funkcjonalność dedykowana bezpośrednio bibliotekom stanowi ok. 25% zaimplementowanej.

Stworzona funkcjonalność przedstawiona jest w tabeli 4.2.

Bajt *CLA*	Bajt *INS*	Nazwa metody	Plik
0xC0	0x30	INITIALIZE	WSZYSTKIE
	0x31	READ ALL	
	0x33	GET ENTRY INDEX	EF.CONFIG
	0x34	UPDATE ENTRY VALUE	EF.ID
	0x35	COUNT ENTRIES	
	0x36	GET ENTRY INDEX	
	0x37	ADD ENTRY	
	0x38	REMOVE ENTRY	
	0x39	GET VERSION	EF.CONFIG
	0x3A	READ EVENT FOR LIBRARY	EF.EVENT
	0x3B	ADD EVENT	
	0x3C	READ VECTOR	EF.LOCK
	0x3E	READ ALL RECORDS	
	0x3F	READ RECORD OF LIBRARY	
	0x40	INIT LIBRARY RECORD	
	0x41	ADD LOCK TO LIBRARY RECORD	
	0x42	UPDATE LIBRARY RECORD	
	0x43	DELETE LOCK FROM LIBRARY RECORD	
	0x44	READ LOCK FIELD	

Bajt *CLA*	Bajt *INS*	Nazwa metody	Plik
0xC0	0x45	UPDATE LOCK FIELD	EF.LOCK

Tablica 4.2: Implementowane komendy dedykowane dla jElib

Dokładny opis komend znajduje się w punkcie 6.3.

4.2 Wymagania pozafunkcjonalne

Poniżej zdefiniowane są wymagania pozafunkcjonalne, które powstały w trakcie dyskusji osób powiązanych z projektem. Wymagania te nie dotyczą funkcjonalności apletu, tylko jego możliwości, charakteryzują *w jaki sposób* aplet ma realizować swoją funkcjonalność.

Pojedyncze litery przy wymaganiu określają jego istotność dla projektu (**W** - wysoka, **Ś** - średnia, **N** - niska).

Dokładność

W Aplet ma przechowywać dane w takiej formie w jakiej je otrzymał oraz przy odczycie zwracać je niezmodyfikowane.

W Aplet przy ustalonych danych powinien odpowiadać w sposób powtarzalny.

Bezpieczeństwo

W Aplet powinien uniemożliwiać nieautoryzowany zapis.

Ś Aplet powinien rozróżniać użytkowników (może to byś zapewnione przez aplet Card Managera).

Tolerancja na błędy

N W razie wystąpienia nieodwracalnego błędu aplet powinien informować o takim stanie. Nie musi dalej realizować swoich funkcji.

Odzyskiwalność

W W razie awarii aplet powinien dać się wyładować i zainstalować na nowo.

Ś Aplet powinien mieć możliwość przywrócenia początkowego stanu systemu plików.

Wymagania czasowe

Ś Aplet powinien odpowiadać na wywołanie funkcji w ciągu 1 minuty.

Wykorzystanie zasobów

W Aplet po instalacji i wypełnieniu jego systemu plików wszystkimi możliwymi (dopuszczalnymi) danymi powinien zmieścić się na karcie wraz z instancją ELS.

W Aplet powinien działać na zasobach oferowanych przez karty wymienione w sekcji 'Adaptowalność'.

Analizowalność

W Wyjątki/Odpowiedzi zwracane przez aplet powinny jednoznacznie i jasno określać stan apletu / zakończenia operacji.

Modyfikowalność

N Aplet powinien zapewniać możliwość dodania obsługi danych nowego typu (innego niż TLV).

N Powinna być możliwa zmiana wersji aplikacji / struktury plików bez utraty danych.

Adaptowalność

W Aplet musi dać się zainstalować na kartach z systemem Java Card przynajmniej w wersji 2.1 oraz systemem Global Platform w wersji 2.1.

Instalowalność

W Jeśli taka funkcjonalność nie będzie zapewniona, to aplet musi dać się zainstalować przy pomocy narzędzi z Java Card Development Kit firmy Sun.

Współistnienie

W Aplet musi działać na jednej karcie wraz z apletem obsługującym ELS oraz w sposób oczywisty nie ograniczać pracy, bądź możliwości instalacji innych apletów.

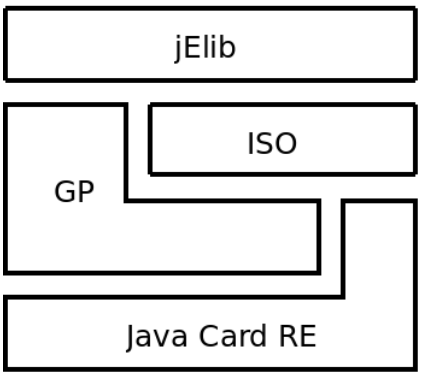
W Aplet powinien umożliwiać działanie systemów bibliotecznych operujących jedynie na wersji plikowej.

4.3 Koncepcja rozwiązania

Koncepcja rozwiązania opiera się na wykorzystaniu w części dedykowanej jElib metod i struktur danych tworzonych z myślą o implementacji standardów. Pozwala to na zachowanie wewnętrznej zgodności i ujednolicenia sposobu przechowywania danych. W tym momencie możliwe jest, by do plików zarządzanych poprzez dedykowane API odwołać się również poprzez metody ze standardu. Jest to swojego rodzaju zabezpieczenie przed

niespodziewanymi błędami w implementacji dedykowanej części apletu. Ponadto dzięki oferowanym przez producentów kart dostępie do API Global Platform można cały mechanizm zarządzania bezpieczeństwem przesyłanych i składowanych danych przenieść poza aplet. Wykorzystujemy w tym momencie implementację komunikacji poprzez tzw. bezpieczny kanał oraz weryfikacji danych oferowaną przez domyślne *Security Domain*, czyli Card Managera.

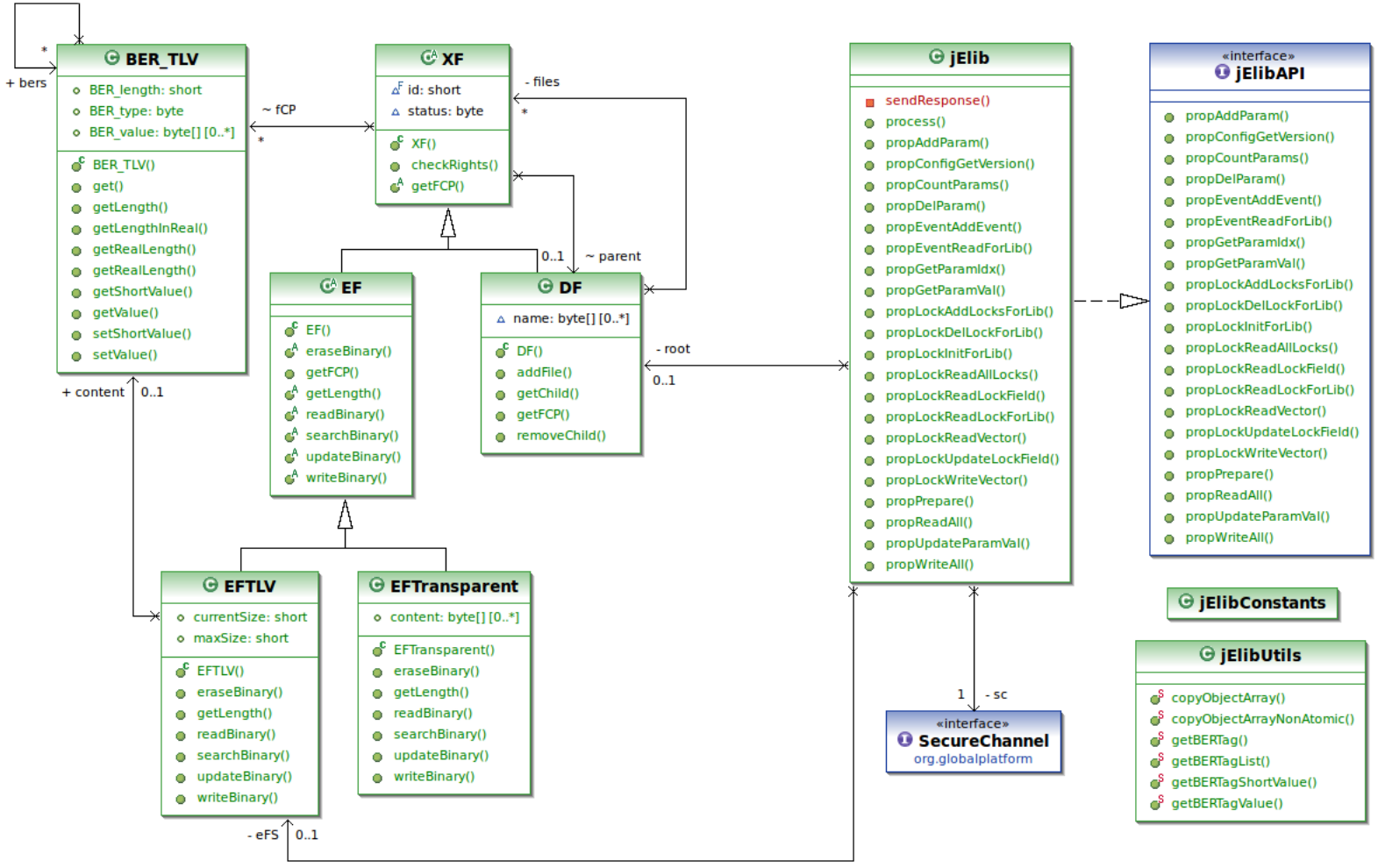
Rysunek 4.2 przedstawia schemat zależności poszczególnych modułów apletu jElib.



Rysunek 4.2: Zależność modułów w jElib

Ponieważ cały aplet nie jest duża aplikacją nie posiada on również wielu klas. Rysunek 4.3 prezentuje schemat klas apletu. Jest on nieznacznie uproszczony, m.in. nie zawiera niektórych pól statycznych.

Klasy na schemacie są tak rozmieszczone, że można je wizualnie podzielić na dwie części. Klasy znajdujące się po lewej stronie realizują funkcjonalność związaną z systemem plików implementowanym przez aplet. Przez to rozumiane jest np. budowanie struktury plików/katalogów, przechowywanie danych, operowanie na nich. Natomiast prawa część schematu odpowiada za operacje na tej strukturze. Przede wszystkim za manipulowanie plikami jako całością, weryfikacja danych zapisywanych do pliku oraz udostępnianie z plików danych odczytanych.



Rysunek 4.3: Diagram klas jElib

Mechanizmy struktury plików znajdują się w klasach*XF, EF, DF, EFTLV, EFTransparent i BER_TLV*, z czego tylko ostatnia nie jest reprezentacją pliku. Poniżej każda z klas została pokrótce opisana:

BER_TLV obiekty tej klasy reprezentują pojedynczą strukturę *ASN.1* (3.3), m.in. jej tag, długość oraz wartość lub struktury zagnieżdżone,

XF klasa abstrakcyjna definiująca obiekt struktury plików (plik bądź katalog), zawiera jego identyfikator (2 bajty) oraz informacje *FCI*. Dodatkowo zawiera implementację metody weryfikującej możliwość operowania na pliku, na podstawie atrybutów bezpiecznego kanału opisanej w rozdziale 4.4.3,

EF klasa abstrakcyjna reprezentują plik, definiuje interfejs metod operowania na pliku,

DF klasa reprezentująca katalog (plik DF), przechowuje nazwę pliku oraz umożliwia dodawanie, usuwanie plików (EF i DF) z poziomu dzieci danego pliku DF oraz wyszukiwanie jego potomków po identyfikatorze,

EFTransparent klasa reprezentująca plik prosty, przechowuje dane jako tablicę bajtów oraz implementuje interfejs określony w klasie *EF*,

EFTLV klasa reprezentująca plik struktur *TLV*, przechowuje dane jako tablicę obiektów *BER_TLV* oraz implementuje interfejs określony w klasie *EF*.

Reprezentacja systemu plików na karcie, zgodnie z opisem w standardzie *ISO 7816-4* ma strukturę drzewa. Korzeniem jest tzw. root (plik MF) o identyfikatorze 0x3f00. Jest to plik najwyższy w hierarchii oraz istnieje tylko jeden. Pod względem implementacji jest to plik DF. Stąd też zachowuje się tak jak każdy inny plik DF, przy czym jako jedyny nie może zostać usunięty. Pliki DF posiadają tzw. 'dzieci', czyli plik będące w strukturze poniżej ich samych. Takie dziecko reprezentowane jest przez klasę *XF*, tak, by mógł nim być zarówno plik EF, jak i DF. Każde dziecko ma również wskazanie na swojego rodzica. Pozwala to na przechodzenie po strukturze plików w kierunku wyższego poziomu zagłębienia, oraz w kierunku niższego poziomu zagłębienia, czyli aż do root'a.

Pliki DF i EF dzielą tylko dwie funkcje: *checkRights*, zdefiniowaną w klasie nadrzędnej, *XF*, i *getFCP*. Druga metoda zwraca, na podstawie informacji *FCP*, odpowiedź dla polecenia *SELECT*. Dla pliku (EF) i katalogu (DF) może ona być inna, stąd też możliwość jej oddzielnej implementacji.

Klasa *EF* dostarcza uniwersalną definicję metod operowania na pliku. Są to wszystkie metody opisane w standardzie *ISO 7816-4*. Każdy nowy typ pliku dodany do apletu musi implementować jej interfejs. Klasy *EFTransparent* i *EFTLV* implementują te metody na swój sposób. Pomijając implementację sposobu operowania na danych pliku, który jest z oczywistych względów różny dla obu typów, klasa *EFTLV* wymaga podania dodatkowych informacji w przypadku operacji zapisu do pliku *WRITE BINARY*. Ta metoda nie jest

opisana w standardzie dla plików typu innego niż prosty. Stąd też implementacja tej metody w aplecie jElib wymaga podania w polu danych dodatkowego parametru, którym jest stopień zagłębienia na jakim mają być zapisane nowe dane. Jest to opisane w załączniku 1 (6.1).

Prawa strona schematu przedstawia natomiast część odpowiedzialną bezpośrednio za funkcjonalność dedykowaną jElib. Jest ona realizowana przez dwie klasy:

jElibAPI interfejs określający metody, jakie musi implementować aplet realizujący funkcjonalność jElib

jElib klasa apletu, realizująca funkcje Java Card RE oraz te, zdefiniowane w *jElibAPI*

Klasa *jElib*, jako implementująca klasę *Applet* z Java Card RE, zajmuje się obsługą poleceń APDU wysyłanych do karty. To ona odpowiada za przekazanie kontroli do odpowiedniej metody swojej, bądź klas struktury plików. Ona również implementuje funkcje dedykowane jElib. W dużej części ich zadanie sprowadza się do walidacji danych wejściowych, ich ewentualnej modyfikacji oraz umieszczenia ich w odpowiednim miejscu w pliku, w zależności od kontekstu. Do celów poprawnej walidacji i uproszczenia operacji na plikach musi być znana struktura plików przechowywanych przez aplet. To dzięki temu możliwa jest implementacja funkcji dedykowanych, które w założeniu mają usprawnić operowanie danymi i uprościć interfejs dostępu do nich. To również w tej klasie zrealizowana jest obsługa mechanizmów Global Platform (3.1.2), m.in. bezpiecznego kanału.

Dodatkowo stworzone zostały dwie klasy, które są klasami pomocniczymi. Pierwsza, *jElibConstants*, przechowuje wyłącznie stałe wykorzystywane w aplikacji. Zarówno wartości bajtu *INS* jak i tagi struktur *ASN.1* czy ich długość zdefiniowana w dokumentacji do plików Elib (4.1.1). Natomiast klasa *jElibUtils* pełni rolę 'zestawu pomocnych narzędzi'. Znajdują się tam metody, których brak w Java Card RE był odczuwalny w trakcie implementacji (*copyObjectArray*), bądź pozwoliły usprawnić operację na strukturach *ASN.1*. Do tych drugich zaliczając się:

getBERTagList z tablicy bajtów o poprawnej strukturze tworzy tablicę obiektów *BER_TLV*

getBERTag wśród tablicy obiektów *BER_TLV* znajduje pierwszy, który ma tag o zadanej wartości

getBERTagValue wśród tablicy obiektów *BER_TLV* znajduje pierwszy, który ma tag o zadanej wartości i zwraca jego wartość

getBERTagShortValue wśród tablicy obiektów *BER_TLV* znajduje pierwszy, który ma tag o zadanej wartości i zwraca jego wartość w postaci wartości *short*

4.4 Implementacja

4.4.1 Przygotowanie platformy deweloperskiej

Projekt realizowany był na komputerze klasy PC, pracującym w architekturze 32-bitowej, korzystającym z systemu GNU Linux (dystrybucja Ubuntu 8.10 Intrepid Ibex). Wykorzystywanym czytnikiem kart był na początku Schlumberger Reflex USB, jednakże okazało się, że przekłamuje on transmisję i niemożliwym było wgranie apletu na kartę. Po ujawnieniu się tego błędu czytnik został wymieniony na CryptoTech SCR 3311 (podłączany również przez port USB). Do celów testowych dostarczono 2 karty firmy Oberthur, model Cosmo 64.

Konfiguracja platformy sprzętowej w systemie GNU Linux

W systemie GNU Linux obsługa kart realizowana jest zgodnie z *PC/SC* (ang. *Personal Computer/Smart Card*). PC/SC jest frameworkiem opracowanym przez grupę przedsiębiorstw (m.in. Apple Inc., Microsoft Corporation, Infineon, Gemalto) mającym na celu integrację kart mikroprocesorowych ze środowiskiem komputerowym. Darmowa implementacja PC/SC, *PC/SC Lite*, dostępna jest dla większości istniejących dzisiaj dystrybucji GNU Linux. Poniżej zaprezentowano instalację oraz uruchomienie PC/SC na przykładzie dystrybucji Ubuntu (wersja 8.10) bazującej na dystrybucji Debian.

W celu zainstalowania PC/SC Lite należy wykonać w terminalu komendę:

```
user@home# apt-get install pcscd
```

Jeśli aplikacja ta nie jest dostępna, można ją pobrać ze strony [3] i samemu skompilować. Dodatkowo w przypadku czytnika firmy Schlumberger była potrzeba zainstalowania sterowników, które można znaleźć na stronie [7].

Konfiguracja platformy sprzętowej w systemie Windows

W związku z faktem, że aplikacja producenta karty, do zarządzania pakietami, apletami oraz kluczami na karcie napisana była pod systemy z rodziny Windows część testów przeprowadzano również pod tym systemem. Dokładniej, był to wirtualizowany w VirtualBox 2.2 system Windows XP SP2.

Systemy rodziny Windows (począwszy od wersji 2000) posiadają wbudowaną obsługę kart zgodną z PC/SC. Należało jednak zainstalować sterowniki do konkretnego czytnika. Te, dla modelu CryptoTech SCR 3311, można pobrać ze strony [5].

Konfiguracja oprogramowania

Podstawowym narzędziem do pisania kodu, które było używane w projekcie jElib było *Eclipse IDE* [6]. Jest to bardzo rozbudowane środowisko programistyczne, ułatwiające tworzenie aplikacji przede wszystkim w Javie, ale również w C/C++, PHP i wielu innych.

Do kompilacji kodu Java wykorzystującego Java Card RE potrzebne były również biblioteki Java udostępniające funkcjonalność Java Card RE w wersji 2.2.2. Te są dostarczone przez Sun Microsystems jako Java Card Development Kit (JCDK) i dostępne do pobrania ze strony [2]. Zostały one dołączone do projektu.

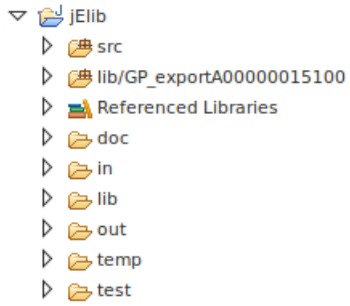
Dodatkowo biblioteki związane z Global Platform były również częścią projektu i zostały dostarczone przez opiekuna.

Struktura projektu

Ważnym jest by kompilacja kodu następowała w trybie zgodnosci z Java w wersji maksymalnie 1.4. Jest to potrzebne do prawidłowego działania pozostałych aplikacji wchodzących w skład środowiska programistycznego. Tryb zgodności można ustawić we właściwościach projektu, w zakładce **Java Compiler**, pole **Compiler compliance level**.

Projekt, który został użyty do przechowywania kodu w Eclipse, to zwykły *Java project*. Sam kod znajduje się w katalogu *src*, skompilowane klasy w *bin*, a biblioteki w katalogu *lib*. Ponadto projekt zawiera jeszcze katalog *doc* z dokumentacją Javadoc, *in* przeznaczony na skrypty testujące, *out*, w którym tworzone są pliki wynikowe konwersji[1], *temp*, na pliki tymczasowe, potrzebne podczas kompilacji oraz *test* zawierający skrypty do narzędzia GPShell (4.4.1).

Drzewo projektu wygląda, jak na ekranie poniżej.



Rysunek 4.4: Drzewo projektu jElib w Eclipse

Plik konfiguracyjny Przed przystąpieniem do programowania należy skonfigurować projekt. Do tego celu służy plik konfiguracyjny *build.properties*. Jego struktura przedstawia się następująco:

```
applet.name=jElib
version=1.0
package.aid=0xD6:0x16:0x00:0x00:0x40
```

[1]W Java Card skompilowane pliki *.class* są konwertowane do pliku .cap (ang. *Converted Applet* - zawiera pliki wgrywane na kartę), .exp (ang. *Export* - interfejs dla innych apletów wykorzystujących funkcjonalność konwertowanego oraz .jca (ang. *Java Card Assembly* - plik ułatwiający debugging apletu)

```
package.aid.length.hex=5
applet.aid=0xD6:0x16:0x00:0x00:0x40:0x01:0x01
applet.aid.length.hex=7
package.name=put.elib
jcdk.path=${basedir}/lib/java_card_kit-2_2_1
jc.export.path=${basedir}/lib/java_card_kit-2_2_1/api_export_files
gp.export.path=${basedir}/lib/GP_exportA00000015100
output.dir=./out
script.file=${basedir}/in/script-test_prop
```

Plik ten zawiera zmienne, które są z kolei używane w skrypcie *Ant* opisanym później (4.4.1). Wartości, jak powyżej, umożliwiają kompilację i konwersję zgodną z wymaganiami oraz strukturą projektu, także nie ma potrzeby ich zmiany. Poszczególne zmienne oznaczają:

applet.name nazwa apletu

version wersja apletu

package.aid AID pakietu

package.aid.length długość, ilość bajtów, potrzebnych do zapisania AID pakietu

applet.aid AID instancji apletu

applet.aid długość, ilość bajtów, potrzebnych do zapisania AID apletu

package.name nazwa pakietu języka Java, w którym przechowywane są klasy apletu

jcdk.path ścieżka do JCDK (nie powinna zawierać spacji)

jc.export.path ścieżka do plików exportu (.exp) JCDK (nie powinna zawierać spacji, nie powinna być zmieniana)

jc.export.path ścieżka do plików exportu (.exp) Global Platform (nie powinna zawierać spacji, nie powinna być zmieniana)

output.dir ścieżka do katalogu, wewnątrz struktury projektu, w którym będą składowane pliki wynikowe

script.file ścieżka do pliku, wewnątrz struktury projektu, który zawiera komendy testujące (zdefiniowane przez programistę) aplet

Skrypt *Ant* W celu usprawnienia procesu kompilacji, konwersji i testowania został napisany skrypt *Ant*, *build.xml*, który wykonywał automatycznie poszczególne sekwencje poleceń potrzebnych do każdego z tych zadań. Operacje, które udostępnia skrypt to:

clear czyści foldery out i temp z plików

convert konwertuje pliki wynikowe *.class* z katalogu *bin* do plików *.cap*, *.exp* oraz *.jca* i umieszcza w katalogu *out*

convert.create.config tworzy plik konfiguracyjny konwersji

help domyślna operacja, wyświetla listę i opis operacji skryptu

prepare tworzy domyślną strukturę katalogów projektu

prepare.api.export.files operacja pomocnicza, umieszcza pliki *.exp* JCDK oraz Global Platform w jednym miejscu

run.scripts uruchamia skrypty testujące

scriptgen tworzy skrypty instalujące aplet, na bazie pliku *.cap*

scriptgen-run.scripts wykonuje zarówno konwersję jak i tworzenie skryptów oraz uruchamia symulator (4.4.1) i testuje aplet

scriptgen.full tworzy skrypty instalujące oraz testujące aplet, na bazie pliku *.cap* oraz skryptu wskazanego przez zmienną *script.file* z pliku konfiguracyjnego

W celu skorzystania z pliku skryptu należy w Eclipse wybrać z menu **Window ->Show View ->Ant** i następnie z okna *Package Explorer* przeciągnąć plik *build.xml* do nowo otwartego widoku *Ant*. W celu przetestowania apletu wystarczy uruchomić operację *scriptgen-run.scripts*. W widoku *Console* pojawią się rezultaty wykonanych poleceń.

By wyłącznie utworzyć plik .cap (wgrywany na kartę) wystarczy uruchomić operację *convert*. Plik .cap znajdziemy w katalogu *out*, wewnątrz struktury odpowiadającej nazwie pakietu, w którym klasy apletu się znajdują i dodatkowo katalogu *javacard*.

Aplikacje Java Card Development Kit W skład JCDK wchodzą aplikacje niezbędne do przygotowania i przetestowania apletu Java Card. Najistotniejsze, które były wykorzystywane podczas procesu tworzenia *jElib* to:

- apdutool,
- converter,
- cref,
- jcwde,

- scriptgen.

Najistotniejszym narzędziem w powyższym zestawie jest *cref*. Nazwa wywodzi się od 'C-language Java Card RE' lub może bardziej od 'C reference Java Card RE', czyli referencyjna implementacja Java Card RE w języku C. W przeciwieństwie do jcwde (4.4.1) jest to symulator, który zachowuje się prawie identycznie jak prawdziwa karta. Przy jego pomocy istnieje możliwość zainstalowania wielu apletów jednocześnie i przechowania tak przygotowanej struktury apletów w pliku w celu późniejszego dalszego użycia. Dodatkowo umożliwia korzystanie z kanałów logicznych, czy usuwania obiektów.

JCWDE jest symulatorem uruchamianym wraz z pojedynczym apletem. Nie ma więc możliwości testowania współistniejących apletów. Nie obsługuje on również wielu innych funkcji dostępnych w *cref*. Szczegółowe różnice dostępne są w dokumentacji JCDK [14].

Scriptgen bazując na podstawie pliku .cap tworzy zestaw komend APDU, które wgrywają aplet na kartę oraz inicjalizują jego instancję.

Apdutool służy do wymiany komend APDU z apletem. Komendy są pobierane ze skryptów stworzonych przez narzędzie *scriptgen*. Można do tych skryptów dodać własne komendy, jeśli tylko będą trzymały się wzoru komend wynikowych *scriptgen*.

Ostatnim narzędziem, które było wykorzystywane jest *convert*. Jego zadanie polega na kowersji plików *.class* do plików *.cap*, *.exp* i *.jca*.

Aplikacje dodatkowe Poza aplikacjami z Java Card Development Kit wykorzystywane były również dwie inne.

Application Loader jest aplikacją middleware, stworzoną przez producenta kart, firmę Oberthur. Jej zadaniem jest zarządzanie apletami i pakietami na karcie, zarządzanie kluczami karty, wykonywanie pojedynczych komend APDU. Była używana przede wszystkim w celu wgrywania/usuwania apletu. Dostarczona została przez opiekuna.

GPShell jest otwartym projektem, mającym za zadanie stworzenie klienta kart mikroprocesorowych umożliwiającego korzystanie z funkcjonalności oferowanej przez Global Platform. Tak jak *apdutool* obsługuje skrypty, lecz ich formaty są niezgodne pomiędzy sobą. Główną zaletą GPShell jest możliwość przesyłania danych do karty poprzez bezpieczny kanał, czyli przesyłane dane zawierały dodatkowo sumę kontrolną potrzebną do ich weryfikacji, bądź mogły być całkowicie szyfrowane. Aplikacja okazała się bardzo pomocna, jednakże w trakcie pracy pojawiały się problemy, gdyż aplikacja nie zawsze potrafiła utworzyć bezpieczny kanał (błędnie wyliczała dane potrzebne do jego inicjalizacji). GPShell jest dostępny do pobrania ze strony [4].

4.4.2 Realizacja transformacji modelu w ASN.1 do modelu obiektowego

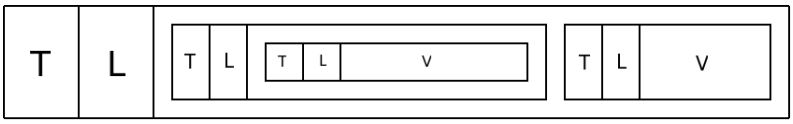
Jak wspomniano w rozdziale 4.1.1 pliki jElib są plikami struktur *TLV*. To oznacza, że dostęp do nich nie jest tak swobodny jak w przypadku plików prostych. Ponadto prze-

chowywanie tagów *TLV* w postaci łańcucha jest bardzo niewydajne. Za każdym razem sprawdzenie długości wartości tagu wiązałoby się z iteracją po tablicy i powtarzającymi się obliczeniami. Stąd też zdecydowano się, by dane zawarte w tagach *TLV* ułożone były w drzewiaste struktury, tak jak to ma miejsce w rzeczywistości.

Numer tagu jaki można nadać strukturze nie ma ograniczeń, prócz takiego, że musi być 1-bajtowy. Można więc stosować tagi dowolnej klasy.

Długość wartości może być zdefiniowana dwojako. Może być to pojedyńczy bajt, jeśli długość wartości tagu nie przekracza 127 bajtów. Najstarszy bit b_8 bajtu długości ma wtedy wartość '0'. Bajt długości ma wartość '01111111'. W drugim przypadku bit b_8 ma wartość '1' a reszta bitów określa liczbę kolejnych bajtów, na których zakodowana jest długość wartości tagu. W tej postaci długość może być niezwykłe duża, jednak ze względu na zmniejszenie złożoności oraz małą ilość dostępnej pamięci na karcie implementacja obsługuje tagi o maksymalnej długości wartości 65536 bajtów (2 bajty). Bajty określające długość wyglądają wtedy następująco: '10000010 11111111 11111111'. W tej formie można również zakodować długość wartości krótszej od 128 bajtów.

Z racji, że tagi tworzą strukturę drzewiastą, wynika, że każdy tag może wewnątrz swojej wartości zawierać inne tagi, lub też może przechowywać wartość jako tablicę bajtów. Sytuacje te są jednak rozłączne. Jeśli wartością jest tablica bajtów, to tag nie może już mieć zagnieżdżonej struktury wewnątrz wartości, i odwrotnie. Poniżej zilustrowana jest ta zależność.



Rysunek 4.5: Przykład struktury *TLV*.

Taka forma przechowywania tagów *TLV* bardzo upraszcza manipulowanie tymi strukturami. Odwołanie się do właściwości tagu ma złożoność liniową, a co za tym idzie ograniczamy zbędny narzut odczytywania i interpretacji danych w postaci liniowej.

W celu usprawnienia pracy ze strukturą tagów stworzono kilka statycznych metod odpowiedzialnych za: konwersję ze struktury prostej do struktury *TLV* (również drzewiastej), wyszukiwanie w drzewie tagów tagu o określonym numerze oraz zwracanie wartości tago o określonym numerze. Dzięki nim operowanie na plikach struktur *TLV* okazało się prostsze, niż w przypadku plików prostych.

Podstawowym założeniem implementacji było operowanie na plikach jako obiektach. Dzięki zaimplementowanemu wyżej opisanemu przekształceniu było to możliwe. Przyjęto również, że pliki EF.CONFIG i EF.ID będą podlegały tym samym operacjom. Wynikało to z identycznej struktury logicznej obu plików (oba przypominają mapę, EF.CONFIG - nazwa parametru, wartość, a EF.ID - nazwa biblioteki, identyfikator studenta). Dzięki temu uniknięto powielania kodu.

4.4.3 Mechanizmy bezpieczeństwa

W związku z faktem, że aplet jElib będzie wykorzystywany w środowisku pozauczelnianym, nad którym uczelnia nie ma kontroli, należało zapewnić odpowiedni poziom bezpieczeństwa danych w nim przechowywanych. Dotyczy to zarówno zabezpieczenia przed nieautoryzowanym zapisem, jak i przed nieautoryzowanym odczytem tych danych. Dzięki temu, będzie można mieć pewność, że dane udostępniane przez aplet pochodzą z zaufanych źródeł oraz to, że osoby niepowołane (np. studenci) nie będą mieli dostępu do informacji niejawnych. Ważne jest również, by weryfikować poprawność przesłanych na kartę danych. Pozwoli to wykluczyć zapisywanie danych, które uległy przekłamaniu podczas transmisji. W tym celu wykorzystano mechanizmy oferowane przez Global Platform oraz własne rozwiązanie opierające się na Java Card RE.

Global Platform

Funkcjonalność Global Platform odpowiada za dwa aspekty bezpieczeństwa w aplecie jElib. Są to:

- weryfikacja przesyłanych danych,
- kontrola dostępu do plików na poziomie operacji.

Sprawdzanie poprawności danych odbywa się poprzez mechanizm bezpiecznego kanału (ang. *secure channel*). Zasada jego działania polega na spreparowaniu polecenia APDU przesyłanego do lub z karty tak, by zawierało ono dodatkowe informacje kontrolne, bądź treść w postaci zaszyfrowanej (zamiast jawnej). Security Domain (SD) na karcie, dzięki parametrom kanału potrafi określić czy wiadomość została wysłana przez wiarygodne źródło oraz ją odszyfrować. By ustanowić bezpieczny kanał należy znać klucz przechowywany w SD. Gdy strona próbująca ustanowić bezpieczny kanał go nie zna, lub jest on błędny, wtedy SD odpowiada błędem. Dzięki temu, wymuszając istnienie bezpiecznego kanału przed wykonaniem operacji, możemy ograniczyć dostęp do metody apletu tylko do tych klientów, którzy znają odpowiedni klucz. SD zachowuje się tak samo, gdy suma kontrolna przesłana z wiadomością nie zgadza się z jej treścią. Zabezpiecza to aplet przed sytuacją, w której treść wiadomości zostaje zmanipulowana w trakcie transmisji (przez błąd lub ingerencję stron trzecich).

Kontrola dostępu do plików

Powyżej omówiono sposób prewencji przed wykonaniem operacji, gdy odpowiednie wymaganie bezpieczeństwa nie jest spełnione (brak bezpiecznego kanału). To ogranicza dostęp do operacji jako takiej, jednakże nie daje możliwości nadania prawa do tej operacji konkretnemu klientowi.

By rozwiązać ten problem skorzystano z autorskiego rozwiązania. Wykorzystuje ono możliwości oferowane przez standard *ISO 7816-4*. Definiuje on, że w strukturze FCI pliku

można przechowywać informacje dotyczące bezpieczeństwa pliku (tag 'A1'). W przypadku jElib zapisywany jest tam numer wersji klucza, przy pomocy którego musi zostać utworzony bezpieczny kanał przed dostępem do tego pliku. W tym momencie tylko klient używający klucza o poprawnym numerze wersji ma dostęp do danych w pliku.

W przypadku jElib jeden numer wersji klucza jest współdzielony przez wszystkie pliki, gdyż wszystkie biblioteki korzystające z apletu są traktowane jako klient o tych samych prawach.

4.5 Testy

Testy z punktu widzenia jakości oprogramowania są bardzo ważne, gdyż dają informację na ile oprogramowanie zachowuje się odpowiednio do zadanych argumentów. Aplet był testowany na dwa sposoby:

1. na symulatorze,
2. na prawdziwej karcie.

Symulatorem używanym do testowania był omówiony wcześniej *cref* (4.4.1). Było to główne narzędzie do testowania podczas rozwijania apletu. Dzięki niemu każda wprowadzona poprawka mogła zostać od razu sprawdzona, bez konieczności operacji na sprzęcie i czasochłonnego wgrywania apletu na kartę. Atutem symulatora było również to, że można go było uruchomić z poziomu środowiska *Eclipse*. Komunikacja pomiędzy narzędziem *apdutool* (4.4.1) a *cref* przebiegała zdecydowanie szybciej, niż w przypadku czytnika i karty. Dodatkowo w przypadku wystąpienia wyjątku Java *cref* informuje o tym użytkownika poprzez odpowiedni wpis w konsoli wynikowej. Pozwalało to na uzyskanie bardziej szczegółowych danych z procesu redukowania liczby błędów w oprogramowaniu niż tylko kod statusu. Istotną rzeczą była również możliwość zobaczenia zestawienia zajętości pamięci przez aplet, pamięci trwałej (*EEPROM*, jak i ulotnej *RAM*). Dawało to pogląd na wielkość apletu i dzięki temu można było już we wczesnej fazie rozwoju starać się optymalizować zarządzanie pamięcią apletu.

Druga faza testów funkcjonalności odbywała się już na karcie firmy Oberthur. Do niej użyto oprogramowania Application Loader (4.4.1) oraz GPShell (4.4.1). Application Loader pozwalał na wgranie nowej wersji apletu/usunięcie starej i wykonanie poleceń APDU bez bezpiecznego kanału. Dzięki temu można było przetestować niezbyt skomplikowane operacje oraz sprawdzić czy nie występują problemy podczas instalacji apletu. Nie było to najlepsze narzędzie do częstej współpracy, gdyż każde polecenie APDU musiało być 'wyklikiwane' w okienku. Generowało to duży narzut czasowy na przygotowanie przypadku testowego oraz momentami wprowadzało w irytację twórcę apletu. Application Loader został pod koniec prac zastąpiony przez GPShell. GPShell umożliwiał już tworzenie skryptów na potrzebę testowania. Serię komend (włącznie z usunięciem starego/wgraniem nowego

apletu) można było zapisać w jednym pliku testowych i przekazać jako parametr do programu. Dzięki temu faza testowania przyspieszyła i łatwiej było testować ciągi operacji.

Należy jednak zwrócić uwagę, że z racji przesunięcia funkcjonalności zgodnej ze standardami *ISO 7816-4* i *ISO 7816-4* na dalszy plan nie została ona przetestowana tak jak część dedykowana.

4.6 Sugestie dotyczące wdrożenia

Nim przystąpi się do wdrożenia apletu do środowiska SELS trzeba mieć kilka rzeczy na uwadze. Pomijając kwestie dotyczące samego procesu wgrywania apletu na kartę (będzie on realizowany najprawdopodobniej poprzez 'Terminal SELS', oprogramowanie już istniejące) należy przyjrzeć się jeszcze samemu apletowi.

Bardzo ważną rzeczą jest ustalenie konkretnej wersji klucza, która ma być wykorzystywana do ograniczenia dostępu do plików. Należy to zrobić poprzez edycję odpowiedniej stałej zdefiniowanej w pliku *jElibConstants.java*. Dla każdego pliku można ustawić inną wartość, jeśli jest taka potrzeba, ale będzie się to wiązało z koniecznością ustanawiania nowego bezpiecznego kanału przy użyciu innego klucza za każdym razem, gdy zmienimy plik, na którym operujemy.

Inną kwestią związaną z kluczami używanymi w aplecie jest ich rodzaj. Jednym rozwiazaniem jest użycie kluczy statycznych, identycznych we wszystkich kartach. Jest to rozwiązanie prostsze, jednakże obarczone sporym niebezpieczeństwem. W momencie odkrycia kluczy dla jednej z kart, jest możliwe nieuprawnione modyfikowanie i odczyt danych na wszystkich kartach korzystających z tego klucza. Drugą opcją jest dywersyfikacja klucza matki. Polega ona na odpowiedniej modyfikacji klucza matki, identycznego dla wszystkich kart, danymi identyfikującymi kartę, jak np. numer seryjny egzemplarza karty. Dzięki temu w pamięci każdej karty zapisane są różne klucze, lecz posiadając klucz matkę oraz numer seryjny karty można zweryfikować ich prawdziwość. Uzyskanie kluczy z jednej karty nie daje w tym momencie dostępu do pozostałych, gdyż klucze te są stworzone specjalnie dla tej karty. Również będąc w ich posiadaniu wraz z numerem seryjnym nie jest możliwe ich odtworzenie, gdyż są one wynikiem użycia funkcji skrótu (kryptograficznej sumy kontrolnej). Jest to rozwiązanie bardziej bezpieczne, ale bardziej skomplikowane.

Rozdział 5

Wnioski

Środowisko kart mikroprocesorowych jest bardzo ciekawe i daje szerokie pole jego wykorzystania. Od przechowywania danych po świadczenie usług uwierzytelniania i weryfikacji tożsamości. Fakt, że prawie każdy z nas posiada w portfelu jeden lub więcej takich blankietów sam za tym przemawia. Ułatwiają one życie, zastępują dokumenty papierowe, oferując jednocześnie rozgraniczenie pomiędzy informacjami, które oferują bez restrykcji jako nadruki na swojej powierzchni, a tymi poufnymi, zaszytymi w pamięci układu. Już widać próby rozwinięcia usług oferowanych przez karty SIM poprzez umieszczanie na nich oprogramowania współpracującego z różnymi instytucjami, takimi jak banki, portale internetowe, serwisy informacyjne. Zwiększa to bezpieczeństwo użytkowania tego typu usług przy równoczesnym uproszczeniu ich obsługi.

jElib jest właśnie takim oprogramowaniem, które pozwala współpracować karcie z systemem bibliotecznym. Dodatkowo odciąża on zarówno system jak i studenta, gdyż sam potrafi dostarczyć informacje, które mogą być przydatne obu stronom. Jest to pierwsza tego typu usługa w Politechnice Poznańskiej.

Zgodnie z kartą pracy dyplomowej implementacja spełnia następujące wymagania. Oferuje dostęp do danych zarówno poprzez API operujące na systemie plików, jak i API stworzone bezpośrednio pod wymagania systemu jElib. Za jej pomocą można reprezentować dowolną aplikację opartą na danych składowanych w plikach zbudowanych ze struktur *TLV*, czy posiadających prostą strukturę. Oczywistym jest jednak, że każda aplikacja stworzona na bazie jElib będzie wymagała indywidualnego podejścia, by dostosować mechanizmy zarządzania danymi w jElib do indywidualnych wymogów. Istotnym jest jednak to, że jElib bardzo ułatwia budowę logicznej reprezentacji danych takiej aplikacji.

Niestety do momentu ukończenia tej pracy nie udało się włączyć jElib do grupy oferowanych przez Politechnikę Poznańską apletów na karty inteligentne. Jest to spowodowane przez czynniki niezależne od autora oprogramowania. Aktualnie są prowadzone rozmowy dotyczące wdrożenia pomiędzy stronami związanymi z tym projektem. Pierwsze decyzje zostaną podjęte najprawdopodobniej jeszcze w lipcu 2009, jednakże wdrożenie będzie miało miejsce najprawdopodobniej dopiero w okolicach października 2009.

Możliwe drogi rozwoju

Postać apletu jElib, w jakim jest on oddawany wraz z tą pracą jest zaledwie zaczątkiem rozwiązania, które można rozwinąć i uczynić sztandarowym produktem Politechniki Poznańskiej oferowanym na karty elektroniczne. Mowa tutaj głównie o części API opracowanej na podstawie standardów. Jest to funkcjonalność niespotykana w rozwiązaniach oferowanych bezpłatnie. O żadnej takiej implementacji autorowi nie wiadomo. Część większych producentów kart oferuje swoje rozwiązania, jednakże nie zawsze są one zgodne ze standardem, a ponadto są to rozwiązania zamknięte, więc nie można pokusić się o ich ulepszenie, bądź dostosowanie do własnych potrzeb. Dzięki jElib problem tego typu został rozwiązany. API umożliwia zarządzanie plikami w sposób w 100% zgodny z wytycznymi *ISO*, co pozwala szacować, że będzie zgodny ze zdecydowaną większością oprogramowania będącego na rynku i wykorzystującego te mechanizmy.

Według autora rozwój apletu dotyczy dwóch aspektów. Pierwszym są mechanizmy bezpieczeństwa mające na celu ograniczenie dostępu do składowanych w aplecie danych oraz zarządzanie prawami do konkretnych operacji na plikach. Mowa tutaj między innymi o możliwości definiowania reguł dla operacji zapisu do/odczytu z/tworzenia/usuwania pliku dla konkretnego użytkownika, bądź grupy użytkowników. Zwiększyłoby to znacznie dokładność definiowania takich praw i umożliwiłoby bardziej efektywne zarządzanie nimi. Implementacja tej funkcjonalności musi być jednak poprzedzona głęboką analizą zagadnienia, łącznie z oszacowaniem jego fizycznego rozmiaru na karcie. Bezzasadnym byłoby oferowanie funkcjonalności, która sama w sobie ograniczałaby jej wykorzystanie. W przypadku, gdy rozmiar kodu odpowiedzialnego za te mechanizmy byłby znaczny, nie pozostałoby nic innego jak czekać na rozwój technologii, a dokładnie na upowszechnienie się kart o większym rozmiarze oferowanej pamięci *EEPROM*. Obecnie w sprzedaży jest już model firmy *Sharp* o rozmiarze pamięci 1 MB, jednak są to karty jeszcze mało popularne, a co za tym idzie, stosunkowo drogie.

Drugim tematem powiązanym z dalszym rozwojem apletu jest jego współdzielenie. Współdzielenie w sensie udostępniania funkcjonalności innym apletom. Miałoby to być zrealizowane poprzez wykorzystanie interfejsu *Shareable*. Dałoby to możliwość korzystania z systemu plików pozostałym apletom na karcie, bez konieczności implementowania go po ich stronie. Cała implementacja byłaby ukryta po stronie jElib (wtedy już może wydzielonego jako osobny aplet odpowiedzialny wyłącznie za system plików), a interfejs udostępniałby wyłącznie obiekty plików. Ujednoliciłoby to zarządzanie plikami w obrębie całej karty. Znamiennym, by było również to, że produkt ten mógłby być instalowany na 'gołych' kartach, generując tym samym oszczędności dla wystawcy karty (w tym wypadku Politechniki Poznańskiej), gdyż karty z mniejszą ilością oprogramowania są po prostu tańsze. Mógłby być to początek bazy oprogramowania, która wraz z istniejącą infrastrukturą systemową uczelni oferowałby większą interakcję i personalizację pomiędzy studentem a jego alma mater. Już w tym momencie są plany, by utworzyć tzw. kioski dla studentów,

gdzie logując się za pomocą karty mieliby dostęp do najważniejszych informacji dotyczących swojej osoby. Jest to dopiero początek tego typu rozwiązań i wiele jest jeszcze przed nami.

Literatura

[1] Dziennik ustaw z 8 grudnia 2006 nr 224. Rozporządzenie dostępne pod adresem `http://www.infor.pl/dziennik-ustaw,rok,2006,nr,224/poz,1634`.

[2] `http://java.sun.com/javacard/devkit/`. Strona projektu Java Card Development Kit.

[3] `http://pcsclite.alioth.debian.org/`. Strona projektu PC/SC Lite.

[4] `http://sourceforge.net/project/showfiles.php?group_id=143343`. Strona projektu GPShell.

[5] `http://www.cryptotech.com.pl/Pomoc_techniczna/Sterowniki,content.html#scr3310`. Sterowniki czytnika kart mikroprocesorowych CryptoTech SCR 3311 dla systemu Windows.

[6] `http://www.eclipse.org/`. Strona projektu Eclipse.

[7] `http://www.linuxnet.com/sourcedrivers.html`. Sterowinki czytników kart mikroprocesorowych dla systemu GNU Linux.

[8] *Information technology — Identification cards — Integrated circuit(s) cards with contacts. Part 3: Electronic signals and transmission protocols.* ISO/IEC, 1997.

[9] *Identification cards — Integrated circuit cards. Part 1: Physical characteristics.* ISO/IEC, 1998.

[10] *Information technology — Identification cards — Integrated circuit(s) cards with contacts. Part 2: Dimensions and location of the contacts.* ISO/IEC, 1999.

[11] *Identification cards — Integrated circuit cards. Part 9: Additional interindustry commands and security attributes.* ISO/IEC, 2000.

[12] *Identification cards — Integrated circuit cards. Part 4: Organization, security and commands for interchange.* ISO/IEC, 2005.

[13] *Inżynieria programowania kart inteligentnych.* Poltechnika Warszawska, 2005.

[14] *Development Kit User's Guide, For the Binary Release with Cryptography Extensions Java Card^{TM}Platform, Version 2.2.2.* Sun Microsystems, 2006.

[15] *Global Platform, Card Specification, Version 2.2.* Global Platform, 2006.

[16] *Runtime Environment Specification, Version 2.2.2.* Sun Microsystems, 2006.

[17] *Virtual Machine Specification, Version 2.2.2.* Sun Microsystems, 2006.

[18] *System Elektronicznej Legitymacji Studenckiej - Struktura elektroniczna ELIB.* Politechnika Poznańska, 2009.

Rozdział 6

Załącznik 1

6.1 API zgodne ze standardem ISO 7816-4

Legenda:

-- oznacza dowolną wartość, chyba że w innym miejscu określone są konkretne wartości, wtedy jest to dowolna wartość, poza określonymi,

XX oznacza konkretną wartość nieopisaną w tym dokumencie,

NN oznacza konkretną wartość uszczegółowioną w opisie.

SELECT (INS = 0xA4)

CEL

Wybranie pliku EF, bądź DF jako aktualnie używany.

PARAMETRY

Metoda obsługuje tylko wybór pierwszego/jedynego pliku (ostatnie 3 bity P2 są równe 0).

P1	P2	Opis
0x00	0x00	Wybór pliku *MF*. Pole danych może być puste lub zawierać identyfikator *0x3f00*.
0x00	--	Wybór pliku EF, DF lub MF. Pole danych zawiera identyfikator pliku.
0x01	--	Wybór pliku potomka (DF). Pole danych zawiera identyfikator pliku.
0x02	--	Wybór pliku potomka (EF). Pole danych zawiera identyfikator pliku.
0x03	--	Wybór pliku rodzica

POLE DANYCH

W większości przypadków zawiera dwubajtowy identyfikator pliku.

ZWRACANE WARTOŚCI

Instrukcja zwraca informację FCI zapisaną w strukturze *TLV* o tagu 6F.

SW1	SW2	Opis
0x69	0x85	Warunki niespełnione (plik nieodpowiedniego typu).
0x6A	0x81	Funkcja nieobsługiwana (wybór kolejnego/niejedynego pliku).
0x6A	0x82	Plik nieznaleziony.

ERASE BINARY (INS = 0x0F)

CEL

Usunięcie danych z pliku. Dane są nadpisywane wartością 'zero' zdefiniowaną w tagu 82 struktury FCI pliku.

PARAMETRY

P1	P2	Opis
0x00	0x00	Aktualnie wybrany plik.
0xXX	0xXX	Dwubajtowy identyfikator pliku (na poziome zagnieżdżenia równym poziomowi aktualnie wybranego pliku lub poziomie o jeden niższym, plik dziecka).

POLE DANYCH

Zawiera wartości przesunięcia zapisane w oddzielnych strukturach *TLV* o tagu 54. W przypadku braku takiej wartości plik kasowany jest od pierwszego bajtu. W przypadku jednej wartości, od niej do końca pliku. W przypadku dwóch wartości od bajtu wskazanego przez pierwsze przesunięcie do bajtu poprzedzającego bajt wskazany przez drugie przesunięcie, lub do końca pliku, w przypadku, gdy drugie przesunięcie jest większe niż wielkość pliku.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x81	Instrukcja niezgodna ze strukturą plików (operacja na pliku DF).
0x69	0x84	Nieprawidłowe dane (w polu danych).
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

SEARCH BINARY (INS = 0xA1)

CEL

Wyszukanie w pliku zadanego ciągu znaków.

PARAMETRY

P1	P2	Opis
0x00	0x00	Aktualnie wybrany plik.
0xXX	0xXX	Dwubajtowy identyfikator pliku (na poziome zagnieżdżenia równym poziomowi aktualnie wybranego pliku lub poziomie o jeden niższym, plik dziecka).

POLE DANYCH

Zawiera wartość przesunięcia zapisaną w strukturze *TLV* o tagu 54. W przypadku braku

takiej wartości plik przeszukiwany jest od pierwszego bajtu. W przypadku jednej wartości, od niej do końca pliku. Zawiera również szukany ciąg bajtów zapisany w strukturze *TLV* o tagu 53.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x62	0x82	Osiągnięto koniec pliku.
0x69	0x81	Instrukcja niezgodna ze strukturą plików (operacja na pliku DF).
0x69	0x84	Nieprawidłowe dane (w polu danych).
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

Instrukcja zwraca numer pierwszego bajtu pierwszego wystąpienia szukanego ciągu bajtów w pliku zapisany w strukturze *TLV* o tagu 54.

READ BINARY (INS = 0xB1)

CEL

Czytanie z pliku ciągu bajtów. Długość czytanego ciągu bajtów określa wartość L_e komendy APDU.

PARAMETRY

P1	P2	Opis
0x00	0x00	Aktualnie wybrany plik.
0xXX	0xXX	Dwubajtowy identyfikator pliku (na poziome zagnieżdżenia równym poziomowi aktualnie wybranego pliku lub poziomie o jeden niższym, plik dziecka).

POLE DANYCH

Zawiera wartość przesunięcia od którego dane z pliku mają być czytane zapisaną w strukturze *TLV* o tagu 54. W przypadku braku takiej wartości plik czytany jest od pierwszego bajtu.

ZWRACANE WARTOŚCI

Instrukcja zwraca przeczytany ciąg znaków zapisany w strukturze *TLV* o tagu 73.

SW1	SW2	Opis
0x69	0x81	Instrukcja niezgodna ze strukturą plików (operacja na pliku DF).
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

WRITE BINARY (INS = 0xD1)

CEL

Zapisanie do pliku ciągu bajtów. Operacja jest wykonywana na zasadzie operacji logicznej danych istniejących w pliku (wartości 'zero' zdefiniowanej w tagu 82 struktury FCI) oraz przesłanych do pliku. W przypadku, gdy wartością 'zero' jest bajt 0x00 to jest to suma logiczna, a gdy 0xff to iloczyn logiczny. Dla plików o strukurze *TLV* ważnym jest również miejsce, w którym dane zostaną zapisane. Możliwe jest tylko zapisywanie od bajtu

następującego po ostatnim bajcie wartości istniejącej struktury (jako nowy tag *TLV* na tym samym, bądź niższym poziomie zagnieżdżenia), oraz jako wartości innego tagu, jeśli jego wartość nie jest jeszcze zapisana (zapis od bajtu następującego po bajcie długości wartości).

PARAMETRY

P1	P2	Opis
0x00	0x00	Aktualnie wybrany plik.
0xXX	0xXX	Dwubajtowy identyfikator pliku (na poziome zagnieżdżenia równym poziomowi aktualnie wybranego pliku lub poziomie o jeden niższym, plik dziecka).

POLE DANYCH

Zawiera wartość przesunięcia od którego dane mają być zapisywane w pliku zapisaną w strukturze *TLV* o tagu 54. Wartość zapisywana przesyłana jest w strukturze *TLV* o tagu 53. Dodatkowo, jeśli mamy do czynienia z plikiem o strukturze *TlV* to w strukturze o tagu 55 podajemy jednobajtową wartość na jakiej głębokości zagnieżdżenia w strukturach *TLV* dane mają zostać zapisane w pliku.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x81	Instrukcja niezgodna ze strukturą plików (operacja na pliku DF).
0x69	0x84	Nieprawidłowe dane (w polu danych).
0x69	0x85	Warunki nie spełnione (złe przesunięcie dla plików o strukurze *TLV* lub błędna głębokość zagnieżdżenia).
0x6A	0x81	Funkcja nieobsługiwana (nieznana wartość bajtu 'zero' w tagu 82 struktury FCI).
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

UPDATE BINARY (INS = 0xD7)

CEL

Nadpisanie ciągu bajtów w pliku. Operacja jest wykonywana na zasadzie operacji logicznej danych istniejących w pliku oraz przesłanych do pliku. W przypadku, gdy wartością 'zero' jest bajt 0x00 to jest to suma logiczna, a gdy 0xff to iloczyn logiczny. W przypadku plików o strukturze *TLV* możemy uaktualnić numer tag lub całość.

PARAMETRY

P1	P2	Opis
0x00	0x00	Aktualnie wybrany plik.
0xXX	0xXX	Dwubajtowy identyfikator pliku (na poziome zagnieżdżenia równym poziomowi aktualnie wybranego pliku lub poziomie o jeden niższym, plik dziecka).

POLE DANYCH

Zawiera wartość przesunięcia od którego dane mają być zapisywane w pliku zapisaną w strukturze *TLV* o tagu 54. Wartość zapisywana przesyłana jest w strukturze *TLV* o tagu 53.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x81	Instrukcja niezgodna ze strukturą plików (operacja na pliku DF).
0x69	0x84	Nieprawidłowe dane (w polu danych).
0x69	0x85	Warunki nie spełnione (złe przesunięcie dla plików o strukurze *TLV*).
0x6A	0x80	Błędne dane (zapisywane dane nie reprezentują numeru tagu, bądź całości).
0x6A	0x81	Funkcja nieobsługiwana (nieznana wartość bajtu 'zero' w tagu 82 struktury FCI).
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

6.2 API zgodne ze standardem ISO 7816-9

CREATE FILE (INS = 0xE0)

CEL

Tworzy plik w systemie plików i alokuje dla niego miejsce.

PARAMETRY

P1	P2	Opis
0x00	0x00	Tworzy plik na podstawie informacji FCP podanej w polu danych.

POLE DANYCH

Zawiera strukturę FCP pliku w postaci strukture *TLV*. FCP musi zawierać tagi 82, 83 oraz 84 w przypadku, gdy tworzymy plik DF. Ich wartości są podane w standardzie *ISO 7816-4*. Dane w nich zawarte są wykorzystywane do określenia identyfikatora pliku (i nazwy w przypadku DF), jego rozmiaru, czy tworzymi plik EF prosty, EF struktur *TLV*, czy plik DF, oraz wartość jego bajtu wartości 'zero'.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x81	Instrukcja niezgodna ze strukturą plików.
0x69	0x84	Nieprawidłowe dane (w polu danych).
0x69	0x85	Warunki nie spełnione (wpierw trzeba utworzyć MF lub ponowne tworzenie MF).
0x6A	0x89	Plik EF o podanym identyfikatorze już istnieje.
0x6A	0x8A	Plik DF o podanej nazwie już istnieje.
0x6B	0x00	Niepoprawna wartość parametru P1 lub P2.
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

DELETE FILE (INS = 0xE4)

CEL

Usuwa plik z systemu plików i zwalnia jego miejsce w pamięci. Aktualnie wybranym plikiem staje się rodzic usuniętego pliku.

PARAMETRY

P1	P2	Opis
0x00	0x00	Usuwa aktualnie wybrany plik.

POLE DANYCH

Dane przekazywane w tym polu nie są istotne.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x85	Warunki nie spełnione (nie można usunąć MF).
0x6B	0x00	Niepoprawna wartość parametru P1 lub P2.
0xXX	0xXX	Wartości opisane w *ISO 7816-4*.

6.3 API dedykowane jElib

W związku z faktem, że aplet jest oprogramowaniem własnościowym i niezgodnym z żadnym standardem posiada własny bajt klasy *CLA* poleceń APDU oraz własny zestaw instrukcji (bajt *INS* polecenia APDU). Bajt klasy należy do przedziału zarezerwowanego na własny użytek i jest równy **0xC0**.

Tabela 6.1 przedstawia wartości P1, dzięki którym w momencie, gdy wykorzystywane są metody, które mogą operować na różnych plikach, możliwe jest wskazanie na konkretny plik apletu.

b7	b6	b5	b4	b3	b2	b1	b0	Opis
0	0	0	0	0	0	0	1	EF.CONFIG
0	0	0	0	0	0	1	0	EF.ID
0	0	0	0	0	1	0	0	EF.EVENT
0	0	0	0	1	0	0	0	EF.LOCK

Tablica 6.1: Wartości P1 dla wyboru pliku jElib.

Polecenia dotyczące wszystkich plików

Poniższe polecenia mogą być stosowane dla wszystkich plików jElib.

READ ALL (INS = 0x31)

CEL

Czyta plik.

PRZYKŁAD

Czytanie pliku EF.CONFIG od 129-tego bajtu (heksadecymalnie 0101).

```
C0 31 01 00 04 54 02 01 01 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1.

POLE DANYCH

Zawiera wartość przesunięcia, od którego dane mają być czytane, zapisaną w strukturze *TLV* o tagu 54. Jeśli brak tej wartości plik czytany jest od początku.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

INITIALIZE (INS = 0x30)

Zanim aplet będzie wykorzystywany należy zainicjalizować jego strukturę plików.

CEL

Stworzenie pustej struktury plików Elib/jElib w pamięci apletu.

PRZYKŁAD

```
C0 30 00 00 00
```

PARAMETRY

P1	P2	Opis
--	--	Wartości parametrów nieznaczące

POLE DANYCH

Puste.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

Polecenia dotyczące plików EF.CONFIG i EF.ID

Pliki EF.CONFIG i EF.ID mają bardzo podobną strukturę logiczną. Stąd też oba są traktowane jak mapa i dzielą większość metod dostępu do danych.

Wszystkie operacje zapisu do plików EF.CONFIG i EF.ID są chronione mechanizmami GlobalPlatform. By móc do nich zapisywać, aplikacja wykorzystująca aplet musi się wpierw uwieżytelnić się przy pomocy klucza o odpowiedniej wersji. Polecenia wysyłane do karty będą przesyłane poprzez bezpieczny kanał, powinny więc zawierać dodatkową wartość MAC, będącą sumą kontrolną wyliczoną na podstawie treści wiadomości.

ADD ENTRY (INS = 0x37)

CEL

Dodaje nową parę ASN.1 <klucz, wartość>. Wykorzystywane w plikach EF.CONFIG i EF.ID.

PRZYKŁAD

Dodanie pary o kluczu 0x212223 i wartości 0x010203.

```
C0 37 01 00 0A 51 03 21 22 23 53 03 01 02 03 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1 (tylko pliki EF.CONFIG i EF.ID)

POLE DANYCH

Zawiera klucz pary zapisany w strukturze *TLV* o tagu 51 oraz wartość pary zapisaną w strukturze *TLV* o tagu 53.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x84	Plik pełny
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Wybrany plik nie istnieje
0x6B	0x00	Wybrany błędny plik
0x67	0x01	Para z podanym kluczem już istnieje
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

REMOVE ENTRY (INS = 0x38)

CEL

Usuwa parę ASN.1 z pliku. Wykorzystywane w plikach EF.CONFIG i EF.ID.

PRZYKŁAD

Usuwanięcie z pliku EF.CONFIG parę o kluczu 0x212223.

```
C0 38 01 00 05 51 03 21 22 23 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1 (tylko pliki EF.CONFIG i EF.ID)

POLE DANYCH

Zawiera klucz pary zapisany w strukurze *TLV* o tagu 51.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x88	Wybrany plik nie istnieje, wskazana para nie istnieje
0x6B	0x00	Wybrany błędny plik
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

COUNT ENTRIES (INS = 0x35)

CEL

Zlicza ilość par ASN.1 w pliku. Wykorzystywane w plikach EF.CONFIG i EF.ID

PRZYKŁAD

Zliczenie ilości identyfikatorów w pliku EF.ID.

```
C0 35 02 00 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1 (tylko pliki EF.CONFIG i EF.ID)

POLE DANYCH

Puste.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x6A	0x88	Wybrany plik nie istnieje
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

GET ENTRY INDEX (INS = 0x36)

CEL

Zwraca numer indeksu pary lub klucz pary ASN.1 wewnątrz sekwencji lub zbioru struktur ASN.1 pliku, dla podanego klucza bądź indeksu. Wykorzystywane w plikach EF.CONFIG i EF.ID.

PRZYKŁAD

Pobranie numeru indeksu pary z pliku EF.CONFIG o kluczu 'TEST' (0x54455354).

```
C0 36 01 00 06 51 04 54 45 53 54 00
```

Pobranie klucza pary z pliku EF.CONFIG o indeksie 2.

```
C0 36 01 01 03 52 01 02 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1 (tylko pliki EF.CONFIG i EF.ID)
--	0x00	Pole danych zawiera klucz (tag 51)
--	0x01	Pole danych zawiera indeks pary (tag 52)

POLE DANYCH

Zawiera klucz zapisany w strukturze *TLV* o tagu 51 lub indeks w strukturze o tagu 52.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne, indeks za duży
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x86	Wartość P2 nie jest zgodna z zawartością pola danych
0x6A	0x88	Wybrany plik nie istnieje, brak pary o podanym indeksie/kluczu
0x6B	0x00	Wybrany błędny plik, błędna operacja
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

GET ENTRY VALUE (INS = 0x33)

CEL

Zwraca wartość pary ASN.1. Wykorzystywany do pobierania wartości parametru z pliku EF.CONFIG lub identyfikatora studenta w bibliotece z pliku EF.ID.

PRZYKŁAD

Pobranie wartości parametru z pliku EF.CONFIG dla klucza 'TEST' (0x54455354).

```
C0 33 01 00 06 51 04 54 45 53 54 00
```

Pobranie wartości parametru z pliku EF.CONFIG o indeksie 2.

```
C0 33 01 01 03 52 01 02 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1 (tylko pliki EF.CONFIG i EF.ID)
--	0x00	Pole danych zawiera klucz (tag 51)
--	0x01	Pole danych zawiera indeks pary (tag 52)

POLE DANYCH

Zawiera klucz zapisany w strukturze *TLV* o tagu 51 lub indeks w strukturze o tagu 52.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne, indeks za duży
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x86	Wartość P2 nie jest zgodna z zawartością pola danych
0x6A	0x88	Wybrany plik nie istnieje, brak pary o podanym indeksie/kluczu
0x6B	0x00	Wybrany błędny plik, błędna operacja
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

UPDATE ENTRY VALUE (INS = 0x34)

CEL

Uaktualnia wartość pary ASN.1. Wykorzystywane do uaktualniania wartości parametru w pliku EF.CONFIG lub identyfikatora studenta w bibliotece w pliku EF.ID.

PRZYKŁAD

Zmiana wartości parametru z pliku EF.CONFIG dla klucza 'TEST' (0x54455354) na wartość 0x1213.

```
C0 34 01 00 0A 51 08 54 45 53 54 53 02 12 13 00
```

Zmiana wartości parametru z pliku EF.CONFIG o indeksie 2 na wartość 0x1213.

```
C0 34 01 01 07 52 01 02 53 02 12 13 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Wybór pliku wg tabeli 6.1 (tylko pliki EF.CONFIG i EF.ID)
--	0x00	Pole danych zawiera klucz (tag 51)
--	0x01	Pole danych zawiera indeks pary (tag 52)

POLE DANYCH

Zawiera klucz zapisany w strukturze *TLV* o tagu 51 lub indeks w strukturze o tagu 52, oraz nową wartość w strukturze o tagu 53.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne, indeks za duży
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x86	Wartość P2 nie jest zgodna z zawartością pola danych
0x6A	0x88	Wybrany plik nie istnieje, brak pary o podanym indeksie/kluczu
0x6B	0x00	Wybrany błędny plik, błędna operacja
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

GET VERSION (INS = 0x39)

CEL

Zwraca numer wersji apletu przechowywany w pliku EF.CONFIG.

PRZYKŁAD

Pobranie numer wersji apletu.

```
C0 39 00 00 00
```

PARAMETRY

P1	P2	Opis
--	--	Wartość parametru nie jest istotna

PARAMETRY

Puste.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x6A	0x88	Plik EF.CONFIG nie istnieje, para z numerem wersji nie znaleziona
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

Polecenia dotyczące pliku EF.EVENT

Wszystkie operacje zapisu do pliku EF.EVENT są chronione mechanizmami GlobalPlatform. By móc do niego zapisywać, aplikacja wykorzystująca aplet musi się wpierw uwieżytelnić się przy pomocy klucza o odpowiedniej wersji. Polecenia wysyłane do karty będą przesyłane poprzez bezpieczny kanał, powinny więc zawierać dodatkową wartość MAC, będącą sumą kontrolną wyliczoną na podstawie treści wiadomości.

ADD EVENT (INS = 0x3B)

CEL

Dodaje nowy obiekt `EVENT` do pliku EF.EVENT.

PRZYKŁAD

Dodanie zdarzenia dla biblioteki (pole `LIBRARY ID`) 0x6162636465, data 08.05.05 12:05:01

GMT+2, książka 0x415554484F52606162636465666768, autor 'MÓJ AUTOR', tytuł 'TYTUŁ' i zdarzenie 0x01020101. Tag sekwencji 16 (hex 0x10), PrintableString 19 (hex 0x13), UTCTime 23 (hex 0x17), UTF8String 12 (hex 0x0C), Integer 2 (hex 0x02).

```
C0 3B 00 00 41 53 3F 10 3D 13 05 61 62 63 64 65 17 08 08 05 05 12 05 01 02 00
10 24 13 10 41 55 54 48 4F 52 60 61 62 63 64 65 66 67 68 0C 06 54 59 54 55 C5
81 0C 0A 4d C3 93 4a 20 41 55 54 4f 52 02 04 01 02 01 01 00
```

PARAMETRY

P1	P2	Opis
--	--	Wartość parametru nie jest istotna

POLE DANYCH

Zawiera całą strukturę *TLV* sekwencji nowego zdarzenia (`EVENT`) zapisanego w strukturze o tagu 53.

ZWRACANE WARTOŚCI

Zwraca ilość wolnych miejsc na nowe zdarzenia lub strukturę najstarszego zdarzenia zapisaną w strukturze o tagu 73 w przypadku, gdy zabrakło miejsca i najstarsze zdarzenie musiało zostać usunięte.

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

READ EVENT OF LIBRARY (INS = 0x3A)

CEL

Zwraca obiekt zdarzenia (`EVENT`) dla danej biblioteki (pole `LIBRARY`).

PRZYKŁAD

Czytanie drugiego zdarzenia dla LIBRARY ID 0x6162636465.

```
C0 3A 01 00 07 51 05 61 62 63 64 65 00
```

PARAMETRY

P1	P2	Opis
0x00	--	Zwracany jest pierwszy znaleziony obiekt dla danej biblioteki (nie ten mający miejsce najwcześniej lub najpóźniej)
0xNN	--	Zwracany jest NN-te zdarzenie dla danej biblioteki lub nic, gdy nie ma tylu zdarzeń dla danej biblioteki

POLE DANYCH

Zawiera pole `LIBRARY ID` zapisane w strukturze o tagu 51.

ZWRACANE WARTOŚCI

Zwracany obiekt zapisany jest w strukturze o tagu 73.

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje, brak zdarzenia dla danej biblioteki
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

Polecenia dotyczące pliku EF.LOCK

Wszystkie operacje odczytu i zapisu do pliku EF.LOCK są chronione mechanizmami GlobalPlatform. By móc z niego czytać lub do niego zapisywać, aplikacja wykorzystująca aplet musi się wpierw uwieżytelnić się przy pomocy klucza o odpowiedniej wersji. Polecenia wysyłane do karty będą przesyłane poprzez bezpieczny kanał, powinny więc zawierać dodatkową wartość MAC, będącą sumą kontrolną wyliczoną na podstawie treści wiadomości.

READ VECTOR (INS = 0x3C)

CEL

Czyta `lock_vector` z pliku EF.LOCK.

PRZYKŁAD

Czytanie wektora na trzeciej pozycji, typ blokady `lost`.

```
C0 3C 03 00 00
```

PARAMETRY

P1	P2	Opis
0xNN	--	Czytanie na NN-tej pozycji (od 01 do 0F).
0xFF	--	Czytanie całego wektora.

POLE DANYCH

Puste.

ZWRACANE WARTOŚCI

Zwraca wartość bajtu na podanej pozycji lub cały wektor, zależno od wartości P1.

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje, brak blokady dla danej biblioteki
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

READ ALL RECORDS (INS = 0x3E)

CEL

Czyta całą sekwencje rekordów *LOCK* wszystkich bibliotek.

PRZYKŁAD

Czytanie wszystkich rekordów *LOCK* od drugiego bajtu (przesunięcie 2-óch bajtów).

```
C0 3E 00 02 00
```

PARAMETRY

P1	P2	Opis
0xNN	0xNN	P1 reprezentuje 8 starszych bitów a P2 8 młodszych bitów wartości przesunięcia (typu *short*)

POLE DANYCH

Puste.

ZWRACANE WARTOŚCI

Zwraca całą sekwencję `LOCK` rekordów wszystkich bibliotek z pliku EF.LOCK.

SW1	SW2	Opis
0x6A	0x88	Plik nie istnieje
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

READ RECORD OF LIBRARY (INS = 0x3F)

CEL

Czyta całą sekwencję `LOCK` rekordu dla danej biblioteki (pole *LIBRARY ID*) z pliku EF.LOCK.

PRZYKŁAD

Czytanie rekordu dla LIBRARY ID 0x6162636465.

```
C0 3F 00 00 07 51 05 61 62 63 64 65 00
```

PARAMETRY

P1	P2	Opis
--	--	Wartość parametru nie jest istotna

POLE DANYCH

Zawiera identyfikator biblioteki (pole *LIBRARY ID*) zapisany w strukturze *TLV* o tagu 51.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x88	Plik nie istnieje, brak rekordu dla danej biblioteki
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

INIT LIBRARY RECORD (INS = 0x40)

CEL

Tworzy rekord dla biblioteki w pliku EF.LOCK z wypełnionymi polami `EXPIRE_DATE`, `USE_DATE` i `BOOKS_COUNT`. **Nie dodaje nowej blokady.**

PRZYKŁAD

Inicjalizacja rekordu dla `LIBRARY ID` 0x6162636465, `EXPIRE_DATE` 08.06.05 12:05:01 GMT+2, `USE_DATE` 08.05.05 12:05:01 GMT+2, and `COOKS_COUNT` 3. Tag PrintableString to 19 (hex 0x13), UTCTime 23 (hex 0x17), Integer 2 (hex 0x02).

```
C0 40 00 00 20 53 1E 13 05 61 62 63 64 65 17 08 08 06 05 12 05 01 02 00 17 08
08 05 05 12 05 01 02 00 02 01 03 00
```

PARAMETRY

P1	P2	Opis
--	--	Wartość parametru nie jest istotna

POLE DANYCH

Zawiera pola: `LIBRARY`, `EXPIRE_DATE`, `USE_DATE` i `BOOKS_COUNT` w podanej kolejności, zapisane w jednej strukturze *TLV* o tagu 53.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

ADD LOCK TO LIBRARY RECORD (INS = 0x41)

CEL

Dodaje/Uaktualnia `LOCKINFO` do/w pliku EF.LOCK. W przypadku dodawania inkrementuje również wartość licznika `lock_vector` na odpowiedniej pozycji. **To polecenie powinno zostać poprzedzone poprzedzone poleceniem** `INIT LIBRARY RECKORD`**.**

PRZYKŁAD

Dodanie dwóch blokad: 'lost' (03) oraz 'damage' (04) dla `LIBRARY ID` 0x6162636465, `LOCK_ID` 04 i 05, `LOCK_DATE` 08.05.05 12:05:01 GMT+2. Tag Sequence to 16 (hex 0x10), UTCTime 23 (hex 0x17), Integer 2 (hex 0x02).

```
C0 41 00 00 43 51 05 61 62 63 64 65 53 2A 10 13 02 01 03 02 04 00 00 00 04 17
08 08 05 05 12 05 01 02 00 10 13 02 01 04 02 04 00 00 00 05 17 08 08 05 05 12
05 01 02 00 00
```

PARAMETRY

P1	P2	Opis
--	--	Wartość parametru nie jest istotna

POLE DANYCH

Zawiera `LIBRARY ID` zapisane w strukturze *TLV* o tagu 51 i listę `LOCKINFO` (nie jako sekwencję czy zbiór ASN.1) zapisaną w strukturze o tagu 53.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje, brak rekordu dla danej biblioteki
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

DELETE LOCK FROM LIBRARY RECORD (INS = 0x43)

CEL

Usuwa `LOCKINFO` z pliku EF.LOCK dla danej biblioteki.

PRZYKŁAD

Usuwanie trzeciej i piątej blokady dla `LIBRARY ID` 0x6162636465.

```
C0 43 00 00 0B 51 05 61 62 63 64 65 53 02 03 05 00
```

PARAMETRY

P1	P2	Opis
0xFF	--	Usuwa cały rekord biblioteki EF.LOCK
--	--	Usuwa `LOCKINFO` rodzajów `LOCKTYPE` podanych w polu danych (tag 53)

POLE DANYCH

Zawiera `LIBRARY ID` zapisane w strukturze *TLV* o tagu 51 i strukturę o tagu 53 zawierającą te wartości `LOCKTYPE`, które chcemy usunąć dla danej biblioteki, lub tylko `LIBRARY ID` w strukturze *TLV* o tagu 51 jeśli z pliku EF.LOCK usunięty ma zostać cały rekord biblioteki. W przypadku, gdy po usunięciu blokad nie ma żadnej blokady dla biblioteki, rekord biblioteki również zostaje usunięty.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku (LOCKTYPE invalid)
0x6A	0x88	Plik nie istnieje, brak rekordu dla danej biblioteki lub `LOCKTYPE`
0xXX	0x0xXX	Zgodne z ISO 7816-4 2.2.4

READ LOCK FIELD (INS = 0x44)

CEL

Czyta pole z sekwencji `LOCK` pliku EF.LOCK.

PRZYKŁAD

Czytanie pola `LOCK_ID` blokady 'damage' (04) dla LIBRARY ID 0x6162636465.

```
C0 44 06 04 07 51 05 61 62 63 64 65 00
```

PARAMETRY

P1	P2	Opis
0x01	--	Zwraca pole `LIBRARY`
0x02	--	Zwraca pole `EXPIRE_DATE`
0x03	--	Zwraca pole `USE_DATE`
0x04	--	Zwraca pole `BOOKS_COUNT`
0x05	0xNN	Zwraca pole `LOCK_TYPE` z `LOCKINFO` na NN-tej pozycji
0x06	0xNN	Zwraca pole `LOCK_ID` z `LOCKINFO` na NN-tej pozycji
0x07	0xNN	Zwraca pole `LOCK_DATE` z `LOCKINFO` na NN-tej pozycji

POLE DANYCH

Zawiera `LIBRARY ID` zapisane w strukturze *TLV* o tagu 51. W przypadku pól `LOCK_TYPE`, `LOCK_ID` i `LOCK_DATE` P2 wskazuje konkretny numer `LOCKTYPE`.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje, brak rekordu dla danej biblioteki lub `LOCKTYPE` (P2)
0x6B	0x00	Niepoprawna wartość P1/P2
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

UPDATE LOCK FIELD (INS = 0x45)

CEL

Uaktualnia pola sekwencji `LOCK` w pliku EF.LOCK.

PRZYKŁAD

Uaktualnienie pola `LOCK_ID` blokady 'damage' (04) dla `LIBRARY ID` 0x6162636465 wartością 0x1213.

```
C0 45 06 04 0B 51 05 61 62 63 64 65 53 02 12 13 00
```

PARAMETRY

P1	P2	Opis
0x01	--	Uaktualnia pole `LIBRARY`
0x02	--	Uaktualnia pole `EXPIRE_DATE`
0x03	--	Uaktualnia pole `USE_DATE`
0x04	--	Uaktualnia pole `BOOKS_COUNT`
0x05	0xNN	Uaktualnia pole `LOCK_TYPE` z `LOCKINFO` na NN-tej pozycji
0x06	0xNN	Uaktualnia pole `LOCK_ID` z `LOCKINFO` na NN-tej pozycji
0x07	0xNN	Uaktualnia pole `LOCK_DATE` z `LOCKINFO` na NN-tej pozycji

POLE DANYCH

Zawiera `LIBRARY ID` zapisane w strukturze *TLV* o tagu 51 oraz nową wartość w strukturze o tagu 53. W przypadku pól `LOCK_TYPE`, `LOCK_ID` i `LOCK_DATE` P2 wskazuje konkretny numer `LOCKTYPE`.

ZWRACANE WARTOŚCI

SW1	SW2	Opis
0x69	0x84	Dane w polu danych są niepoprawne
0x6A	0x85	Dane w polu danych są niezgodne ze strukturą *TLV* pliku
0x6A	0x88	Plik nie istnieje, brak rekordu dla danej biblioteki lub `LOCKTYPE` (P2)
0x6B	0x00	Niepoprawna wartość P1/P2
0xXX	0xXX	Zgodne z ISO 7816-4 2.2.4

Rozdział 7

Załącznik 2

Na dołączonym nośniku CD znajduje się cała struktura projektu, czyli kod źródłowy projektu oraz narzędzia niezbędne do jego kompilacji.