Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa inżynierska

APLET PKI NA ELEKTRONICZNEJ LEGITYMACJI STUDENCKIEJ

Karol Michałowski, 127217
Paweł Ruminkiewicz, 127212
Michał Zalewski, 122569

Promotor
dr hab. inż. Marek Mika

Opiekun
mgr inż. Marek Gosławski

Poznań, 2019 r.

Miejsce na kartę pracy dyplomowej.

Spis treści

Rozdział 1

Wstęp

Jakiś czas temu legitymacja studencka występowała jedynie w formie papierowej. Aktualnie przepisy wymuszają od uczelni stosowanie jedynie legitymacji w formie kart elektronicznych, dzięki czemu zaczyna ona spełniać coraz więcej nowych funkcji. Niestety implikuje to potrzebę stosowania odpowiednich zabezpieczeń cyfrowych. Połączenie tych dwóch światów jest elementem, którym warto się zająć i tym właśnie zajmuje się poniższa praca.

1.1 Cel i zakres pracy

Celem pracy był przegląd istniejących rozwiązań i wybór apletu PKI *(ang. Public Key Infrastructure)* z otwartym kodem źródłowym dla kart elektronicznych, który spełnia wymagania Elektronicznej Legitymacji Studenckiej (ELS) i Elektronicznej Legitymacji Doktoranckiej (ELD). Projekt polegał na przygotowaniu środowiska wraz z adaptacją apletu dla kart elektronicznych używanych jako ELS/ELD na uczelni oraz kompilacją tego apletu. Jako przedmiot pracy zostały wykorzystane karty producentów Oberthur, JCOP oraz Giesecke & Devrient. Wobec apletu postawiono następujące wymagania: miał on być dostosowany do wymagań uczelni oraz miał pozwalać na ochronę zawartości karty numerami PIN *(ang. Personal Identification Number)* oraz PUK *(ang. Personal Unblocking Key)*, zapewniać składowanie certyfikatów, kluczy prywatnych oraz publicznych. Funkcjonalność musiała również umożliwiać generowanie kluczy bezpośrednio na karcie aby zagwarantować, że nie zostaną one skompromitowane przed pierwszym użyciem. Koniecznością dla apletu było także spełnianie normy ISO 7816. Kolejnym zadaniem w pracy było przygotowanie oprogramowania pośredniczącego między systemem operacyjnym użytkownika a kartą w oparciu o bibliotekę z otwartym kodem źródłowym. Ostatnim zadaniem pracy było dostosowanie aplikacji SmartCard Suite do korzystania z apletu.

1.2 Motywacja

Motywacją do podjęcia tematu pracy była chęć stworzenia rozwiązania, które pozwalałoby na korzystanie z Infrastruktury Klucza Publicznego na wszystkich obecnie stosowanych w uczelni ELS/ELD. Dotychczas było to możliwe jedynie na kartach posiadających fabryczne aplety z tą funkcjonalnością. Tym samym zmniejsza to koszty wynikające z konieczności zakupu kart posiadających preinstalowany aplet PKI producenta. Wybrane rozwiązanie udostępnia również otwarte oprogramowanie pośredniczące (middleware), co niweluje koszty związane z zakupem tego rodzaju oprogramowania od producenta kart.

Dodatkową motywację stanowi chęć popularyzacji rozwiązań infrastruktury PKI wśród studentów, propagując wykorzystywanie ELS jako bezpiecznego nośnika certyfikatów. Takie działanie poszerza grupę osób stosujących bezpieczne praktyki w cyfrowym świecie. Dodatkowe funkcjonalności legitymacji zapewniają możliwość korzystania z karty przy szyfrowaniu dokumentów oraz wiadomości email, a także potwierdzania ich autentyczności poprzez podpis cyfrowy.

1.3 Podział pracy

Zakres pracy oraz jej złożoność pozwoliła na podzielenie pracy na w miarę niezależne zadania i tym samym jej zrównoleglenie. Podział przedstawia się następująco:

- dostosowanie apletu i obsługa oprogramowania pośredniczącego w zakresie obsługi przez kartę produkcji Oberthur - Karol Michałowski
- stworzenie skryptów dla oprogramowania pośredniczącego w ramach generowania i usuwania kluczy kryptograficznych oraz zmiany kodu PIN - Karol Michałowski
- testowanie działania apletu w ramach aplikacji klienckiej Thunderbird, Microsoft Word i Adobe Reader - Karol Michałowski
- dostosowanie apletu i obsługa oprogramowania pośredniczącego w zakresie obsługi przez kartę produkcji JCOP - Paweł Ruminkiewicz
- stworzenie skryptów dla oprogramowania pośredniczącego w ramach ładowania certyfikatów i weryfikacji działania apletu - Paweł Ruminkiewicz
- testowanie działania apletu w ramach aplikacji klienckiej VeraCrypt i dostosowanie biblioteki OpenSC do obsługi użytkowanych kart z zasobnikiem systemu Windows - Paweł Ruminkiewicz
- dostostosowanie aplikacji SmartCard Suite do korzystania z apletu w ramach zmiany kodu PIN, wprowadzenia polityki bezpieczeństwa dla kodu PIN, odświeżania certyfikatów oraz generowania kluczy kryptograficznych - Michał Zalewski

Rozdział 2

Podstawy teoretyczne

W niniejszym rozdziale przedstawiono teoretyczne wiadomości wprowadzające w temat niniejszej pracy. Poszczególne podrozdziały dotyczą historii i budowy kart elektronicznych, kryptografii, certyfikatów, Infrastruktury Klucza Publicznego, standardów kryptografii PKCS oraz struktury komunikacji z kartą.

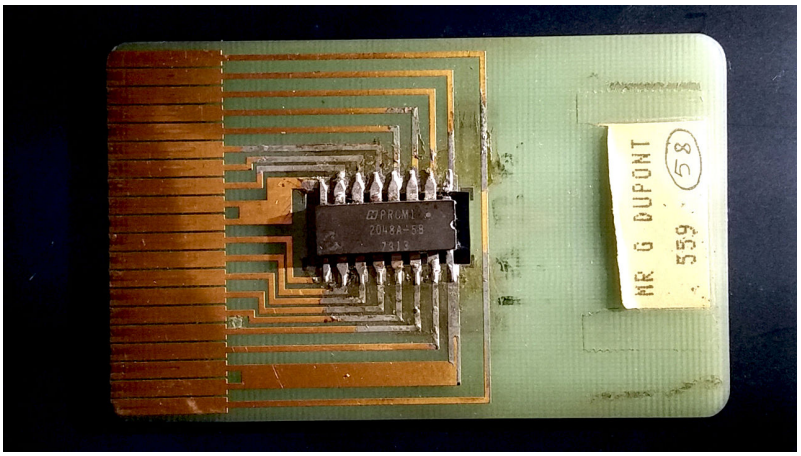
2.1 Karty elektroniczne

Podrozdział karty elektroniczne przedstawia informacje teoretyczne na temat budowy kart, struktury danych znajdujących się na nich oraz krótki rys historyczny.

2.1.1 Historia kart elektronicznych

Historia kart elektronicznych rozpoczyna się wraz z historią kart płatniczych. Elektroniczne karty płatnicze powstały jako ewolucja kart z wbudowanym paskiem. Głównym powodem ich powstania była konieczność przeciwdziałania kopiowaniu i późniejszemu nieautoryzowanemu użyciu [10].

Prace nad modelem karty elektronicznej zostały zapoczątkowane przez niemieckich wynalazców Jurgena Dethloffa i Helmuta Grotruppa [11], którzy zgłosili patent automatycznej karty chipowej w 1968 roku. Pierwszą osobą, która przedstawiła realny projekt karty elektronicznej był Francuz Roland Moreno. W 1974 roku zarejestrował patent karty, a w 1975 roku przedstawił prototyp oraz dokonał pierwszej transakcji płatniczej za jej pomocą. Jego prace były możliwe dzięki stworzeniu przez firmę Intel jednoukładowego procesora komputerowego Intel 4004 w 1971 roku [53]. Podobne prace prowadził również, niezależnie, Japończyk Kunitaka Arimura, który zgłosił patent w 1970 roku.



Rysunek 2.1: Prototyp kart elektronicznych stworzony przez Rolanda Moreno około 1975 roku, wersja z jeszcze niezminiaturyzowanym układem. [32]

W 1984 w Niemczech oraz Francji wydano pierwsze karty telefoniczne oparte na kartach elektronicznych(karty pamięciowe) oraz rozpoczęto ich wdrażanie w sektorze bankowym(karty proce-

sorowe) [11]. Ważnym krokiem w rozwoju kart była pierwsza specyfikacja elektronicznych kart płatniczych, opublikowana w 1994 roku jako wspólny produkt firm EuroPay, MasterCard i Visa [47]. Obecnie karty mogą pełnić więcej niż jedną funkcję. Przykładem może być Elektroniczna Legitymacja Studencka (ELS), która może spełniać funkcję legitymacji, karty bibliotecznej, nośnika biletu komunikacji miejskiej, nośnika certyfikatu właściciela karty czy środka płatniczego [55].

Karty elektroniczne są również szerzej wykorzystywane w kontekście identyfikacji. 3 kwietnia 2014 roku Parlament Europejski przyjął projekt rozporządzenia w sprawie identyfikacji elektronicznej i usług zaufania, znanego pod skróconą nazwą jako „Rozporządzenie eIDAS" [56], które ma na celu ujednolicenie systemów identyfikacji obywateli państw członkowskich Unii Europejskiej. Karty elektroniczne stanowią nośnik dla kwalifikowanego podpisu elektronicznego, który w myśl ustawy i rozporządzenia eIDAS jest równoważny pod względem skutków prawnych podpisowi własnoręcznemu.

Kolejnym etapem w historii kart są elektroniczne dokumenty, które w Polsce są wdrażane w postaci e-Dowodu [52], będącego cyfrowym nośnikiem danych osobowych obywateli Rzeczypospolitej Polskiej. W przeciwieństwie do tradycyjnego dowodu pozwala na przechowywanie większej ilości danych i skrócenie czasu obsługi obywatela przez administrację publiczną, dzięki zastosowaniu czytników, gdyż dane zostaną zaimportowane do systemu z karty, bez konieczności ich ręcznego wprowadzania.

2.1.2 Techniczne aspekty kart elektronicznych

Karty elektroniczne możemy podzielić m.in. ze względu na [7]:

- rodzaj zastosowanego układu elektronicznego, gdzie wyróżniamy:
  - karty procesorowe - karty posiadające wbudowany mikroprocesor, pozwalający na realizację szerszej logiki i umożliwiający pracę wielu aplikacji na jednej karcie. Karty tego typu znajdują zastosowanie między innymi jako Elektroniczne Legitymacje Studenckie bądź karty debetowe,
  - karty pamięciowe - karty pozwalające na bezpośredni dostęp do układu pamięci, cechują się one niskim kosztem produkcji i znajdują zastosowanie głównie jako karty telefoniczne,
- rodzaj interfejsu, gdzie wyróżniamy:
  - karty z interfejsem stykowym - karta do nawiązania komunikacji używa wbudowanych styków, które wymagają bezpośredniego kontaktu ze stykami terminala,
  - karty z interfejsem bezstykowym - karta nie wymaga bezpośredniego kontaktu z terminalem. Zasilanie karty i realizacja połączenia odbywa się na zasadzie działania sprzężenia indukcyjnego bądź pojemnościowego obwodu anteny karty i anteny terminala.

Karty elektroniczne doczekały się standaryzacji. Dla kart elektronicznych z interfejsem stykowym opracowano normę ISO/IEC 7816 [14]. Opisane są w niej fizyczne parametry karty takie jak odporność na czynniki zewnętrzne, wymiary i położenie styków (część 1 i 2 normy), dwa najczęściej stosowane protokoły komunikacji (T=0 i T=1) (część 3 normy). Czwarta część normy specyfikuje system plików w kartach elektronicznych i komendy APDU pozwalające na komunikację, piąta część definiuje rejestr dostawców aplikacji (jak korzystać z identyfikatora aplikacji, jak pobierać dane z aplikacji na karcie), szósta opisuje elementy danych, które mogą być stosowane w wymianie międzybranżowej (określa cechy identyfikatora, nazwy, opisu i referencji oraz format i kodowanie). Siódma część przedstawia międzybranżowe komendy dla języka SCQL *(ang. Structured Card Query Language)*, ósma referuje komendy i mechanizmy dla operacji bezpieczeństwa (jak skonstruować mechanizmy bezpieczeństwa, wykorzystując komendy z czwartej części normy). Dziewiąta część normy charakteryzuje komendy do zarządzania kartą (pozwalające zarządzać cyklem życia karty). Dziesiąta część normy określa sygnały elektryczne i odpowiedzi na sygnał reset dla kart synchronicznych. Jedenasta część normy objaśnia sposoby weryfikacji osobistej poprzez zastosowanie biometrii. Dwunasta część specyfikuje warunki działania karty z układem scalonym interfejsu USB (pojawiające się błędy, standardowe deskryptory USB, używanie transferów zbiorczych oraz kontrolnych). Trzynasta część normy listuje dostępne komendy do zarządzania aplikacjami w środowisku multiaplikacyjnym (polecenia obejmują cały cykl życia karty i mogą być używane przed i po wydaniu karty użytkownikowi końcowemu). Ostatnia, piętnasta część normy, opisuje aplikację na karcie, która zawiera informacje o funkcjach kryptograficznych karty.

Odpowiednikiem dla kart z interfejsem bezstykowym jest norma ISO/IEC 14443 [15], składająca się z 4 rozdziałów, opisujących kolejno dane techniczne, parametry łącza radiowego, wywołanie protokołu i zabezpieczenia oraz protokół transmisji danych.
Zaawansowane, wieloaplikacyjne karty procesorowe zostały opisane w specyfikacji GlobalPtaform [39], definiującej specyfikacje dotyczące bezpieczeństwa kart chip-owych, które pozwalają na bezpieczne zarządzanie w całym cyklu życia urządzeń. Dla kart procesorowych stworzone zostały dedykowane systemy operacyjne, będące specjalnymi apletami, pozwalającymi na uruchomienie aplikacji napisanych w dedykowanych dla nich językach. Spośród takich systemów możemy wyróżnić [4]: JavaCard (będący również nazwą odmiany języka Java, służącym do tworzenia apletów), Small-OS, SOSSE, MULTOS, SetCOS, Cryptoflex, MPCOS, GPK oraz GemClub-Micro.

2.1.3 Elektroniczna Legitymacja Studencka

Elektroniczna Legitymacja Studencka jako dokument wydawany przez polskie uczelnie podlega regulacjom prawnym w postaci Rozporządzenia Ministra Nauki i Szkolnictwa Wyższego z dnia 27 września 2018 w sprawie studiów [13]. Dotychczas prawo pozwalało na wydawanie legitymacji w formie papierowej, jednakże wyżej wymienione rozporządzenie definiuje ELS jedynie jako elektroniczną kartę procesorową z interfejsem stykowym określonym w normach ISO/IEC 7816-2 i ISO/IEC 7816-3.

Rysunek 2.2: Wzór legitymacji studenckiej z dn. 28.09.2018 [13]

Rozporządzenie określa format plików dla legitymacji studenckiej, definiując konieczność lokalizacji w pamięci karty pliku DF.SELS oraz dwóch plików potomnych: EF.CERT i EF.ELS. Plik DF.SELS powinien być dostępny za pomocą polecenia SELECT FILE bezpośrednio po resecie karty. Szczegółowe informacje na temat budowy tych plików oraz wzoru legitymacji możliwe są do odnalezienia w załączniku do rozporządzenia.

2.2 Kryptografia

Kryptografia to gałąź kryptologii obejmująca zagadnienia związane z utajnianiem danych poprzez operacje szyfrowania i deszyfrowania. Do ich wykonania potrzebny jest specjalny klucz. Metody kryptograficzne, ze względu na sposób konstruowania klucza, dzieli się na metody symetryczne i asymetryczne. Obecnie kryptografia obejmuje szerszy zakres, m.in. identyfikację użytkowników i podpisy elektroniczne.

Symetryczne metody kryptograficzne [9] opierają się na istnieniu jednego klucza szyfrującego. Służy on do szyfrowania i deszyfrowania wiadomości. Bezpieczeństwo systemu jest zależne od tajności klucza. Przez występowanie tylko jednego klucza, kryptografia symetryczna sprawia pewne niedogodności. Jako przykład można podać brak sposobu na udowodnienie, że dana wiadomość została wysłana przez danego nadawcę (ograniczona autentyczność). Również kanały przez, które przesyłana jest wiadomość, muszą być dużo bezpieczniejsze. Przykładami wykorzystania tego typu kryptografii są między innymi algorytmy DES oraz AES.

Asymetryczne metody kryptograficzne [6] zostały zapoczątkowane Whita Diffiego i Martina Hellmana w 1976 roku. Opierają się one na koncepcji istnienia dwóch kluczy (publicznego i prywatnego), służących do szyfrowania i rozszyfrowywania wiadomości. Wiadomość zaszyfrowaną z użyciem klucza publicznego można odszyfrować wyłącznie przy użyciu odpowiedniego klucza prywatnego. System ten zażegnał problem z przesyłaniem klucza szyfrującego pomiędzy nadawcą a odbiorcą. Koncepcję kryptografii asymetrycznej wykorzystują między innymi algorytmy: RSA i DSA [6].

2.2.1 Algorytmy kryptograficzne

W poniższym podrozdziale przedstawione zostały najpopularniejsze algorytmy kryptografii symetrycznej (DES, AES) oraz kryptografii asymetrycznej (RSA, DSA, ECC).

Algorytm **Data Encryption Standard** [45] to algorytm opracowany przez firmę IBM w połowie lat 70. Algorytm został stworzony na zlecenie *National Seciurity Agency*. Proces szyfrowania danych jest oparty o 64 bitowy klucz symetryczny oraz sieć Feistela. Obecnie algorytm DES nie stanowi odpowiedniego zabezpieczenia danych. Kryptoanaliza metodą przeszukiwania wyczerpującego pozwala na złamanie algorytmu DES w kilka minut. Przykładowe modyfikacje algorytmu DES to: 3DES, DESX, DES-768.

Advanced Encryption Standard [42] jest blokowym algorytmem symetrycznym wspierającym klucze 128, 192 oraz 256 bitowe. Dane natomiast są przetrzymywane w 128 bitowych blokach. Twórcami algorytmu są belgijscy kryptolodzy Vincent Rijmen oraz Joan Daemen, a oryginalna nazwa algorytmu to *Rijndael*. Algorytm AES jest oparty o sieć substytucji-permutacji. Algorytm AES jest następcą wcześniej opisanego algorytmu DES.

Z kolei **RSA** [45] to asymetryczny algorytm kryptograficzny, a jego nazwa pochodzi od pierwszych liter nazwisk twórców algorytmu: Ronalda Rivesta, Adi Shamira oraz Leonarda Adlema. Algorytm RSA został opublikowany w 1978 roku. Generowanie pary kluczy jest oparte o wartość funkcji Eulera oraz liczby względnie pierwsze [40]. Algorytm RSA jest obecnie jednym z najczęściej wykorzystywanych algorytmów kryptografii asymetrycznej. Rozwinięciem algorytmu RSA jest algorytm RSA_CRT wykorzystujący chińskie twierdzenie o resztach. Wykorzystanie tego twierdzenia skróciło czas operacji generowania pary kluczy [46].

Następnym z algorytmów asymetrycznych jest **Digital Signature Algorithm** [30] został został stworzony przez Narodowy Instytut Standaryzacji i Technologii Stanów Zjednoczonych *NIST*. Działanie algorytmu jest oparte o algebraiczne właściwości potęgowania modularnego wraz z problemem logarytmów dyskretnych. Opisywany algorytm jest rozwinięciem algorytmu *ElGamal* [54]. W przeciwieństwie do algorytmu *ElGamal*, podpis elektroniczny wygenerowany algorytmem DSA składa się z mniejszej liczby bitów, a jego weryfikacja wykorzystuje dwie (zamiast trzech) operacje potęgowania modularnego.

Kryptografia krzywych eliptycznych *(ang. Elliptic-curve cryptography (ECC))* [3] jest kryptosystemem opracowany niezależnie przez matematyków: Victora S. Millera oraz Neala Koblitza w latach 80-tych XX wieku. Podstawowym założeniem omawianej kryptografii jest wykorzystanie operacji opartych o krzywe eliptyczne oraz ciała skończone [3]. Istnieją dwa podstawowe rodzaje ciał skończonych: ciała skończone jako rozszerzenia pojedyncze *ang. Prime Fields (EC_FP)* oraz ciała skończone implementowane przy pomocy wielomianów *ang. Non-Prime Fields* [1].
Wykorzystanie kryptografii krzywych eliptycznych pozwala generować klucze o krótszej długości bitowej niż algorytm RSA, zapewniając taką samą moc kryptograficzną [3]. Przykładowymi algorytmami, wykorzystującymi krzywe eliptyczne, są: ECDSA oraz ECDH [16].

2.2.2 Podpis cyfrowy

Podpis cyfrowy umożliwia weryfikację autentyczności i integralności otrzymanej wiadomości elektronicznej. W celu utworzenia podpisu cyfrowego wykorzystujemy kryptografię asymetryczną. Proces generacji podpisu oraz jego weryfikacji jest następujący [2]:

1. Nadawca po utworzeniu wiadomości generuje skrót wiadomości za pomocą funkcji skrótu
2. Nadawca szyfruje skrót swoim kluczem prywatnym, tworząc **podpis cyfrowy**
3. Podpis cyfrowy oraz wiadomość zostają połączone i przekazane do adresata kanałem komunikacyjnym
4. Adresat, wykorzystując klucz publiczny nadawcy, deszyfruje otrzymany wraz z wiadomością podpis elektroniczny
5. Adresat, wykorzystując tę samą funkcję skrótu co nadawca, generuje własny, niezależny skrót otrzymanej wiadomości
6. Skróty - obliczone przez adresata i otrzymany od nadawcy są porównywane:
   a) jeśli, wartości są takie same - podpis potwierdza autentyczność nadawcy oraz integralność wiadomości

b) jeśli, wartości różnią się - podpis nie potwierdza autentyczności nadawcy oraz integralności wiadomości, ponieważ:
   - albo nastąpiła modyfikacja wiadomości w kanale - stworzenie dwóch identycznych skrótów za pomocą dwóch różnych wiadomości jest mało prawdopodobne
   - nastąpiła próba podszycia się pod nadawcę - skrót został zaszyfrowany za pomocą innego klucza niż klucz prywatny nadawcy

2.3 Certyfikat

Koncepcja certyfikatu po raz pierwszy została przedstawiona w pracy inżynierskiej Lorena Kohnfeldera [12]. Certyfikat został przedstawiony jako podpisana struktura danych połączona bezpośrednio z kluczem publicznym podmiotu. Certyfikat cyfrowy jest dokumentem elektronicznym, zapewniającym integralność klucza publicznego i pozwalającym na weryfikację podmiotu oraz klucza publicznego, który został przypisany temu podmiotowi. Koncepcja certyfikatu oraz jego formaty zostały szerzej omówione w [5].

2.3.1 Formaty certyfikatów cyfrowych

Certyfikat klucza publicznego X.509 [25] został wdrożony przez grupę roboczą IETF PKIX, a jego pierwsza wersja ukazała się w 1988 roku. Obecnie format X.509 w wersji trzeciej jest najpopularniejszym formatem certyfikatu klucza publicznego. Zawdzięcza to mechanizmowi rozszerzania, dzięki któremu może być szeroko wykorzystywany. Struktura certyfikatu jest następująca:

- *Wersja* - określa wersję certyfikatu,
- *Numer seryjny* - unikatowy identyfikator certyfikatu,
- *Podpis elektroniczny właściciela* - podpis oraz informacje dotyczące wykorzystywanej funkcji skrótu,
- *Wydawca* - nazwa organizacji wydającej certyfikat,
- *Termin ważności* - oznacza okres ważności certyfikatu,
- *Właściciel* - nazwa właściciela certyfikatu,
- *Informacja o kluczu publicznym właściciela* - klucz publiczny oraz związane z nim informacje,
- *Unikatowy identyfikator wydawcy* - jest to identyfikator urzędu wydającego certyfikat,
- *Rozszerzenia* - występują rozszerzenia: krytyczne (muszą zostać przetworzone i zaakceptowane), niekrytyczne (mogą zostać pominięte), prywatne (wykorzystywane w domenach),
- *Podpis elektroniczny wydawcy* - podpis oraz informacje dotyczące wykorzystywanej funkcji skrótu.

Prosta infrastruktura klucza publicznego *(ang. Simple Public Key Infrastructure - SPKI)* [23] jest alternatywnym typem certyfikatu klucza publicznego, stworzonym przez grupę IETF SPKI. Podstawowym założeniem grupy było uproszczenie istniejącego już systemu X.509. Certyfikaty SPKI są określane jako *certyfikaty autoryzacyjne*, ponieważ ich podstawowym zadaniem jest potwierdzenie uprawnień podmiotu.

Metoda **Pretty Good Privacy** [19] została opracowana przez Phila Zimmermana w 1991 roku, a jej głównym zadaniem jest szyfrowanie komunikacji internetowej. Certyfikaty PGP oraz certyfikaty X.509 znacząco różnią się od siebie, co uniemożliwia łatwą współpracę użytkowników. Obecnie format certyfikatu PGP jest określany przez grupę roboczą OpenPGP. Certyfikat PGP podlega weryfikacji opartej na *sieci zaufania*, przez co nie jest szeroko wykorzystywany przez przedsiębiorstwa. Niezależną implementacją PGP jest *GNU Privacy Guard (GPG)* [51].

2.4 Infrastruktura Klucza Publicznego (PKI)

Infrastruktura Klucza Publicznego *(ang. Public Key Infrastructure)* [2] jest zbiorem polityk, narzędzi i instytucji niezbędnych do tworzenia, zarządzania, korzystania, magazynowania oraz rozpowszechniania certyfikatów klucza publicznego. Koncepcja Infrastruktury Klucza Publicznego, w ramach jej struktury, zadań, protokołów bezpiecznej komunikacji oraz modeli zaufania, została szerzej przedstawiona w [5].

2.4.1 Podstawowe elementy struktury PKI

W rozdziale zostaną przedstawione podstawowe elementy struktury Infrastruktury Klucza Publicznego.

Podstawowym elementem Infrastruktury Klucza Publicznego **Urząd Certyfikacji (CA)** *(ang. Certificate authority)*. Jego podstawowym zadaniem jest wydawanie certyfikatów oraz znakowanie ich swoim podpisem cyfrowym. Taki podpis zapewnia użytkownika, że właściciel certyfikatu oraz dane w nim zawarte są poprawne.

Urząd Rejestracji (RA) *(ang. Registration authority)* jest odpowiedzialny za weryfikację danych użytkownika i jego późniejszą rejestrację. Wdrożenie RA pozwala na obniżenie kosztów operacyjnych, jednak nie jest on niezbędnym elementem PKI, ponieważ jego zadania mogą być realizowane przez CA.

Repozytorium Certyfikatów jest ogólnodostępną bazą certyfikatów, które są certyfikowane przez CA. Istnienie repozytorium certyfikatów ułatwia znalezienie klucza publicznego podmiotu, któremu chcemy przesłać zaszyfrowaną wiadomość.

Lista Unieważnionych Certyfikatów (CRL) *(ang. Certificate Revocation List)* jest to podpisana struktura danych zawierająca certyfikaty, które utraciły swoją ważność. Listy Unieważnionych Certyfikatów są często preinstalowane w aplikacjach w celu optymalizacji. Obecnie CRL jest podzielone na :

- **EPRL *(ang. End-entity Public Key Revocation List)*** - listy zawierające unieważnione certyfikaty użytkowników końcowych[17],
- **CARL *(ang. Certification Authority Revocation List)*** - listy zawierające unieważnione certyfikaty Urzędów Certyfikacji [17].

2.4.2 Zadania PKI

Do zadań PKI zaliczamy [5]: rejestrację, certyfikację, generację kluczy, aktualizację kluczy, odnawianie certyfikatu, certyfikację wzajemną oraz unieważnianie certyfikatów.

Rejestracją nazywamy proces uzyskiwania certyfikatu przez petenta, w tym celu składa on wniosek do Urzędu Rejestracji, a także dostarcza niezbędne dane określone przez Kodeks Postępowania Certyfikacyjnego *(ang. Certification Practices Statement)*. Zwykle proces rejestracji użytkownika końcowego nie jest rygorystyczny, jednak w niektórych przypadkach, podczas rejestracji niezbędna jest fizyczna obecność petenta w RA lub wieloetapowa identyfikacja (ukazanie dowodu osobistego, legitymacji pracowniczej) na przykład, gdy podmiot będzie autoryzował przelewy bankowe.

Certyfikacja to tworzenie, wystawienie oraz podpisywanie certyfikatu przez jednostkę certyfikującą i późniejsze udostępnienie go użytkownikowi. Aby certyfikacja mogła przebiec pomyślnie, niezbędna jest walidacja:

- podpisu elektronicznego, poprzez obliczenie jego wartości przy użyciu klucza publicznego,
- powiązania klucza publicznego oraz podmiotu,
- innych wymaganych danych.

Generowanie kluczy to proces tworzenia klucza prywatnego oraz klucza publicznego. Klucze mogą zostać wygenerowane przez: klienta końcowego, CA lub RA. Jeśli klucz został wygenerowany przez zaufaną trzecią stronę to w celu zapewnienia poufności, klucze dostarczane są do użytkownika w postaci mikroprocesorowych kart elektronicznych (SmartCard) lub tokenów USB. Miejsce generowania kluczy zależy od ich przeznaczenia i odbywa się w opisany poniżej sposób w zależności od następujących właściwości:

- **Niezaprzeczalność** - w celu zapewnienia niezaprzeczalności zalecanym jest, by generowanie klucza odbywała się po stronie użytkownika. W ten sposób możemy zapewnić, że klucz prywatny wykorzystywany do podpisu znajduje się tylko i wyłączenie po stronie użytkownika
- **Wydajność** - w przypadku, gdy mamy do czynienia ze sprzętem o niskiej wydajności lub mającym ograniczone możliwości, możemy zlecić wygenerowanie klucza Urzędowi Certyfikacji lub zaufanej trzeciej stronie
- **Gwarancja poprawności** - by zapewnić, że para kluczy została wykonana zgodnie z wymaganiami, należy zlecić proces tworzenia kluczy Urzędowi Certyfikującemu lub zaufanej trzeciej stronie.
- **Następstwa prawne** - użytkownik końcowy może nie mieć możliwości zapewnienia wymaganego stopnia niezawodności, w tym przypadku proces generowania pary kluczy należy zlecić Urzędowi Certyfikującemu lub zaufanej trzeciej stronie

Aktualizacja kluczy to proces generowania nowego klucza prywatnego oraz publicznego, a także związanego z nimi certyfikatu w momencie upływu terminu jego ważności, który zależy od reguł certyfikatów oraz Kodeksu Postępowania Certyfikacyjnego.

W przypadku, gdy termin ważności certyfikatu zbliża się ku końcowi, może dojść do **odnowienia certyfikatu**, czyli ponownego użycia pary kluczy w nowym certyfikacie. Klucze mogą zostać ponownie użyte tylko i wyłącznie w przypadku, gdy klucz prywatny nie został ujawniony. Podstawową różnicą pomiędzy odnowieniem certyfikatu a aktualizacją kluczy jest to, że podczas aktualizacji musi zostać wygenerowana nowa para kluczy.

W związku z brakiem istnienia globalnego organu certyfikacji powstało wiele niezależnych organów certyfikacji. **Certyfikacja wzajemna** pozwala użytkownikom jednego CA ufać certyfikatom wystawionym przez inne CA. Istnieje certyfikacja jednokierunkowa oraz dwukierunkowa.

Unieważnianie certyfikatu to mechanizm ostrzegający pozostałych użytkowników, iż certyfikat nie potwierdza już tożsamości użytkownika. W tym przypadku identyfikator certyfikatu zostaje umieszczony na Liście Unieważnionych Certyfikatów. Do unieważnienia certyfikatu dochodzi w momencie, gdy:

- klucz prywatny został ujawniony,
- użytkownik końcowy zmienił nazwę,
- pracownik zakończył współpracę z firmą, która wystawiła mu certyfikat.

Mechanizm **odzyskiwania klucza** to zabezpieczenie pozwalające użytkownikowi odzyskać utracone klucze w przypadku, gdy:

- klucz prywatny znajduje się w miejscu niezmienionym, ale nie można uzyskać do niego dostępu (utrata hasła),
- nastąpiło zniszczenie nośnika - usterka dysku twardego lub nośnika klucza, np. karty elektronicznej.

Głównymi problemami występującymi w procesie odzyskiwania kluczy jest weryfikacja właściciela oraz ochrona kluczy przed atakującym. Jednym z rozwiązań jest zastosowanie kopii zapasowej. Stosowanie kopii zapasowej jest niezalecane dla kluczy podpisujących, gdyż zapewnienie niezaprzeczalności oraz integralności nie będzie już możliwe.

2.4.3 Protokoły bezpiecznej komunikacji

W poniższym rozdziale przedstawione zostały wybrane protokoły bezpiecznej komunikacji: SSL, TLS, S/MIME, SET, czy IPsec.

Protokół **Secure Socket Layer** [27] był jednym z pierwszych protokołów umożliwiających bezpieczną komunikację. Podstawowymi zadaniami tego protokołu jest zapewnienie prywatności oraz integralności przesyłanych informacji. Odpowiada również za uwierzytelnienie między klientem a serwerem w sieci WWW, której bezpieczeństwo zapewnia Infrastruktura Klucza Publicznego. Protokół składa się z dwóch warstw:

- *SSL Record Protocol* - odpowiada za zdefiniowanie formatu przesyłanych danych,

- *SSL Handshake Protocol* - pozwala na uwierzytelnienie pomiędzy serwerem i klientem oraz negocjację algorytmu szyfrującego i kluczy kryptograficznych.

Natomiast **Transport Layer Seciurity** [24] jest rozwinięciem protokołu SSL.

Protokół **Secure/Multipurpose Internet Mail Extensions** [22] zapewnia: autentyczność, integralność , niezaprzeczalność, prywatność oraz ochronę przesyłanych danych. S/MIME jest wykorzystywany do szyfrowania i deszyfrowania wiadomości w klientach poczty elektronicznej (np. Mozilla Thunderbird). Zanim klucz publiczny zostanie wykorzystany protokół S/MIME weryfikuje poprawność klucza publicznego oraz sprawdza, czy jego certyfikat nie znajduje się na Liście Unieważnionych Certyfikatów [21].

Z kolei protokół **Secure Electronic Transaction (SET)** [5] to protokół zapewniający bezpieczne transakcje elektroniczne w sieci WWW przy użyciu kart kredytowych. Posiada mechanizmy zapewniające poufność informacji oraz integralność danych. Jest on również wyposażony w funkcjonalność weryfikującą dane użytkownika. Podstawą działania protokołu SET jest certyfikat X.509.

Ostatni z protokołów - **IP Security** to w zasadzie zbiór protokołów, które umożliwiają bezpieczną wymianę klucza szyfrującego, a także zapewniają bezpieczną komunikację w Internecie. IPSec jest wykorzystany w tunelach VPN [26].

2.4.4 Modele zaufania

W infrastrukturze PKI stosowanych jest wiele modeli zaufania począwszy od ścisłej hierarchii certyfikatów, aż po zaufanie oparte na użytkowniku.

Ścisła hierarchia urzędów certyfikacji [5] to model zaufania przedstawiany za pomocą drzewa. W korzeniu znajduje się Główny CA *(ang. root CA)*. Główny CA jest początkiem architektury, ale również podstawą zaufania. Niżej w wierzchołkach znajdują się pośrednie CA certyfikowane przez Główny CA lub inne pośrednie CA, znajdujące się wyżej w hierarchii. Natomiast w liściach znajdują się użytkownicy końcowi (jednostki nie będące CA). Każdy z elementów infrastruktury jest certyfikowany przez jeden CA oraz posiada klucz publiczny Głównego CA.
Weryfikacja certyfikatów w tym modelu odbywa się następująco:

1. Załóżmy, że certyfikat Macieja jest certyfikowany przez pośrednie CA, tj. CA_2, natomiast certyfikat CA_2 przez CA_1 (kolejne pośrednie CA), którego certyfikat jest podpisany przez Główne CA.

2. Karolina posiada klucz publiczny Głównego CA, więc może zweryfikować certyfikat CA_1 i pozyskać kopię jego klucza publicznego.

3. Gdy Karolina posiada już kopię klucza publicznego CA_1, może w podobny sposób uzyskać kopię klucza publicznego CA_2.

4. Posługując się kopią klucza publicznego CA_2, Karolina może zweryfikować poprawność certyfikatu Macieja.

Luźna hierarchia urzędów certyfikacji [8] jest uproszczeniem hierarchii ścisłej. W przypadku, gdy użytkownicy posiadają certyfikaty podpisane przez tę samą jednostkę certyfikującą, nie ma potrzeby odwoływania się do Głównego CA.

Hierarchie oparte na regułach [8] również rozszerzają ścisłą hierarchię. Podstawową różnicą jest to, że certyfikaty jednostek końcowych oraz certyfikaty pośrednich CA mogą być podpisane przez więcej niż jeden urząd certyfikacji.

Kolejnym modelem zaufania jest **rozproszona architektura zaufania** [8], w tym modelu muszą występować, co najmniej dwa niezależne Główne CA tworzące własną infrastrukturę zaufania. Domeny PKI niezależnych CA mogą być zbudowane w sposób hierarchiczny lub równorzędny (brak nadrzędnych oraz podrzędnych CA). Pomiędzy równorzędnymi CA możemy ustanawiać zaufanie poprzez:

- **Certyfikację wzajemną**,
- **Wzajemne uznanie *(ang. cross-recognition)*** - niezależne domeny PKI są akredytowane przez zaufaną trzecią stronę, dzięki czemu mogą sobie wzajemnie ufać,

- **Listę Zaufanych Certyfikatów *(ang. Certificate Trust List)*** - lista certyfikatów podpisana przez zaufaną jednostkę [34],
- **Certyfikat akredytacji** - poręczenie dobrze znanego CA, że inne CA jest godne zaufania. Akredytacja powstaje w oparciu o jednokierunkową certyfikację wzajemną, bez wprowadzania hierarchii.

Podstawowymi konfiguracjami w modelu rozproszonym są [8]:

- **Konfiguracja siatki** - to konfiguracja, w której poszczególne równorzędne, niezależne Główne CA certyfikują się wzajemnie. Możliwe jest stworzenie pełnego (każdy z każdym) lub częściowego modelu zaufania. Wadą tego rozwiązania jest słaba skalowalność, gdy chcemy zastosować model pełnego zaufania. Posiadając n Głównych CA musimy stworzyć $n(n-1)$ połączeń.
- **Konfiguracja z koncentratorem** - każdy z Głównych CA ufa Centralnemu CA, nazywanym dalej koncentratorem. Konfiguracja ta nazywana jest również konfiguracją pomostową. Zadaniem koncentratora jest połączenie niezależnych domen PKI. Model z koncentratorem nie wprowadza hierarchii. Podstawową różnica miedzy modelem hierarchicznym a pomostowym jest fakt, iż jednostki końcowe nie posiadają kopii klucza publicznego koncentratora. Przewagą konfiguracji pomostowej nad konfiguracją siatki jest lepsza skalowalność - posiadając n Głównych CA, potrzebujemy n połączeń z koncentratorem.

W **Czworokątnym modelu zaufania** [5] występują cztery podstawowe postacie: użytkownik, CA użytkownika, strona oraz CA strony. Użytkownik oraz strona, wykonując między sobą transakcję, w celu uzyskania poświadczenia o wiarygodności, kontaktują się ze swoimi CA, które wykonują całą pracę związaną z autoryzacją i weryfikacją. Model ten jest szeroko wykorzystywany w elektronicznych transakcjach bankowych.

W **modelu sieci WWW** [8] każda przeglądarka internetowa posiada preinstalowaną listę certyfikatów publicznych, którym może ufać.

Ostatnim przedstawianym modelem jest **model zaufania oparty na użytkowniku**. W modelu tym użytkownik tworzy własną sieć certyfikatów, którym ufa. Decyzja odnośnie zaufania jest podejmowana przez użytkownika końcowego. Użytkownik swój wybór może opierać na osobistej znajomości lub poprzez *sieć zaufania.*

2.5 Standardy PKCS

Standardy określone wspólnym mianem **Public-Key Cryptography Standards (PKCS)** są zbiorem standardów kryptograficznych, umożliwiających bezpieczną wymianę klucza publicznego w Internecie przy użyciu Infrastruktury Klucza Publicznego (PKI) [41]. Pierwsze standardy PKCS powstały na początku lat 90. i były wynikiem współpracy pomiędzy firmą RSA Laboratories, kryptografami oraz informatykami z całego świata [57].

2.5.1 Standard PKCS#7

Standard PKCS#7: Cryptographic Message Syntax [20] to standard wykorzystywany do generowania i weryfikacji cyfrowych podpisów i certyfikatów zarządzanych przez PKI. Standard proponuje składnię i format do tworzenia podpisów cyfrowych.

2.5.2 Standard PKCS#11

Standard PKCS#11: Cryptographic Token Interface Standard [29] określa API, nazwane *Cryptoki*, dla urządzeń pozwalających przechowywać dane kryptograficzne oraz wykonywać funkcje kryptograficzne. *Cryptoki* jest niezależne technologicznie, co oznacza że może być wykorzystywane przez urządzenia różnego typu i umożliwia współdzielenie zasobów. Ważnym dla aplikacji oraz bibliotek pracujących z tokenami PKCS#11 jest akceptowanie wspólnych identyfikatorów, którymi posługuje się użytkownik, tak by znaleźć przechowywane obiekty PKCS#11 na tokenach PKCS#11. Obiekty, które mogą być przechowywane to: certyfikaty klucza publicznego, dane, klucze publiczne, klucze prywatne, klucze szyfrujące. Obiekt może być opisany za pomocą unikalnego identyfikatora, etykiety oraz typu.

2.5.3 Standard PKCS#12

Standard PKCS#12: Personal Information Exchange Syntax [28] to standard opisujący składnię transferu oraz sposób przechowywania danych osobistych takich jak: klucze prywatne, certyfikaty oraz rozszerzenia. Urządzenia oraz aplikacje wspierające ten standard pozwalają użytkownikowi na importowanie, eksportowanie oraz wykorzystywanie tych danych. Standard ten posiada wiele trybów zapewniających integralność oraz prywatność, pozwalających na bezpieczny transfer danych osobistych. W jednym z trybów integralności i prywatności wymagane jest, by na platformie źródłowej oraz docelowej znajdowała się zaufana para kluczy. Wspierane są również tryby integralności oparte na haśle. Są one wykorzystywane głównie wtedy, gdy para kluczy nie jest dostępna. PKCS#12 powinien być kompatybilny dla implementacji sprzętowej, jak i software'owej. Implementacja sprzętowa pozwala na przechowywanie danych na fizycznych urządzeniach, takich jak karty elektroniczne.

2.5.4 Standard PKCS#15

Standard PKCS#15 Cryptographic Token Information Syntax Standard [18] jest standardem zawierającym informacje dotyczące dwóch grup urządzeń: tokenów sprzętowych (np. karty inteligentne) oraz tzw. *soft-tokens* [43]. Standard ten określa wspólny format przechowywania danych cyfrowych na tokenach wykorzystywanych do uwierzytelnienia i autoryzacji. Standard PKCS#15 umożliwił certyfikatom pochodzącym od dowolnego dostawcy weryfikację danych użytkownika w dowolnej aplikacji na dowolnym urządzeniu. W styczniu 2004 roku część dotycząca kart elektronicznych została przeniesiona do standardu ISO7816-15 [48].

2.5.5 Norma ISO7816-15

Podstawowe informacje o normie ISO 7816 [14] oraz krótkie wprowadzenie do części piętnastej normy zostały opisane w rozdziale 2.1.2. Piętnasta część normy definiuje wspólną składnię i format informacji kryptograficznych oraz mechanizmy do udostępniania tych informacji w razie potrzeby. Cele tej części normy przedstawiają się następująco:

- ułatwienie interoperacyjności pomiędzy komponentami działającymi na różnych platformach,
- umożliwienie aplikacjom korzystanie z produktów i komponentów wielu producentów (neutralność dla zewnętrznych dostawców),
- umożliwienie korzystania z postępów technologicznych bez konieczności przepisywania kodu aplikacji na nowo (neutralność aplikacji),
- zachowanie spójności z powiązanymi normami oraz rozszerzanie ich tylko jeżeli jest to konieczne i praktyczne.

Ponadto piętnasta część normy definiuje:

- przechowywanie wielu informacji kryptograficznych na karcie,
- wykorzystywanie tych informacji kryptograficznych,
- pobieranie informacji kryptograficznych,
- powiązanie informacji kryptograficznych z obiektami zdefiniowanymi w innych częściach normy,
- różne mechanizmy uwierzytelnienia,
- wiele algorytmów kryptograficznych.

ISO 7816-15 nie obejmuje wewnętrznej implementacji. W przypadku implementacji zgodnych z niniejszą normą nie ma obowiązku wspierania wszystkich wyżej wymienionych opcji.

2.6 Komunikacja z kartą

Komunikacja z kartą jest realizowana za pomocą struktury APDU *(ang. Application Protocol Data Unit)*, będącej strukturą danych w protokole komunikacyjnym pomiędzy kartą a czytnikiem. Ramka APDU jest określona poprzez normę ISO/IEC 7816-4 [14]. APDU można podzielić na dwie kategorie:

- polecenie APDU - zapytanie wysyłane za pośrednictwem czytnika do karty,
- odpowiedź APDU - struktura stanowiąca odpowiedź na wydane zapytanie (wysłane przez kartę do czytnika).

Rozdział 3

Przegląd istniejących rozwiązań

W tym rozdziale został zawarty przegląd wybranych rozwiązań w zakresie oprogramowania, które można zainstalować na kartach elektronicznych oraz nośnikach stosowanych dla oprogramowania tego typu. Przedstawiono także uzasadnienie wyboru apletu, który został wykorzystany podczas realizacji tej pracy.

3.1 Nośniki certyfikatów

Wśród nośników certyfikatów należy wyróżnić kilka typów urządzeń [50]:

- karta elektroniczna,
- token USB,
- Sprzętowy moduł bezpieczeństwa *(ang. Hardware security module)*.

W ramach pracy zostały wykorzystane karty elektroniczne, gdyż są one używane jako elektroniczne legitymacje studenckie i w zasadzie stanowią jeden z najlepszych wyborów do tego celu. Jednakże do przenoszenia certyfikatów oraz kluczy możemy posłużyć się również tokenami USB oraz modułami HSM.

3.2 Oprogramowanie

Oprogramowanie, które poddano analizie występuje w postaci apletów, czyli programów komputerowych o niewielkich wymaganiach zasobowych, które są wywoływane przez aplikację zewnętrzną i usuwane wraz z jej zamknięciem. Wykorzystanie technologii Java Card pozwala na pisanie apletów w języku Java oraz łatwe wykorzystanie ich na kartach elektronicznych.

Wśród apletów PKI dla kart elektronicznych można wyróżnić: aplety preinstalowane, M.U.S.C.L.E oraz isoApplet.

Pre-instalowane aplety PKI zostają wgrane na karty inteligentne w procesie ich produkcji. Wadą tego rozwiązania jest fakt, że nie wszystkie karty elektroniczne posiadają aplety preinstalowane przez producenta. Aplety tego typu nie mogą być rozwijane przez użytkownika. Mogą też występować problemy z dostępnością narzędzi pozwalających na komunikację z nimi.

M.U.S.C.L.E [36] to aplet, który został stworzony i był rozwijany przez Martina Paljaka. Autor nie zaleca jednak korzystania z tego apletu, ponieważ nie jest zgodny z obecnie obowiązującymi standardami.

IsoApplet [38] jest programem autorstwa Philipa Wendlanda. Aplet jest zgodny ze standardem ISO-7816 i dostępny w ramach licencji Open Source GNU General Public License v3.0, co pozwala na jego wykorzystanie oraz dalszy rozwój. Został stworzony przy użyciu technologii Java Card w wersji 2.2.2, co zapewnia mu aktualność oraz możliwość wykorzystania w wielu kartach elektronicznych. IsoApplet umożliwia generowanie par kluczy na karcie przy użyciu algorytmu RSA o długości klucza 2048 bitów, oparte o chińskie twierdzenie o resztach. Generowanie kluczy bezpośrednio na karcie gwarantuje, że klucz nie zostanie skompromitowany przed użyciem, a algorytm kryptograficzny RSA pozwoli zapobiec późniejszej kompromitacji klucza. Dodatkowo wykorzystanie chińskiego twierdzenia o resztach przyspiesza działanie algorytmu RSA. Aplet pozwala również na wykorzystywanie kryptografii krzywych eliptycznych oraz umożliwia import

kluczy prywatnych, kluczy publicznych i certyfikatów. Niewątpliwą zaletą tego rozwiązania jest wieloplatformowy middleware, który daje możliwość łatwego korzystania z funkcji apletu. Wraz z middlewarem dostępna jest biblioteka DLL, która pozwala korzystać z obiektów znajdujących się na karcie przez programy zewnętrzne takie jak: VeraCrypt, Mozilla Thunderbird czy Mozilla Firefox. IsoApplet został przetestowany na wielu kartach elektronicznych.

3.3 Wykorzystane karty elektroniczne

Niezbędnym elementem pracy było odpowiednie przetestowanie możliwości technicznych kart elektronicznych oraz przeanalizowanie ich parametrów.

3.3.1 Aplet testujący

Sprawdzenie parametrów oraz możliwości kart elektronicznych jest możliwe, dzięki narzędziu JCAlgTest [49]. Aplet jest dostępny w ramach licencji Creative Commons Attribution 4.0 International License. Program został zaimplementowany przez członków organizacji *CRoCS* [33]. Narzędzie to jest przeznaczone dla kart elektronicznych wykorzystujących technologię Java Card i umożliwia automatyczne sprawdzenie funkcji kryptograficznych karty oraz jej parametrów takich jak: ilość wolnej pamięci RAM oraz ilość wolnej pamięci EEPROM.
Zainicjowanie procesu ładowania oraz instalacji apletu testującego w wersji 1.2 dla Java Card 2.2.2 jest możliwe przy użyciu narzędzia GlobalPlatformPro [35] poprzez wydanie w konsoli polecenia:

```
gp -install [-emv] AlgTest_v1.2_jc2.2.2.cap
```

Aby wykonać test karty elektronicznej należy w wierszu polecenia wywołać komendę:

```
java -jar AlgTestJClient.jar
```

Konsola zapyta użytkownika, jakie testy chciałby wykonać. Wybór jest możliwy poprzez podanie odpowiadającej testowi wartości numerycznej.

3.3.2 Parametry kart

W ramach pracy testowano karty elektroniczne trzech producentów: JCOP, Oberthur oraz Giesecke & Devrient. Dla każdego typu kart przedstawiono jej odpowiedź na sygnał reset, czyli ATR *(ang. Answer to reset)*, obsługiwane przez kartę funkcje skrótu i algorytmy kryptograficzne wraz z długościami kluczy.

Komunikat ATR wygenerowany przez kartę JCOP ma następującą postać:

ATR karty: 3B F8 13 00 00 81 31 FE 45 4A 43 4F 50 32 30 31 38 FD

Algorytm	SHA	MD5	SHA_256	SHA_384	SHA_512	SHA_224
Obsługiwany	TAK	TAK	TAK	NIE	NIE	NIE

Tabela 3.1: Wspierane funkcje skrótu przez kartę JCOP

Algorytm	**Długość klucza**
DES	64
3DES	128, 192
AES	128, 192, 256
RSA_CRT	512, 768, 896, 1024, 1280, 1536, 1984, 2048
EC_FP	128, 160, 192, 224, 256

Tabela 3.2: Wspierane algorytmy kryptograficzne przez kartę JCOP

W tabeli 3.1 przedstawione zostały obsługiwane przez kartę funkcje skrótu, natomiast w tabeli 3.2 obsługiwane algorytmy kryptograficzne. Jak widać, wygenerowanie pary kluczy przy użyciu podstawowego algorytmu RSA nie jest możliwe, natomiast klucz prywatny oraz publiczny mogą być

wygenerowane przy użyciu algorytmu RSA opartego o chińskie twierdzenie o resztach. Karta JCOP pozwala nam również na stworzenie pary kluczy przy użyciu kryptografii krzywych eliptycznych.

Rodzaj pamięci	**Dostępna pamięć**
RAM	1757 B
EEPROM	63686 B

Tabela 3.3: Ilość pamięci możliwa do wykorzystania na karcie JCOP

Przedstawione w tabeli 3.3 wartości odnoszą się do ilości wolnej pamięci przed dokonaniem jakichkolwiek zmian na karcie.

Komunikat wygenerowany przez kartę Oberthur ma następującą postać:

ATR karty: 3B DD 18 00 81 31 FE 45 80 F9 A0 00 00 00 77 01 08 00 07 90 00 FE

Algorytm	SHA	MD5	SHA_256	SHA_384	SHA_512	SHA_224
Obsługiwany	TAK	NIE	TAK	TAK	TAK	NIE

Tabela 3.4: Wspierane funkcje skrótu przez kartę Oberthur

Algorytm	**Długość klucza**
DES	64
3DES	128, 192
AES	128, 192, 256
RSA	1024, 1280, 1536, 2048
RSA_CRT	1024, 1280, 1536, 2048
EC_FP	192, 224, 256, 384, 521

Tabela 3.5: Wspierane algorytmy kryptograficzne przez kartę Oberthur

W tabeli 3.4 przedstawione zostały obsługiwane przez kartę funkcje skrótu, natomiast w tabeli 3.5 zostały zobrazowane obsługiwane przez kartę algorytmy kryptograficzne. W przeciwieństwie do karty JCOP na karcie Oberthur jest możliwe wygenerowanie pary kluczy zarówno przy pomocy podstawowego algorytmu RSA jak i algorytmu RSA opartego o chińskie twierdzenie o resztach. Podobnie jak w przypadku karty JCOP możliwe jest wygenerowanie pary kluczy przy użyciu kryptografii krzywych eliptycznych.

Rodzaj pamięci	**Dostępna pamięć**
RAM	1275 B
EEPROM	29472 B

Tabela 3.6: Ilość pamięci możliwa do wykorzystania dla karty Oberthur

Przedstawione w tabeli 3.6 wartości odnoszą się do ilości wolnej pamięci przed dokonaniem jakichkolwiek zmian na karcie.

Komunikat wygenerowany przez kartę Giesecke & Devrient ma następującą postać:

ATR karty: 3B 6D 00 00 00 73 C8 00 13 64 54 37 43 35 00 90 00

Algorytm	SHA	MD5	SHA_256	SHA_384	SHA_512	SHA_224
Obsługiwany	TAK	NIE	NIE	NIE	NIE	NIE

Tabela 3.7: Wspierane funkcje skrótu przez kartę Giesecke & Devrient

Algorytm	**Długość klucza**
DES	64
3DES	128, 192

Tabela 3.8: Wspierane algorytmy kryptograficzne przez kartę Giesecke & Devrient

Odpowiednio w tabeli 3.7 oraz tabeli 3.8 zostały przedstawione funkcje skrótu oraz algorytmy kryptograficzne obsługiwane przez kartę Giesecke & Devrient. Jak widać, nie ma możliwości wygenerowania pary kluczy przy użyciu algorytmów kryptografii asymetrycznej na karcie Giesecke & Devrient.

Rodzaj pamięci	**Dostępna pamięć**
RAM	2334 B
EEPROM	21737 B

Tabela 3.9: Fabrycznie dostępna ilość pamięci dla karty Giesecke & Devrient

Przedstawione w tabeli 3.9 wartości odnoszą się do ilości wolnej pamięci przed dokonaniem jakichkolwiek zmian na karcie.

3.4 Wybrane rozwiązanie

W dalszej części pracy zostały wykorzystane karty: **JCOP** oraz **Oberthur**. Karta Giesecke & Devrient została odrzucona z powodu braku możliwości wygenerowania na niej par kluczy kryptografii asymetrycznej. Spośród apletów wybrano aplet **isoApplet**. Jego wysoka funkcjonalność, zgodność ze współczesnymi standardami dotyczącymi kart elektronicznych, biblioteka DLL, łatwy w obsłudze middleware, a także licencja pozwalająca na dowolne użycie kodu apletu zadecydowała o jego wyborze do realizacji pracy.

Rozdział 4

Aplet PKI

Aplet, o którym mowa w tym rozdziale to program napisany przy użyciu środowiska programistycznego Java Card, pozwalającego na uruchamianie aplikacji opartych o język Java w ramach środowiska sprzętowego kart elektronicznych. W pracy został wykorzystany aplet stworzony przez Philipa Wendlanda [38], dostępny w ramach licencji Open Source GNU General Public License v3.0. Niewątpliwą zaletą wyboru tego apletu jest istnienie przystosowanej wersji biblioteki OpenSC, stanowiącej główny filar middleware'u.

4.1 Implementacja

Aplet został przygotowany w języku Java Card, będącym swoistą odmianą języka Java. Projekt apletu jest podzielony na poszczególne paczki, a główna funkcja *main* znajduje się w pliku *IsoApplet.java*.
Do realizacji pracy konieczne były zmiany w kodzie źródłowym apletu. Pierwszą z nich było ustawienie flagi zezwolenia na zapis kluczy prywatnych na karcie elektronicznej. Domyślnie aplet pozwala jedynie na korzystanie z kluczy wygenerowanych przez kartę. Aby to zmienić należy ustawić zmienną *DEF_PRIVATE_KEY_IMPORT_ALLOWED* na wartość *true*, co reprezentuje poniższy fragment kodu:

```
public static final boolean DEF_PRIVATE_KEY_IMPORT_ALLOWED = true;
```

Druga zmiana dotyczy wymuszenia ustawiania kodu PUK przy inicjalizacji. Zapewnia to poprawną współpracę apletu z zasobnikiem systemu Windows i umożliwia reset kodu PIN bez konieczności reinstalacji apletu. Do wykonania tej zmiany konieczna jest zmiana wartości zmiennej *PUK_MUST_BE_SET* na wartość *true*, co reprezentuje linijka poniżej:

```
private static final boolean PUK_MUST_BE_SET = true;
```

Poza ustawieniami w kodzie apletu, które wymagają jego rekompilacji przed instalacją, oprogramowanie biblioteki OpenSC pozwala na definiowanie części ustawień poprzez pliki profili. Profile są prostymi plikami tekstowymi w formacie *.profile*, znajdującymi się w katalogu instalacyjnym biblioteki w podfolderze *profiles*. Zakres ustawień, które oferują, jest zależny od implementacji apletu. Zdefiniowane ustawienia są przenoszone podczas procesu inicjalizacji. Ustawienia zapisane w profilu apletu isoApplet pozwalają decydować o maksymalnej i minimalnej długości kodu PIN, ustawieniach kodów PIN i PUK dla tzw. Security Officera, o ile ta funkcjonalność jest używana. Ponadto możliwe jest również zdefiniowanie przyjaznej nazwy apletu i producenta. Informacje te można wylistować z pomocą biblioteki OpenSC.

4.2 Kompilacja

Kompilację apletu przeprowadzono z pomocą narzędzia Ant [31] przeznaczonego do zautomatyzowania procesu budowy oprogramowania. Plik *build.xml* definuje zależności procesu budowy. Jedyną zmianę, jaka była konieczna do wykonania, stanowiła podmiana ścieżki do oprogramowania Java Card SDK 2.2.2, czego dokonano przez następujący wpis w odpowiednim miejscu tego pliku:

```
<cap jckit="C:\Program Files\javacard\" aid="f2:76:a2:88:bc:fb:a6:9d:34:f3:10" ...
```

Po dokonanych zmianach należy wydać z poziomu konsoli polecenie *Ant*.

4.3 Ładowanie i instalacja na kartach elektronicznych

Proces ładowania i instalacji wybranego apletu na kartach elektronicznych wymagał spełnienia pewnych warunków przez środowisko, a także przeprowadzenia odpowiednich działań.

4.3.1 Wymagania przedinstalacyjne

Proces ładowania i instalacji apletu wymaga oprogramowania GlobalPlatformPro [35] oraz zainstalowanej biblioteki OpenSC [37]. W systemach 32 bitowych należy zainstalować wersję 32-bitową biblioteki, w systemach 64-bitowych należy zainstalować zarówno wersję 64-bitową jak i 32-bitową. Wynika to z faktu, że 32-bitowa aplikacja Java Card pracująca na systemie 64-bitowym, wymaga 32-bitowej wersji mini-sterownika i 32-bitowej wersji modułu PKCS. Ponadto wymagany jest również czytnik kart elektronicznych wraz z zainstalowanymi sterownikami. Do realizacji pracy został wykorzystany czytnik GemPC Twin, którego producentem jest firma Gemalto.

4.3.2 Proces ładowania oraz instalacji

Pierwszą czynnością konieczną do realizacji zadania jest umieszczenie karty elektronicznej w czytniku. Zainicjowanie procesu ładowania oraz instalacji apletu jest możliwe poprzez wydanie z poziomu konsoli polecenia:

```
gp -install IsoApplet.cap -default
```

lub w przypadku karty Oberthur:

```
gp -emv -install IsoApplet.cap -default
```

Użyte w tych poleceniach parametry default i emv oznaczają odpowiednio:

- $-default$ - umożliwia wskazanie domyślnych uprawnień dla apletu,
- $-emv$ - umożliwia wykorzystanie dywersyfikacji EMV, która jest niezbędna do instalacji apletu na karcie Oberthur.

4.3.3 Inicjalizacja struktur PKCS#15

Do poprawnego działania apletu konieczna jest inicjalizacja struktury plików standardu PKCS#15. Proces ten jest możliwy do wykonania poprzez wydanie z poziomu konsoli polecenia:

```
pkcs15-init --create-pkcs15
```

Polecenie spowoduje przydział pamięci i utworzenie odpowiedniej struktury plików. Po poprawnym zakończeniu tego procesu oprogramowanie poprosi o podanie kodu PIN oraz jego potwierdzenie. Zgodnie z ustawieniami apletu kod PIN powinien mieć od 4 do 16 znaków. Następnie konieczne jest ustawienie kodu PUK, który powinien składać się z co najmniej 16 znaków. W następnym rozdziale została przedstawiona obsługa apletu oraz jego wykorzystanie w aplikacjach klienckich.

Rozdział 5

Middleware dla apletu PKI

Middleware dla apletu stanowi oprogramowanie pośredniczące pomiędzy aplikacją zainstalowaną na karcie elektronicznej a systemem operacyjnym komputera. Middleware komunikuje się z kartą poprzez polecenia APDU, dzięki czemu użytkownik w sposób komfortowy może wydawać polecenia karcie, nie musząc za każdym razem definiować szczegółowych zapytań.

5.1 Opis implementacji

Middleware dla apletu został zrealizowany w formie programów wsadowych systemu Windows, opierając się o narzędzia biblioteki OpenSC. Skrypty wpływają na ustawienia systemu, definiując zmienne systemowe, jednakże nie wpływa to na jego ogólną pracę.

5.2 Instrukcja uruchomienia

Przedstawione w dalszej części tego rozdziału skrypty mogą zostać uruchomione bezpośrednio (poprzez dwuklik myszą) lub wywołane z wiersza poleceń konsoli. Poza skryptem dotyczącym dodatkowej zmiennej systemowej, nie wymagają one uprawnień administracyjnych.

Do poprawnego działania, middleware wymaga zainstalowanej biblioteki OpenSC, co zostało opisane szerzej w podrozdziale 4.3.1.

Dla pewnych typów kart wymagane jest ograniczenie sterowników (profili) dostępnych w systemie poprzez ustawienie zmiennej systemowej *OPENSC_DRIVER*. Taka sytuacja ma miejsce między innymi dla kart produkowanych przez firmę Oberthur, w przypadku których middleware wykrywa wśród wartości zwróconych z karty komunikatem ATR jej producenta i próbuje połączyć się za pomocą fabrycznego apletu IAS. Niestety, taka operacja zakończy się błędem. Aby takiej sytuacji uniknąć przygotowany został również osobny skrypt, będący swoistym sterownikiem dla apletu, który wykona operację ustawienia zmiennej systemowej *OPENSC_DRIVER*. Jego uruchomienie może okazać się konieczne celem skorzystania z magazynu certyfikatów na karcie przez aplikacje zewnętrzne, takie jak programy pocztowe czy programy do szyfrowania danych.

5.3 Zawartość poszczególnych elementów middleware

W rozdziale przedstawiono funkcje middleware, realizowane przez poszczególne skrypty wsadowe systemu Windows.

5.3.1 Ustawienie domyślnego wyboru apletu

Do poprawnej obsługi apletu przez aplikacje zewnętrzne konieczna jest konfiguracja zmiennej środowiskowej systemu *OPENSC_DRIVER*. Aplikacja zewnętrzna rozpoczyna komunikację z kartą poprzez próby komunikacji z poszczególnymi apletami z listy w zmiennej środowiskowej. Jeżeli karta będzie mieć swój aplet producenta (jak na przykład karty produkcji Oberthura) i aplet ten będzie wyżej na liście, aplikacja może błędnie wybrać komunikację za jego pomocą. Z tego powodu należy dokonać zmiany w powyższej zmiennej. Poniższy skrypt dodaje zmienną środowiskową systemu *OPENSC_DRIVER*. Do tego celu wymagane są uprawnienia administracyjne. Aby skrypt został wykonany poprawnie, należy uruchomić go opcją *"Uruchom jako administrator"*.

```
@echo off
SETX OPENSC_DRIVER isoApplet /M
```

5.3.2 Testowanie poprawności instalacji apletu

Kolejny skrypt powłoki sprawdza poprawność instalacji apletu na karcie. Realizacja tego celu odbywa się poprzez wykonanie polecenia *opensc-tool -n*, które odpytuje kartę o jej nazwę (nazwę domyślnego apletu). Treść tego skryptu wygląda następująco

```
@echo off
SETLOCAL EnableDelayedExpansion
SET TOOL=opensc-tool.exe
SET TOOL_PATH="C:\Program Files (x86)\OpenSC Project\OpenSC\tools"
IF NOT EXIST %TOOL_PATH% (
    echo Error: OpenSC Tools not installed. Insert correct path to OpenSC tools:
    :set_tools
        set /p TOOL_PATH=
        set TOOL_PATH="!TOOL_PATH!"
        IF NOT EXIST !TOOL_PATH!\%TOOL% (
            echo Wrong path. Insert correct path to OpenSC tools:
            goto set_tools
        )
)
for /f "tokens=*" %%a in ('CALL %TOOL_PATH%\%TOOL% -n 2^> nul') do set res=%%a
:loop
IF "%res%"=="" (
    goto loop
)
if "%res%" == "Javacard with IsoApplet" (
    echo:
    echo SUCCESS: You're aplet works fine!
) else (
    echo:
    echo Error: Aplet not recognized
)
echo:
set /p EXIT=Hit ENTER to exit...
```

5.3.3 Zmiana kodu PIN

Zmiana kodu PIN jest realizowana poprzez narzędzie *pkcs15-tool* z przełącznikiem *–change-pin*, a także parametrami *–pin* oraz *–new-pin*. Po uruchomieniu narzędzia konsola poprosi o podanie dotychczasowego kodu PIN. Poprawność kodu PIN jest sprawdzana za pomocą polecenia *pkcs15-tool –verify-pin*. Jeżeli podany PIN nie jest prawidłowy, użytkownik zostanie poproszony o ponowne jego podanie. W przeciwnym wypadku, użytkownik zostanie poproszony o podanie nowego kodu PIN zgodnego z polityką bezpieczeństwa:

- musi składać się on z co najmniej 4 i co najwyżej 16 znaków - polityka zdefiniowana wewnątrz kodu apletu,
- musi składać się on z co najmniej: jednej wielkiej litery, jednej małej litery, jednej cyfry - realizowane za pomocą potoku oraz odpowiedniego wyrażenia regularnego.

W przypadku, gdy użytkownik nie zrealizuje wymaganej polityki bezpieczeństwa, zostanie on poproszony o ponowne podanie starego kodu PIN, a także nowego kodu PIN.

```
@ECHO off
SETLOCAL EnableDelayedExpansion
SET TOOL_1=pkcs15-init.exe
SET TOOL_2=pkcs15-tool.exe
```

```
SET TOOL_PATH="C:\Program Files (x86)\OpenSC Project\OpenSC\tools"
IF NOT EXIST %TOOL_PATH% (
    ECHO Error: OpenSC Tools not installed. Insert correct path to OpenSC tools:
    :SET_TOOLS
        SET /p TOOL_PATH=
        SET TOOL_PATH="!TOOL_PATH!"
        IF NOT EXIST !TOOL_PATH!\%TOOL_1% (
            ECHO Wrong path. Insert correct path to OpenSC tools:
            goto SET_TOOLS
        )
)

SET UPPERCASE=[ABCDEFGHIJKLMNOPQRSTUVWXYZ]
SET LOWERCASE=[abcdefghijklmnopqrstuvwxyz]
:TRY_AGAIN
        ECHO Please type below your current pin:
        SET /p old=
        %TOOL_PATH%\%TOOL_2% --verify-pin --pin %old% 2>NUL
        IF /I "%ERRORLEVEL%" EQU "0" (
                ECHO Pin is correct!
                ECHO New pin must contain:
                ECHO ===== more than 4 characters and less than 16 characters
                ECHO ===== at least: one capital, one lowercase, one numeric
                ECHO Please type below your new pin:
                SET /p new=

                FOR /F "tokens=* USEBACKQ" %%F IN (`ECHO !new! ^|^
                findstr /r !UPPERCASE! ^|^
                findstr /r !LOWERCASE! ^|^
                findstr /r [0-9]`) DO (
                        SET is_empty=%%F
                )

                IF "!is_empty!"=="" (
                        ECHO New password does not contain:
                        ECHO ===== capital
                        ECHO ===== lowercase
                        ECHO ===== digit...
                        ECHO Please try again...
                        GOTO :TRY_AGAIN
                ) ELSE (
                        !TOOL_PATH!\!TOOL_2! --change-pin --pin !old! ^
                        --new-pin !new! 2>NUL
                        IF /I "!ERRORLEVEL!" EQU "0" (
                                ECHO New password has been set!
                        ) ELSE (
                                ECHO New password is too long or too short
                                ECHO Please try again...
                                GOTO :TRY_AGAIN
                        )

                )
        ) ELSE (
                ECHO Typed pin is incorrect please try again...
                GOTO :TRY_AGAIN
        )

SET /p EXIT=Hit ENTER to exit...
```

5.3.4 Odblokowanie kodu PIN

Odblokowanie kodu PIN jest realizowane poprzez narzędzie *pkcs15-tool* z przełącznikiem *–unblock-pin* oraz parametrem *–new-pin*. Po uruchomieniu narzędzia użytkownik zostanie poproszony o podanie nowego kodu PIN, który musi być zgodny z polityką bezpieczeństwa omówioną w podrozdziale 5.3.3. W przypadku, gdy użytkownik nie zrealizuje wymaganej polityki bezpieczeństwa, zostanie poproszony o ponowne podanie innego kodu PIN, zgodnego z polityką. W następnym kroku należy podać kod PUK karty. W przypadku podania niepoprawnego kodu PIN, użytkownik zostanie o tym poinformowany odpowiednim komunikatem i zostanie poproszony o rozpoczęcie całego procesu od początku. Zatwierdzenie poprawnego kodu PUK, spowoduje zmianę kodu PIN.

```
@ECHO off
SETLOCAL EnableDelayedExpansion
SET TOOL_1=pkcs15-init.exe
SET TOOL_2=pkcs15-tool.exe
SET TOOL_PATH="C:\Program Files (x86)\OpenSC Project\OpenSC\tools"
IF NOT EXIST %TOOL_PATH% (
    ECHO Error: OpenSC Tools not installed. Insert correct path to OpenSC tools:
    :SET_TOOLS
        SET /p TOOL_PATH=
        SET TOOL_PATH="!TOOL_PATH!"
        IF NOT EXIST !TOOL_PATH!\%TOOL_1% (
            ECHO Wrong path. Insert correct path to OpenSC tools:
            goto SET_TOOLS
        )
)

SET UPPERCASE=[ABCDEFGHIJKLMNOPQRSTUVWXYZ]
SET LOWERCASE=[abcdefghijklmnopqrstuvwxyz]
:TRY_AGAIN
        ECHO Please type new PIN
        ECHO New pin must contain:
        ECHO ===== more than 4 characters and less than 16 characters
        ECHO ===== at least: one capital, one lowercase, one numeric
        ECHO Please type below your new pin:
        SET /p new=

        FOR /F "tokens=* USEBACKQ" %%F IN (`ECHO !new! ^|^
        findstr /r !UPPERCASE! ^|^
        findstr /r !LOWERCASE! ^|^
        findstr /r [0-9]`) DO (
                SET is_empty=%%F
        )

        IF "!is_empty!"=="" (
                ECHO New password does not contain capital, lowercase or digit...
                ECHO Please try again...
                GOTO :TRY_AGAIN
        ) ELSE (
                !TOOL_PATH!\!TOOL_2! --unblock-pin --new-pin !new! 2>NUL
                ECHO.
                IF /I "!ERRORLEVEL!" EQU "0" (
                        ECHO New password has been set!
                ) ELSE (
                        ECHO New password is too long or too short or
                        ECHO Typed PUK code is incorrect
                        ECHO Please try again...
                        GOTO :TRY_AGAIN
                )
```

```
        )


SET /p EXIT=Hit ENTER to exit...
```

5.3.5 Usunięcie kluczy

Omówiony w tej sekcji skrypt powłoki umożliwia usunięcie z karty wybranych obiektów. Implementacja tej funkcjonalności została oparta o instrukcję wyboru *switch ... case*. Użytkownik może wybrać jedną z następujących opcji:

- **privkey** - umożliwia usunięcie klucza prywatnego,
- **pubkey** - umożliwia usunięcie klucza publicznego,
- **pair** - umożliwia usunięcie pary kluczy, w przypadku gdy jeden z elementów z pary nie istnieje, drugi zostaje usunięty,
- **cert** - umożliwia usunięcie certyfikatu,
- **all** - umożliwia usunięcie pary kluczy oraz certyfikatu jednocześnie, w przypadku gdy jeden lub więcej elementów nie istnieje, to usunięte zostaną tylko dostępne elementy,
- **exit** - zakończenie skryptu.

W przypadku wpisania nieprawidłowej opcji użytkownik zostanie o tym poinformowany i poproszony o ponowne wybranie jednej z dostępnych opcji.
Usunięcie obiektu jest możliwe przy pomocy polecenia *pkcs15-init -D <arg> –id <id>*, gdzie w miejsce <arg> użytkownik podaje odpowiedni typ obiektu, a w miejsce <id> właściwy identyfikator. Możliwe jest jednoczesne usunięcie kilku obiektów o takim samym numerze identyfikacyjnym i różnym typie, gdy w miejsce <arg> podane zostaną typy obiektów oddzielone przecinkami. Każda próba usunięcia elementu jest poprzedzona wyświetleniem dostępnych obiektów wybranego wcześniej typu wraz z ich identyfikatorami, co pozwala użytkownikowi skasować odpowiedni element.

```
@ECHO off
SETLOCAL EnableDelayedExpansion
SET TOOL_1=pkcs15-init.exe
SET TOOL_2=pkcs15-tool.exe
SET TOOL_PATH="C:\Program Files (x86)\OpenSC Project\OpenSC\tools"
IF NOT EXIST %TOOL_PATH% (
    ECHO Error: OpenSC Tools not installed. Insert correct path to OpenSC tools:
    :SET_TOOLS
        SET /p TOOL_PATH=
        SET TOOL_PATH="!TOOL_PATH!"
        IF NOT EXIST !TOOL_PATH!\%TOOL_1% (
            ECHO Wrong path. Insert correct path to OpenSC tools:
            GOTO SET_TOOLS
        )
)
:MAIN_LOOP
        ECHO Choose an element to remove
        ECHO (type: privkey, pubkey, pair, cert, all) or exit:
        SET /P TYPE=
        ECHO Option selected : %TYPE%
        2>NUL CALL :CASE_%TYPE% rem jump to :CASE_privkey, :CASE_pubkey, etc.
        IF ERRORLEVEL 1 CALL :DEFAULT_CASE rem If label doesn't exist
        EXIT /B
        :CASE_privkey
                ECHO ========== [ LIST OF PRIVATE KEYS] ==========
                %TOOL_PATH%\%TOOL_2% --list-keys ^
```

```
        | findstr /r /c:"^[A-Z]" /c:"^[^A-Z]*ID" 2>NUL
        ============================================
        SET /P ID="Choose one of listed ID to remove "
        !TOOL_PATH!\%TOOL_1% -D privkey --id %ID% 2>NUL
        GOTO MAIN_LOOP
:CASE_pubkey
        ECHO ========== [ LIST OF PUBLIC KEYS] ==========
        %TOOL_PATH%\%TOOL_2% --list-public-keys ^
        | findstr /r /c:"^[A-Z]" /c:"^[^A-Z]*ID" 2>NUL
        ECHO ============================================
        SET /P ID="Choose one of listed ID to remove "
        !TOOL_PATH!\%TOOL_1% -D pubkey --id %ID% 2>NUL
        GOTO MAIN_LOOP
:CASE_pair
        ECHO ========== [ LIST OF ALL KEYS] ==========
        (%TOOL_PATH%\%TOOL_2% --list-public-keys ^
         & %TOOL_PATH%\%TOOL_2% --list-keys) ^
        | findstr /r /c:"^[A-Z]" /c:"^[^A-Z]*ID" 2>NUL
        ECHO ========================================
        SET /P ID="Choose one of listed ID to remove "
        !TOOL_PATH!\%TOOL_1% -D pubkey,privkey --id %ID% 2>NUL
        GOTO MAIN_LOOP
:CASE_cert
        ECHO ========== [ LIST OF CERTIFIACTES] ==========
        (%TOOL_PATH%\%TOOL_2% --list-cert) ^
        | findstr /r /c:"^[A-Z]" /c:"^[^A-Z]*ID" 2>NUL
        ECHO ============================================
        SET /P ID="Choose one of listed ID to remove "
        !TOOL_PATH!\%TOOL_1% -D cert --id %ID% 2>NUL
        GOTO MAIN_LOOP
:CASE_all
        ECHO ========== [ LIST OF ALL OBJECTS ] ==========
        %TOOL_PATH%\%TOOL_2% --dump ^
        | findstr /r /c:"^[A-Z]" /c:"^[^A-Z]*ID" 2>NUL
        ECHO ============================================
        SET /P ID="Choose one of listed ID to remove "
        !TOOL_PATH!\%TOOL_1% -D cert,pubkey,privkey --id %ID% 2>NUL
        GOTO MAIN_LOOP
:CASE_exit
        SET /p EXIT=Hit ENTER to exit...
        VER > NUL rem reSET ERRORLEVEL
        GOTO :EOF
:DEFAULT_CASE
        ECHO Unknown option "%TYPE%"
        GOTO MAIN_LOOP
```

5.3.6 Generowanie kluczy

Kolejny skrypt powłoki odpowiada za wygenerowanie na karcie pary kluczy. Generowanie kluczy odbywa się przy pomocy polecenia *pkcs15-init –generate-key "rsa/2048" –auth-id "FF" –label %name%*. Wygenerowane klucze są zaszyfrowane przy użyciu asymetrycznego algorytmu kryptograficznego RSA 2048. Zanim para kluczy zostanie wygenerowana, użytkownik zostanie poinformowany o istniejących obiektach i ich nazwach. Nadanie odpowiedniej etykiety jest ważnym elementem generowania pary kluczy, ponieważ odpowiednia etykieta umożliwi łatwą identyfikację obiektów w przyszłości.

```
@ECHO off
SETLOCAL EnableDelayedExpansion
SET TOOL_1=pkcs15-init.exe
```

```
SET TOOL_2=pkcs15-tool.exe
SET TOOL_PATH="C:\Program Files (x86)\OpenSC Project\OpenSC\tools"
IF NOT EXIST %TOOL_PATH% (
    ECHO Error: OpenSC Tools not installed. Insert correct path to OpenSC tools:
    :SET_TOOLS
        SET /p TOOL_PATH=
        SET TOOL_PATH="!TOOL_PATH!"
        IF NOT EXIST !TOOL_PATH!\%TOOL_1% (
            ECHO Wrong path. Insert correct path to OpenSC tools:
            GOTO SET_TOOLS
        )
)
ECHO Current using objects names:
ECHO.
ECHO.
ECHO ========== [ LIST OF OBJECTS NAMES ] ==========
ECHO.
(%TOOL_PATH%\%TOOL_2% -k 2>NUL & %TOOL_PATH%\%TOOL_2% --list-public-keys 2>NUL ) ^
| findstr /r "^[a-z].*"
ECHO.
ECHO ================================================
ECHO.
:TRY_AGAIN
        ECHO Enter a properly name of pair. Choose name wisely it will help
        ECHO you to detect pair further:

        SET /p name=
        %TOOL_PATH%\%TOOL_1% --generate-key "rsa/2048" ^
        --auth-id "FF" --label %name% 2>NUL
        ECHO.
        IF /I "%ERRORLEVEL%" EQU "0" (
                ECHO Key pair %name% have been generated properly
        ) ELSE (
                ECHO Something went wrong.. Try again..
                GOTO :TRY_AGAIN
        )

SET /p EXIT=Hit ENTER to exit...
```

5.3.7 Ładowanie certyfikatu na kartę

Następny skrypt powłoki odpowiada za ładowanie certyfikatów na kartę. Realizacja tego zadania jest wykonywana za pomocą narzędzia *pkcs15-init*, narzędzie *pkcs15-tool* jest wykorzystywane do wylistowania certyfikatów. Po sprawdzeniu wymagań, konsola wydaje zapytanie o nazwę (lokalizację) certyfikatu. Jeżeli nazwa bądź ścieżka zawiera spację, należy użyć znaków cudzysłowu (co zaproponuje użytkownikowi również domyślnie wiersz poleceń systemu Windows, jeżeli użyje on podpowiedzi). Po uzyskaniu informacji o lokalizacji certyfikatu skrypt przystąpi do jego ładowania na kartę, w międzyczasie prosząc o uwierzytelnienie poprzez PIN do karty oraz hasło do certyfikatu (jeżeli jest wymagane). Na zakończenie zostanie wyświetlona aktualna lista certyfikatów.

```
@echo off
SETLOCAL EnableDelayedExpansion
SET TOOL_1=pkcs15-init.exe
SET TOOL_2=pkcs15-tool.exe
SET TOOL_PATH="C:\Program Files (x86)\OpenSC Project\OpenSC\tools"
IF NOT EXIST %TOOL_PATH% (
    echo Error: OpenSC Tools not installed. Insert correct path to OpenSC tools:
    :set_tools
        set /p TOOL_PATH=
```

```
        set TOOL_PATH="!TOOL_PATH!"
        IF NOT EXIST !TOOL_PATH!\%TOOL_1% (
            echo Wrong path. Insert correct path to OpenSC tools:
            goto set_tools
        )
)
:set_certificate
echo Enter name of certificate that you want to load:
SET /P certificate=
IF NOT EXIST %certificate% (
    echo File does not exist. Try again.
    goto set_certificate
)
%TOOL_PATH%\%TOOL_1% -S %certificate% --auth-id ff --format pkcs12
IF /I "%ERRORLEVEL%" EQU "0" (
    echo.
    echo.
    echo Certificate %name% have been loaded properly
) ELSE (
    echo.
    echo.
    echo Something went wrong.. Try again..
    echo.
    goto :set_certificate
)
echo.
echo.
echo ========== [ LIST OF STORED CERTIFICATES ] ==========
echo.
%TOOL_PATH%\%TOOL_2% -D 2>NUL
echo ====================================================
echo.
set /p EXIT=Hit ENTER to exit...
```

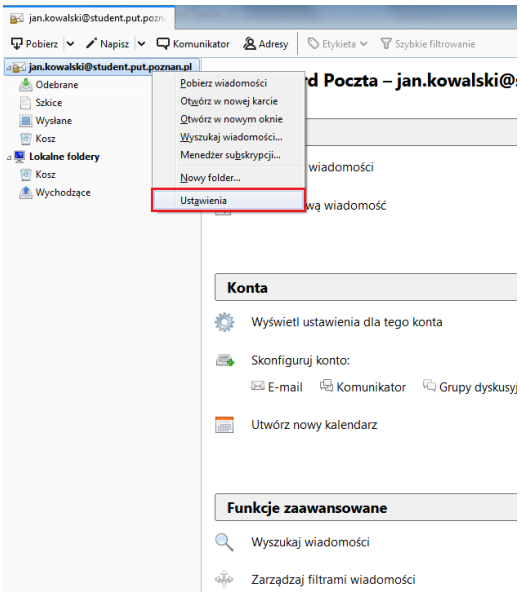
5.4 Biblioteka DLL dla aplikacji zewnętrznych

Biblioteka DLL pochodzi z pakietu instalacyjnego biblioteki OpenSC, stanowiącej główny filar oprogramowania middleware. Biblioteka została przygotowana w wersji 32 oraz 64-bitowej. Zastosowanie odpowiedniej wersji jest zależne od wersji docelowej aplikacji zewnętrznej. Biblioteka DLL jest wymagana przez aplikację *SmartCard Suite* oraz aplikacje zewnętrzne, korzystające z wykorzystanego apletu PKI. Plik biblioteki DLL implementującej PKCS#11, dostarczanej z pakietem OpenSC nazwano *opensc-pkcs11.dll*. W systemie Windows, przy domyślnej instalacji znajduje się on w katalogu *C:\Program Files (x86)\OpenSC Project\OpenSC\pkcs11* dla wersji 32 bitowej, a dla wersji 64 bitowej w katalogu *C:\Program Files\OpenSC Project\OpenSC\pkcs11*.

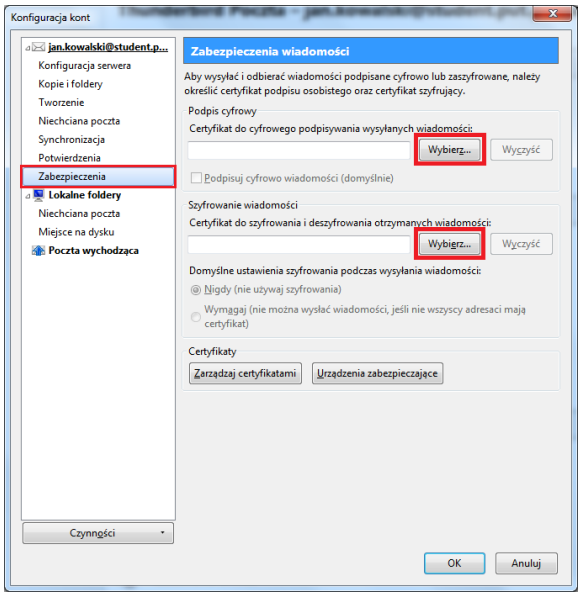
5.4.1 Program pocztowy na przykładzie Mozilla Thunderbird

Poniższy opis obejmuje przykładowe użycie biblioteki wraz z kartą elektroniczną, na której znajduje się zainstalowany aplet PKI. Instrukcja została przygotowana dla 32-bitowej wersji aplikacji Mozilla Thunderbird w wersji *60.4.0* i obejmuje wczytanie biblioteki, zalogowanie karty w aplikacji oraz podpisanie wiadomości email certyfikatem zainstalowanym na karcie.

Do skorzystania z certyfikatów zawartych na karcie konieczne jest **wczytanie biblioteki DLL**. Aby tego dokonać, użytkownik powinien kliknąć prawym przyciskiem myszy na nazwie swojego konta pocztowego, wybrać z menu podręcznego pozycje *Ustawienia* (rysunek 5.1). Po tej czynności uzyskany zostanie dostęp do okna *Konfiguracja kont*. Z menu podręcznego, znajdującego się po lewej stronie, należy wybrać pozycję *Zabezpieczenia*, będącą w grupie naszego konta pocztowego (rysunek 5.2).

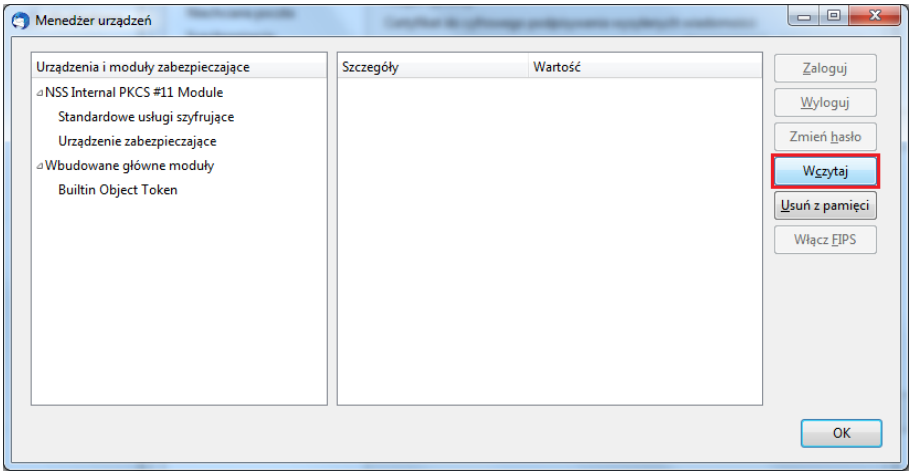


Rysunek 5.1: Mozilla Thinderbird - ustawienia konta



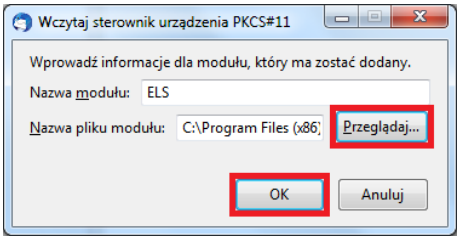
Rysunek 5.2: Mozilla Thunderbird - zabezpieczenia wiadomości

Następnie należy wybrać opcję *Urządzenia zabezpieczające*, po czym pojawi się okno *Menedżer urządzeń*. W nowo otwartym oknie trzeba wybrać pozycję *Wczytaj* (rysunek 5.3).



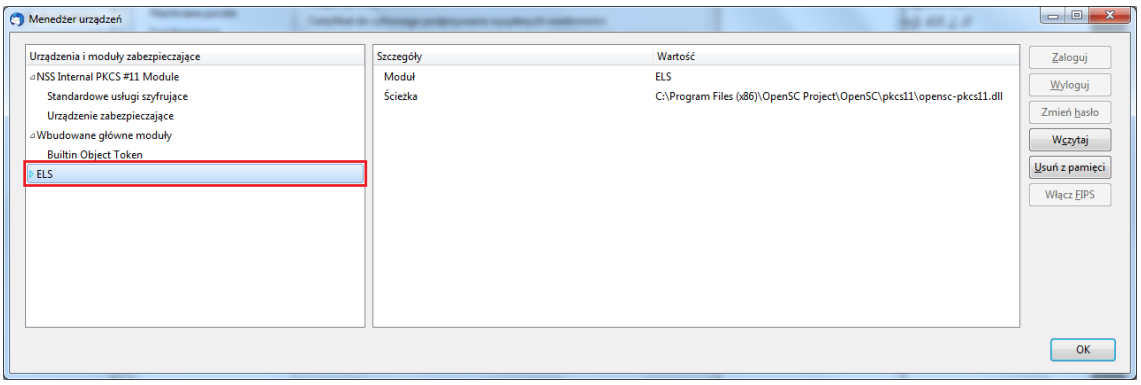
Rysunek 5.3: Mozilla Thunderbird - menedżer urządzeń

W kolejnym kroku użytkownik zostanie poproszony o wprowadzenie nazwy modułu oraz wskazanie ścieżki do biblioteki DLL poprzez przycisk *Przeglądaj*. Domyślna ścieżka została wskazana w podrozdziale 5.4.1. Po wskazaniu lokalizacji, zatwierdzamy nowy sterownik przyciskiem *OK* (rysunek 5.4).



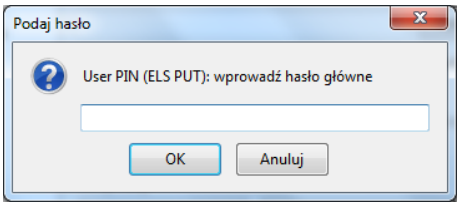
Rysunek 5.4: Mozilla Thunderbird - wczytanie sterownika PKCS#11

Jeżeli operacja się powiodła, w zakładce *Urządzenia i moduły zabezpieczające* powinna pojawić się nowa pozycja. Należy zamknąć okno *Menedżera urządzeń* (rysunek 5.5).



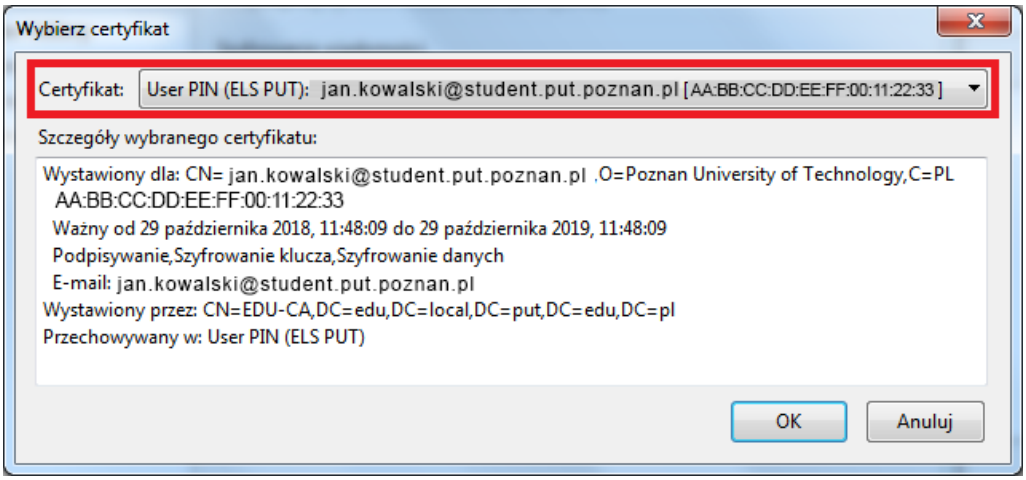
Rysunek 5.5: Mozilla Thunderbird - menadżer urządzeń po dodaniu sterownika

Po poprawnym wczytaniu biblioteki należy umieścić kartę w czytniku. Znajdując się w oknie *Konfiguracja kont*, w zakładce *Zabezpieczenia wiadomości*, należy kliknąć przycisk *Wybierz* w sekcji *Podpis cyfrowy* (rysunek 5.2). Aplikacja dokona próby uwierzytelnienia poprzez kod PIN do karty (rysunek 5.6), a po poprawnym zakończeniu tego procesu zostanie wyświetlone okno wyboru certyfikatu.



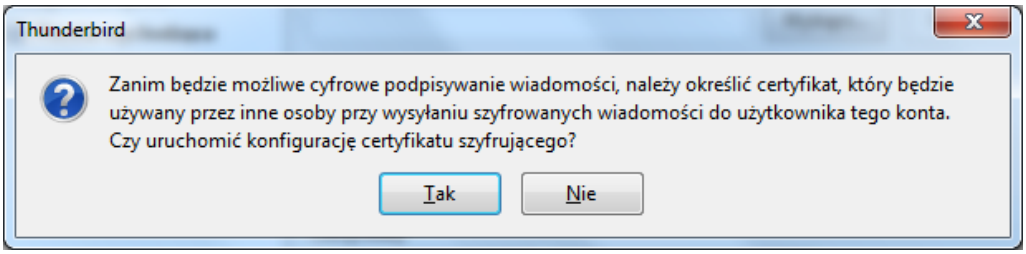
Rysunek 5.6: Mozilla Thunderbird - okno uwierzytelniania

Domyślnie zostanie wskazany jeden z certyfikatów z karty. Korzystając z opcji *Wybierz certyfikat*, można dokonać wyboru innego certyfikatu (rysunek 5.7).



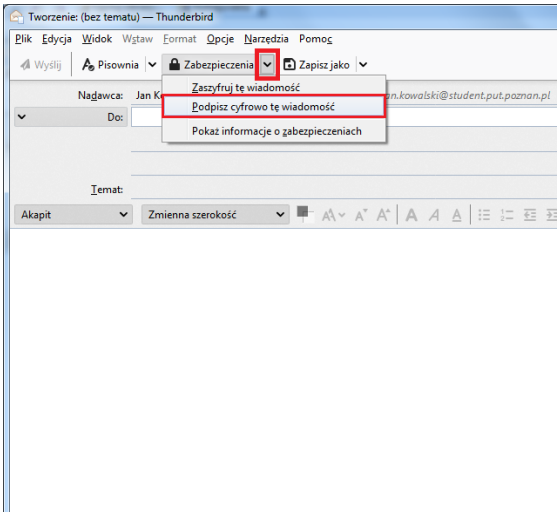
Rysunek 5.7: Mozilla Thunderbird - okno wyboru certyfikatu

Po wybraniu poprawnego certyfikatu, należy zatwierdzić wybór przyciskiem *OK*. Po zatwierdzeniu, aplikacja pocztowa zapyta o możliwość użycia tego certyfikatu również przy procesie szyfrowania wiadomości (rysunek 5.8). W przypadku, gdy użytkownik nie posiada innego certyfikatu, powinien wybrać odpowiedź twierdzącą. W przeciwnym wypadku należy odrzucić propozycję aplikacji i powtórzyć procedurę dla sekcji *Szyfrowanie wiadomości*.

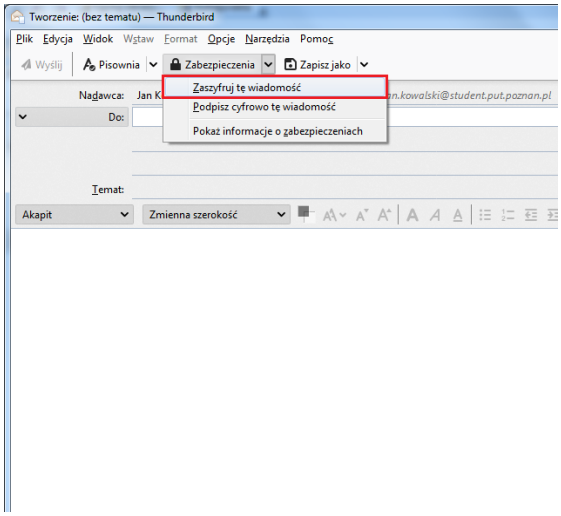


Rysunek 5.8: Mozilla Thunderbird - uruchamianie konfiguracji certyfikatu szyfrującego

Aby **podpisać wiadomość** kluczem prywatnym, podczas tworzenia nowej wiadomości email, należy kliknąć w „strzałkę skierowaną w dół" obok pozycji *Zabezpieczenia*, a następnie wskazać opcję *Podpisz cyfrowo tę wiadomość* (rysunek 5.9).



Rysunek 5.9: Mozilla Thunderbird - podpisanie wiadomości email



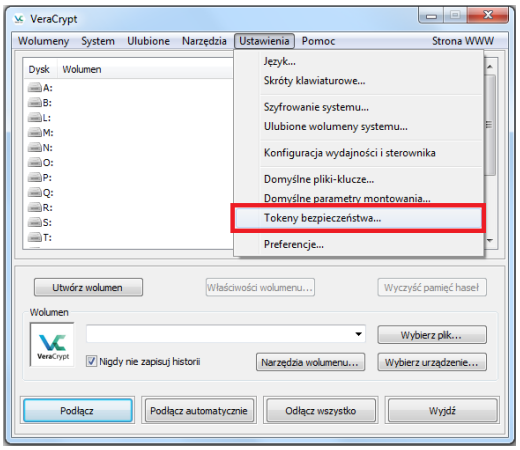
Rysunek 5.10: Mozilla Thunderbird - szyfrowanie wiadomości email

Natomiast aby **zaszyfrować wiadomość** kluczem publicznym odbiorcy, podczas tworzenia nowej wiadomości email, należy również kliknąć w „strzałkę skierowaną w dół" obok pozycji *Zabezpieczenia*, a następnie wskazać opcję *Zaszyfruj tę wiadomość* (rysunek 5.10).

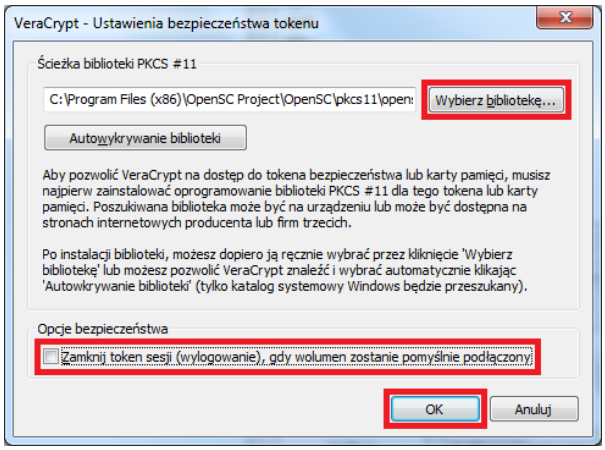
5.4.2 Program do szyfrowania danych na przykładzie aplikacji VeraCrypt

W tym rozdziale przedstawione będzie przykładowe użycie biblioteki wraz z kartą elektroniczną, na której znajduje się zainstalowany aplet PKI. Instrukcja została przygotowana dla 32-bitowej wersji aplikacji VeraCrypt w wersji *1.23-Hotfix-2* i obejmuje załadowanie biblioteki DLL, załadowanie pliku-klucza na kartę, zaszyfrowanie oraz odszyfrowanie magazynu plików przy użyciu wspomnianego pliku-klucza.

Załadowanie biblioteki DLL do programu jest możliwe poprzez wybranie z paska menu pozycji *Ustawienia*, a następnie pozycji *Tokeny bezpieczeństwa* (rysunek 5.11). Następnie należy wskazać ścieżkę do biblioteki oraz zatwierdzić wybór przyciskiem *OK* (rysunek 5.12).

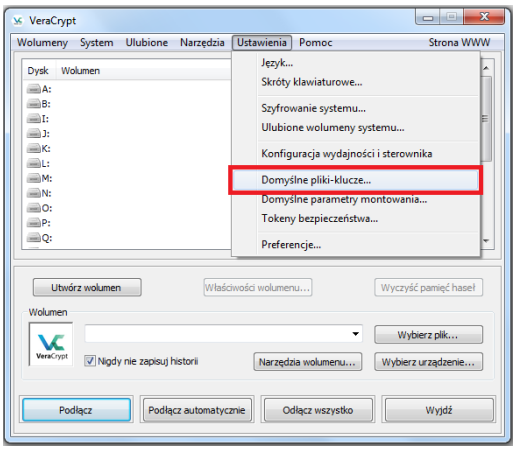


Rysunek 5.11: VeraCrypt - okno główne, wybór tokenów bezpieczeństwa

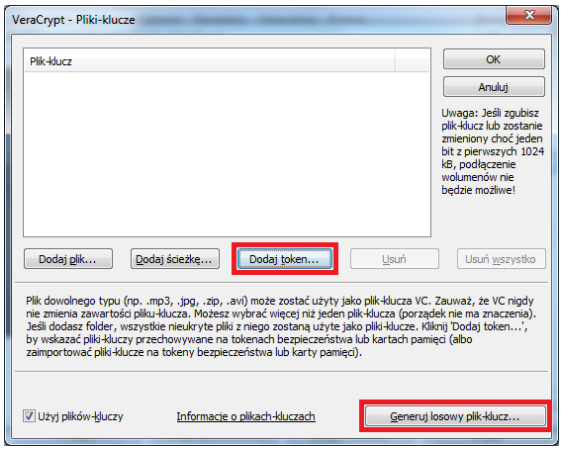


Rysunek 5.12: VeraCrypt - okno wyboru biblioteki DLL dla apletu

Kolejny krok przedstawia **załadowanie pliku-klucza na kartę**. Jako plik-klucz użytkownik może wybrać własny plik lub skorzystać z narzędzia do generowania losowego pliku-klucza. Załadowanie pliku-klucza na kartę jest możliwe poprzez narzędzie *Domyślne pliki-klucze*, z pozycji Ustawienia (rysunek 5.13). Aby przejść dalej użytkownik, powinien posiadać własny plik-klucz. W przypadku korzystania z pliku losowego, należy wybrać przycisk *Generuj losowy plik-klucz* w oknie *Pliki-klucze* (rysunek 5.14).

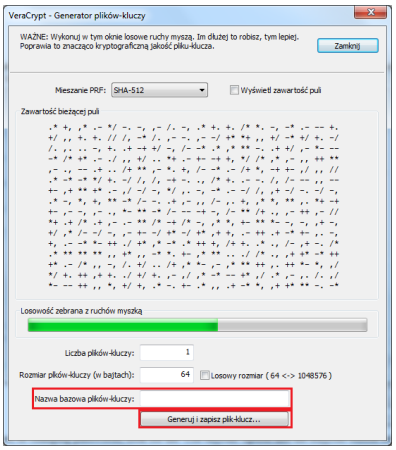


Rysunek 5.13: VeraCrypt - okno główne, dodawanie pliku-klucza

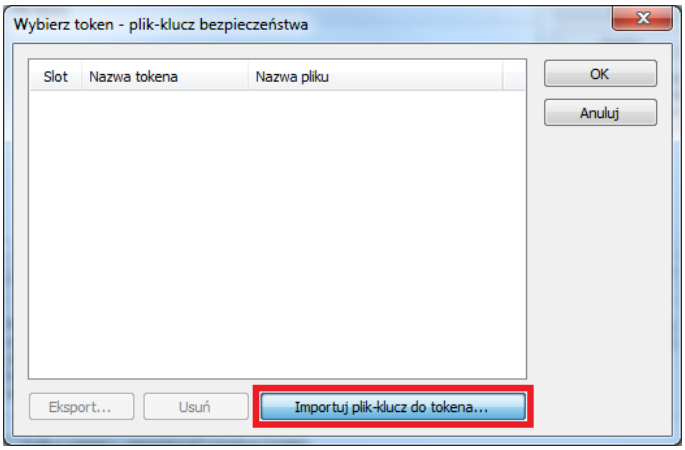


Rysunek 5.14: VeraCrypt - okno wyboru pliku-klucza wraz z dodatkowymi opcjami

W nowo otwartym oknie *Generator plików-kluczy* należy podać nazwę bazową, a następnie w celu uzyskania właściwej entropii - wykonywać ruchy myszą do momentu załadowania paska postępu. Kiedy pasek zostanie w pełni załadowany, należy wybrać przycisk *Generuj i zapisz plik-klucz* (rysunek 5.15). W dalszej części, w oknie *Pliki-klucze* należy wybrać przycisk *Dodaj token* (rysunek 5.14), po czym konieczne jest uwierzytelnienie numerem PIN karty. Po poprawnym procesie uwierzytelnienia pojawi się okno *Wybierz token - plik-klucz bezpieczeństwa*, w którym należy wybrać przycisk *Importuj plik-klucz do tokena* (rysunek 5.16) i wybrać z dysku plik-klucz (własny bądź wcześniej wygenerowany losowo).

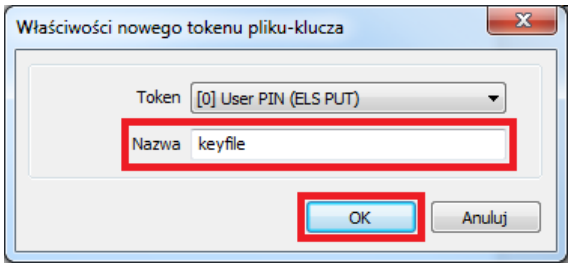


Rysunek 5.15: VeraCrypt - okno generatora pliku-klucza

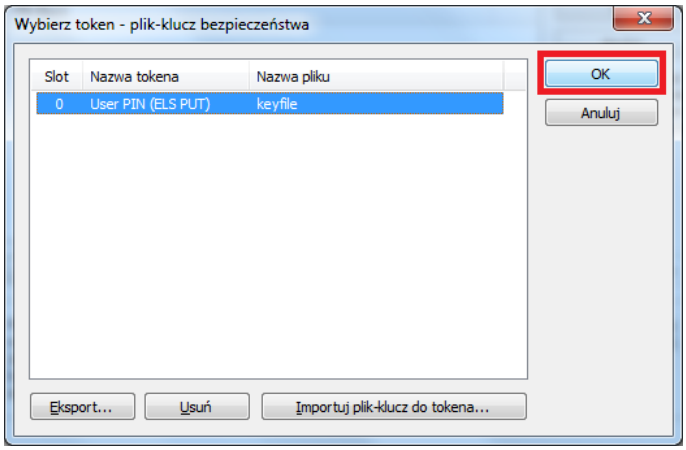


Rysunek 5.16: VeraCrypt - okno wyboru pliku-klucza dla tokenu (karty)

Po wybraniu pliku należy wprowadzić nazwę dla pliku-klucza i zatwierdzić operację poprzez przycisk *OK* (rysunek 5.17), w następnym oknie również przycisk *OK* (rysunek 5.18).

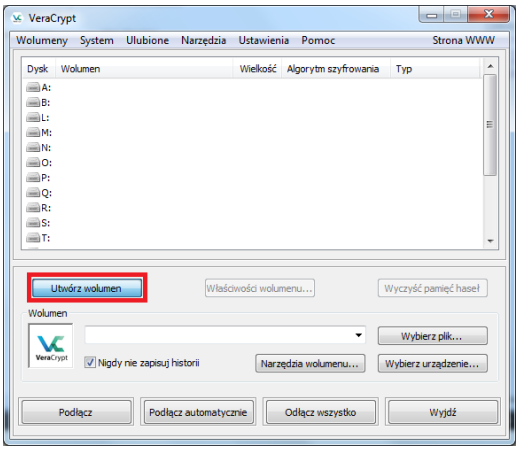


Rysunek 5.17: VeraCrypt - okno właściwości nowego tokenu

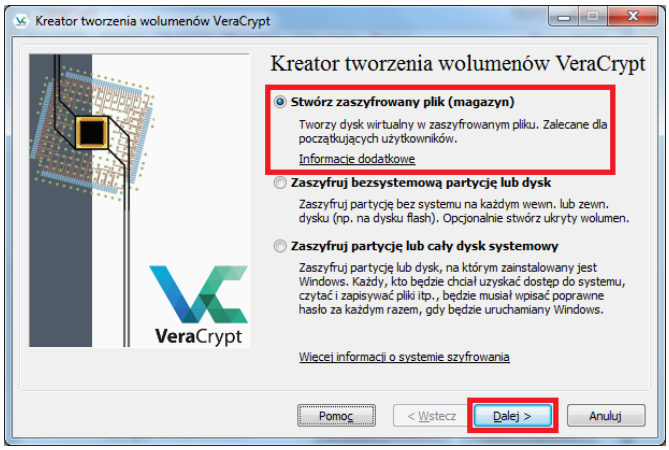


Rysunek 5.18: VeraCrypt - okno wyboru pliku-klucza z załadowanym już plikiem-kluczem

Za pomocą programu VeraCrypt można dokonać **szyfrowania przy użyciu pliku-klucza z karty**. Zaszyfrować można magazyn plików, bez-systemową lub systemową partycję/dysk. Jako przykład posłużyło szyfrowanie magazynu plików.
Z głównego okna należy wybrać przycisk *Utwórz wolumen* (rysunek 5.19). Następnie zostanie uruchomiony kreator, w którym w pierwszym kroku należy wybrać *Stwórz zaszyfrowany plik (magazyn)* (rysunek 5.20).

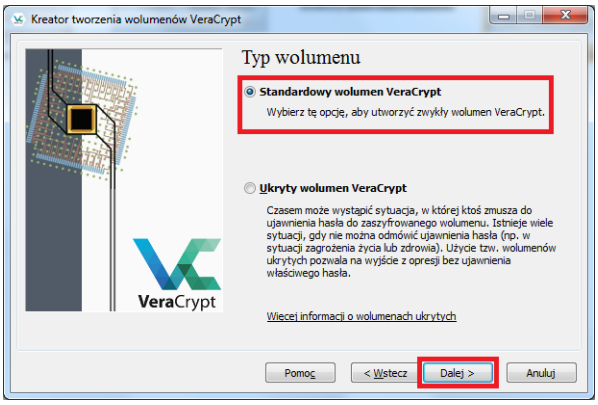


Rysunek 5.19: VeraCrypt - okno główne, tworzenie wolumenu

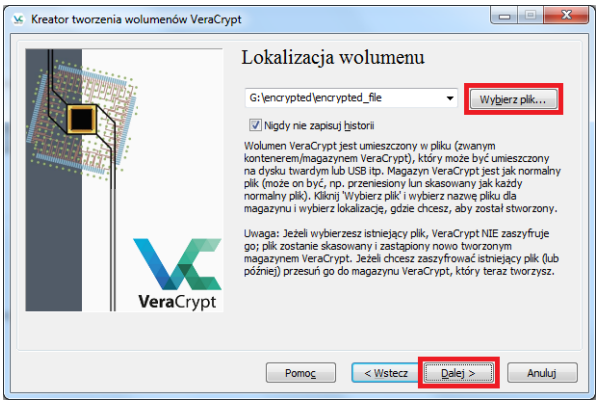


Rysunek 5.20: VeraCrypt - kreator tworzenia wolumenu etap 1

W kolejnym kroku należy wskazać *Standardowy wolumen VeraCrypt* (rysunek 5.21). Dalej należy wskazać lokalizację pliku z magazynem (rysunek 5.22).

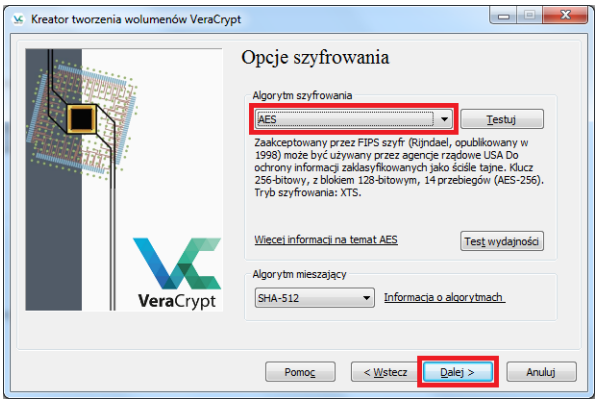


Rysunek 5.21: VeraCrypt - kreator tworzenia wolumenu etap 2

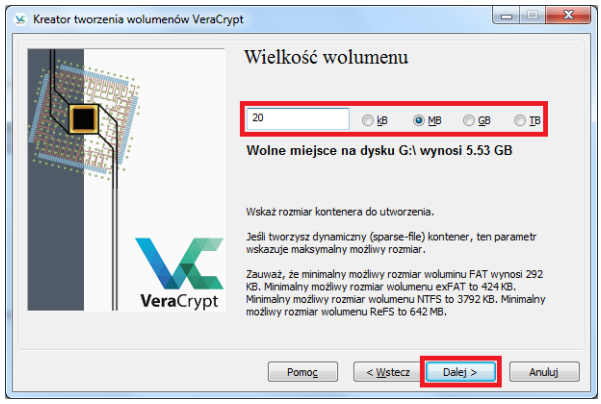


Rysunek 5.22: VeraCrypt - kreator tworzenia wolumenu etap 3

W następnym kroku należy wybrać algorytm szyfrowania, domyślnie wybrany jest algorytm *AES* (rysunek 5.23), jednakże dla jak największego bezpieczeństwa wskazane jest wybranie szyfrowania kaskadowego, będącego połączeniem kilku szyfrów (dane szyfrowane są jednym algorytmem, a następnie kolejnym, tym samym jeżeli wystąpi luka w jednym z nich, napastnik będzie miał do pokonania kolejne szyfry). W tym wypadku zalecane jest wybranie pozycji *AES(Twofish(Serpent))*. Dalej należy wybrać wielkość wolumenu (rysunek 5.24).

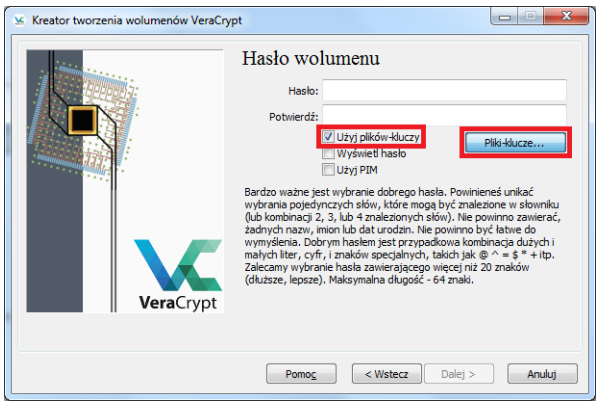


Rysunek 5.23: VeraCrypt - kreator tworzenia wolumenu etap 4

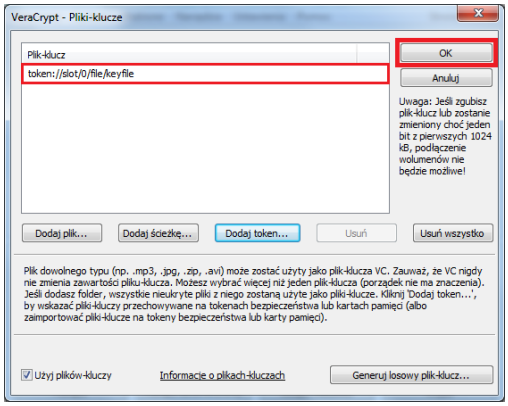


Rysunek 5.24: VeraCrypt - kreator tworzenia wolumenu etap 5

W następnym kroku konieczne jest zaznaczenie opcji *Użyj plików-kluczy* oraz wybranie przycisku *Pliki-klucze* w celu wskazania pliku-klucza (rysunek 5.25) (konieczne może okazać się ponowne uwierzytelnienie kodem PIN karty). Po wybraniu pliku-klucza należy zatwierdzić wybór w pierwszym (rysunek 5.18) oraz drugim oknie (rysunek 5.26).

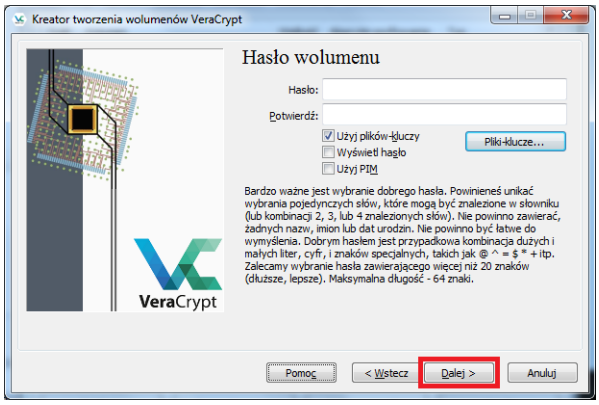


Rysunek 5.25: VeraCrypt - kreator tworzenia wolumenu etap 6

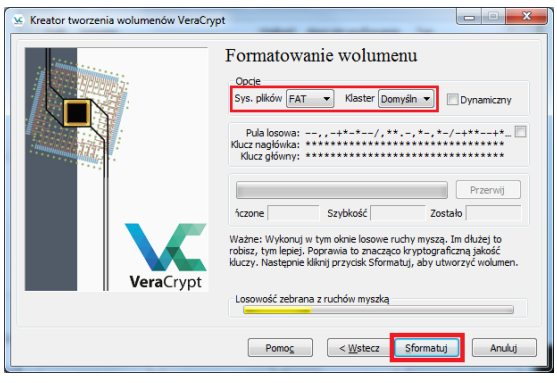


Rysunek 5.26: VeraCrypt - kreator tworzenia wolumenu wybór pliku-klucza

Następnie użytkownik może przejść dalej (rysunek 5.27). W kolejnym etapie do wyboru jest system plików tworzonego magazynu. Podczas tego etapu program ponownie wykorzystuje losowe ruchy myszy do polepszenia kryptograficznej jakości kluczy. Kiedy pasek postępu dotrze do końca, należy wybrać przycisk *Sformatuj* (rysunek 5.28).

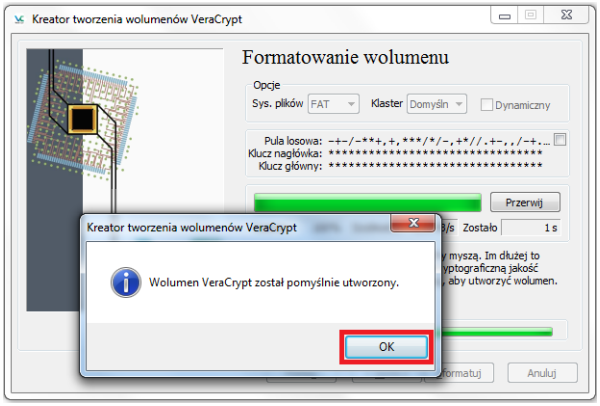


Rysunek 5.27: VeraCrypt - kreator tworzenia wolumenu - przejście do następnego etapu

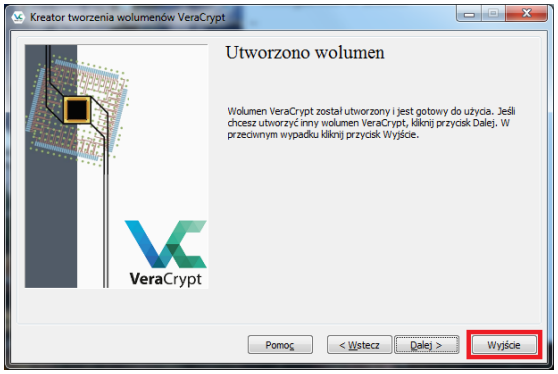


Rysunek 5.28: VeraCrypt - kreator tworzenia wolumenu etap 7

Po zakończonym procesie tworzenia magazynu zostanie wyświetlone okno potwierdzające pozytywne zakończenie operacji (rysunek 5.29). Jeżeli użytkownik nie będzie tworzył kolejnego magazynu, po potwierdzeniu komunikatu powinien w następnym kroku wybrać przycisk *Wyjście* (rysunek 5.30).

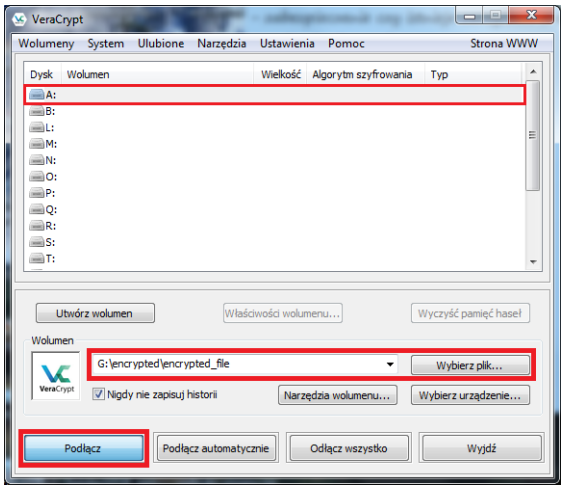


Rysunek 5.29: VeraCrypt - kreator tworzenia wolumenu zakończenie procesu

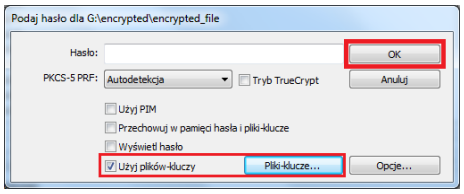


Rysunek 5.30: VeraCrypt - kreator tworzenia wolumenu wyście z kreatora

W przeciwieństwie do szyfrowania, **proces odszyfrowania magazynu pliku przy użyciu pliku-klucza z karty** jest krótki i prosty. Użytkownik powinien wybrać literę, do której zostanie zamapowany magazyn, wybrać plik magazynu oraz wybrać przycisk *Podłącz* (rysunek 5.31). W kolejnym, oknie analogicznie jak przy procesie szyfrowania, wymagane jest wskazanie pliku-klucza (rysunek 5.32).

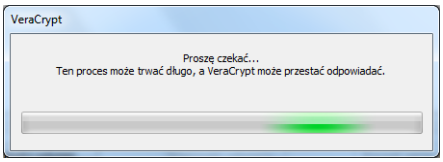


Rysunek 5.31: VeraCrypt - podłączenie wolumenu



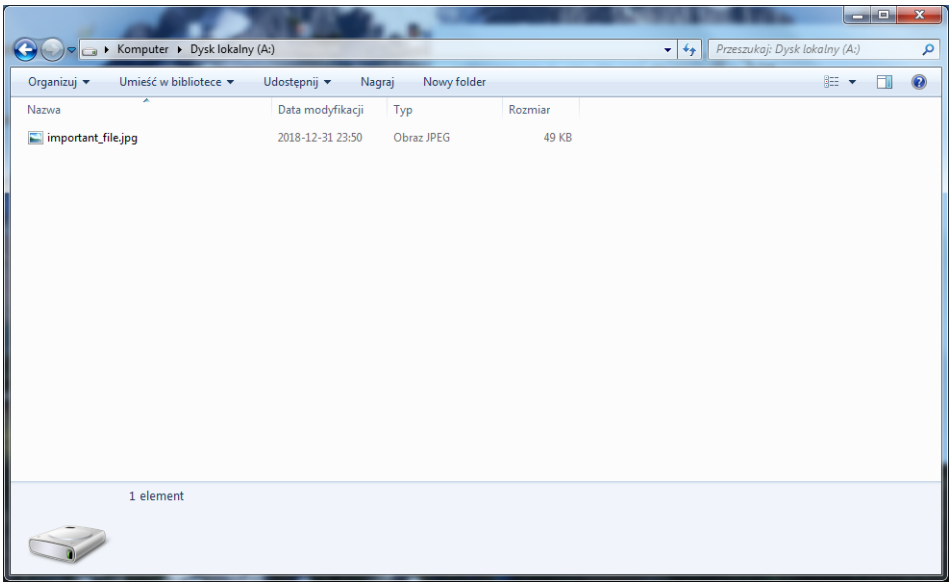
Rysunek 5.32: VeraCrypt - okno uwierzytelnienia podłączenia

Program rozpocznie procedurę odszyfrowywania (rysunek 5.33).

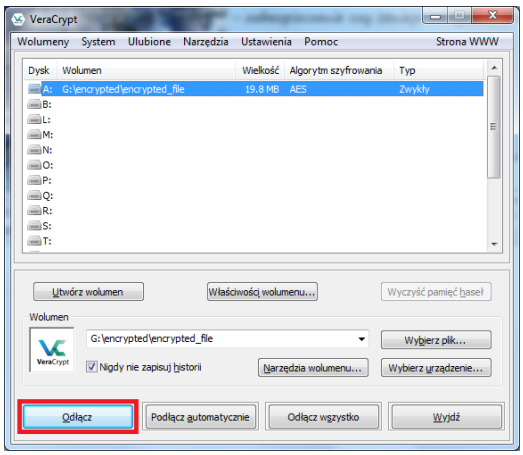


Rysunek 5.33: VeraCrypt - proces odszyfrowywania

Po jego zakończeniu magazyn zostanie zamapowany jak klasyczna partycja (rysunek 5.34). W tym momencie można na niej rozpocząć umieszczanie plików. Po zakończonej pracy należy odłączyć wolumen, co użytkownik może uczynić wybierając przycisk *Odłącz* (rysunek 5.35).



Rysunek 5.34: Zamapowany magazyn plików



Rysunek 5.35: VeraCrypt - odłączanie magazynu

5.4.3 Programy korzystające z zasobnika certyfikatów systemu operacyjnego

System Windows korzysta z biblioteki Cryptographic Service Provider do implementacji Microsoft CryptoAPI (interfejsu programowania aplikacji wbudowanego w systemy Microsoft Windows dla usług kryptografii). Aby karta była widziana przez bibliotekę CSP, należy wykonać kilka zmian w kodzie biblioteki OpenSC. Pierwszym krokiem jest poszerzenie implementacji funkcji *isoApplet_match_card(sc_card_t card)*, znajdującej się w pliku *src/libopensc/card-isoApplet.c* o następujące linie:

```
int i;
    SC_FUNC_CALLED(card->ctx, SC_LOG_DEBUG_VERBOSE);

    i = _sc_match_atr(card, isoApplet_atrs, &card->type);
    if (i < 0)
            return 0;
```

Kolejnym krokiem jest dodanie definicji kart, które mają być obsługiwane. Tutaj konieczne są zmiany w trzech plikach. Pierwszą ponownie realizujemy w pliku *src/libopensc/card-isoApplet.c*. Po sekcji zawierającej definicje należy zadeklarować strukturę typu *sc_atr_table* zawierającą wzorce oraz maski komunikatów ATR kart, które mają być wykrywane przez system. Poniżej zostały zamieszczone przykłady dla kart, dla których była realizowana praca. W przedstawionym przykładzie maski ATR nie zostały zdefiniowane ze względu na korzystanie z pojedynczych kart.

```
static const struct sc_atr_table isoApplet_atrs[] = {
        /* JCOP cards */
        { "3b:f8:13:00:00:81:31:fe:45:4a:43:4f:50:32:30:31:38:fd" \
        , NULL, "JCOP", SC_CARD_TYPE_ISOAPPLET_JCOP, 0, NULL },
        /* Oberthut Cosmo v7 IAS ECC cards */
        { "3b:dd:18:00:81:31:fe:45:80:f9:a0:00:00:00:77:01:08:00:07:90:00:fe", \
        NULL, "Oberthut Cosmo v7 IAS ECC", \
        SC_CARD_TYPE_ISOAPPLET_OBERTHUR_COSMO_7_1, 0, NULL }
};
```

Następnie należy umieścić zdefiniowane typy w pliku *src/libopensc/cards.h*, w sekcji przewidzianej dla sterownika apletu *isoApplet* (sekcja rozpoczynająca się komentarzem *JavaCards with isoApplet*). Dla zdefiniowanych wyżej przykładów konieczne są deklaracje:

```
    /* JavaCards with isoApplet */
        SC_CARD_TYPE_ISO_APPLET_BASE=28000, // wpis istniejacy
        SC_CARD_TYPE_ISO_APPLET_GENERIC, // wpis istniejacy
        SC_CARD_TYPE_ISOAPPLET_JCOP, // wpis konieczny do dodania
        SC_CARD_TYPE_ISOAPPLET_OBERTHUR_COSMO_7_1, // wpis konieczny do dodania
```

Ostatnią zmianę należy wykonać w pliku *win32/customactions.cpp*. W strukturze *minidriver_registration* konieczne jest umieszczenie nowego wpisu typu *MD_REGISTRATION*, zawierającego treść komunikatu ATR karty, maskę ATR oraz nazwę karty widzianą przez system Windows. Dla zdefiniowanych wyżej przykładów konieczne są deklaracje:

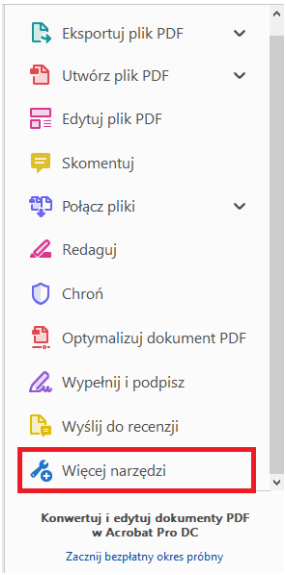
```
/* from card-isoApplet.c */
        {TEXT("JCOP isoApplet"), \
        {0x3B,0xF8,0x13,0x00,0x00,0x81,0x31,0xfe,0x45,0x4A,0x43,0x4F,0x50,0x32, \
        ,0x31,0x38,0xFD},
    18, {0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff, \
    0xff,0xff,0xff,0xff,0xff}},

        {TEXT("Oberthur Cosmo v7"), \
        {0x3B,0xDD,0x18,0x00,0x81,0x31,0xFE,0x45,0x80,0xF9,0xA0,0x00,0x00,0x00, \
        0x77,0x01,0x08,0x00,0x07,0x90,0x00,0xFE}, \
        22, {0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff, \
        0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff}},
```

Po dokonaniu zmian należy przebudować źródła, celem uzyskania nowych plików instalacyjnych. Do tego zadania najlepszym narzędziem jest *AppVeyor* [44], będący internetowym narzędziem CI (Continuous integration) dla systemów Microsoft Windows. Narzędzie jest bezpłatne dla projektów Open Source. Konfiguracja narzędzia sprowadza się jedynie do umieszczenia kodu aplikacji na jednym z popularnych systemów kontroli wersji i dokonaniem integracji między nim a kontem AppVeyor.

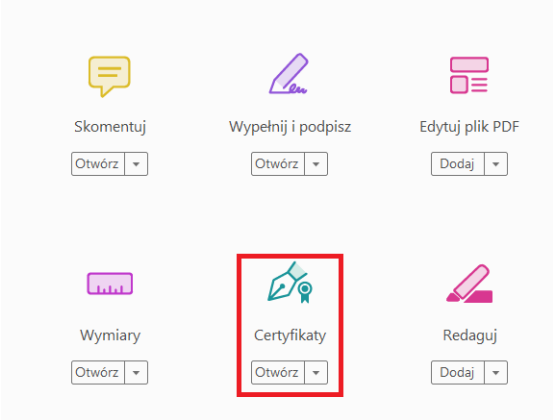
Adobe Reader

Poniższy opis obejmuje przykładowe użycie zasobnika certyfikatów systemu operacyjnego wraz z kartą elektroniczną, na której znajduje się zainstalowany aplet PKI oraz certyfikat użytkownika. Instrukcja została przygotowana dla programu Adobe Reader DC w wersji 2019.010.20069 i przedstawia podpisanie dokumentu w formacie *pdf*.
W pierwszym kroku należy wybrać z przybornika, znajdującego się z prawej strony okna głównego opcję *Więcej narzędzi* (rysunek 5.36)



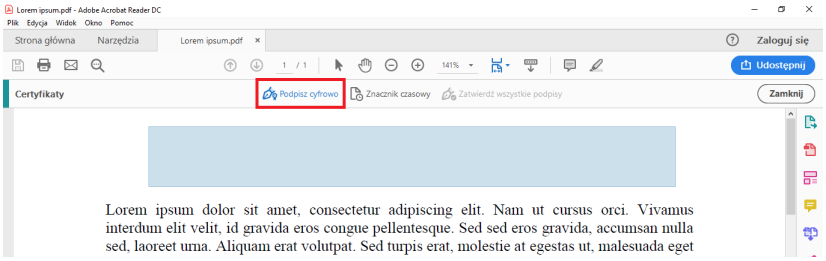
Rysunek 5.36: Adobe Reader DC - wybór opcji *Więcej narzędzi*

W następnym oknie użytkownik musi wybrać opcję *Certyfikaty* (rysunek 5.37).



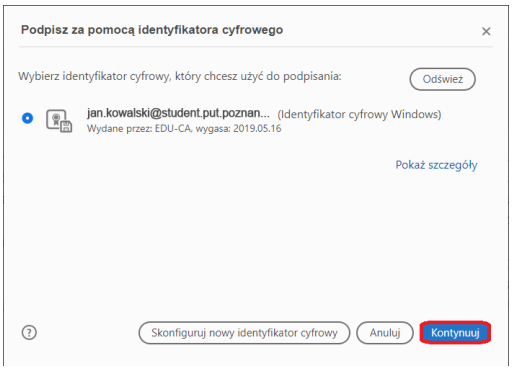
Rysunek 5.37: Adobe Reader DC - wybór opcji *Certyfikaty*

Następuje powrót do okna głównego, gdzie użytkownik musi wybrać opcję *Podpisz cyfrowo* (pasek *Certyfikaty*) i zaznaczyć obszar, w którym chce utworzyć podpis cyfrowy (rysunek 5.38).

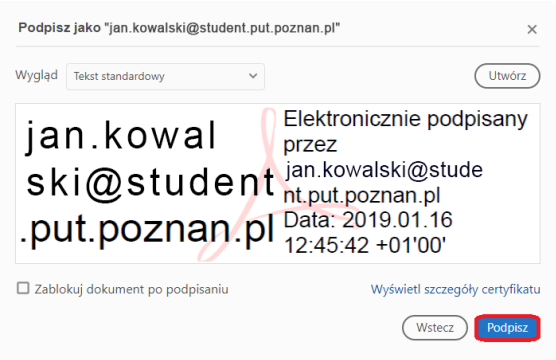


Rysunek 5.38: Adobe Reader DC - zaznaczenie miejsca, w którym zostanie utworzony podpis cyfrowy

W następnym kroku użytkownik wybiera odpowiedni certyfikat i zatwierdza wybór przyciskiem *Kontynuuj*(rysunek 5.39). Pojawi się kolejne okno, w którym użytkownik zobowiązany jest sprawdzić czy dane podpisu są odpowiednie, jeżeli tak należy nacisnąć przycisk *Podpisz*(rysunek 5.40).

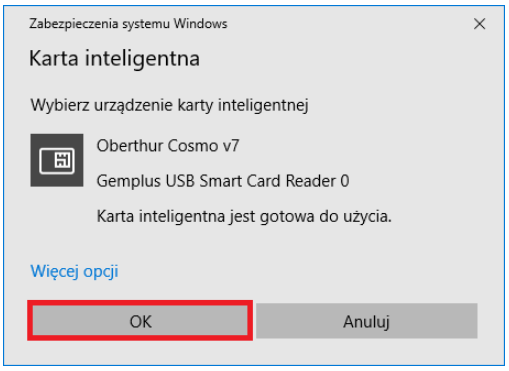


Rysunek 5.39: Adobe Reader DC - wybór certyfikatu

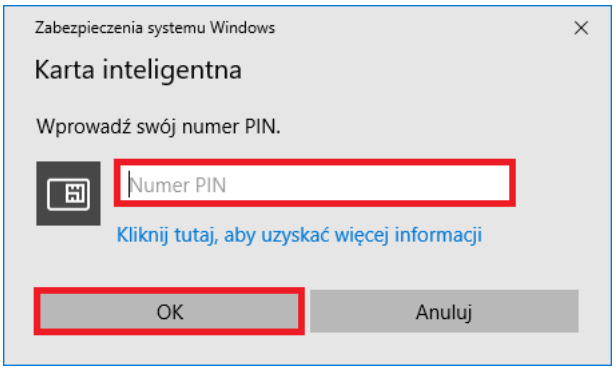


Rysunek 5.40: Adobe Reader DC - podpisywanie dokumentu wybranym certyfikatem

Następnie w oknie *Zabezpieczenia systemu* należy sprawdzić, czy został wybrany odpowiedni czytnik kart elektronicznych, jeżeli tak, użytkownik zatwierdza wybór przyciskiem *OK* (rysunek 5.41). Następnie, aby dokument został podpisany użytkownik, musi podać odpowiedni kod PIN oraz zatwierdzić operację przyciskiem *OK* (rysunek 5.42).

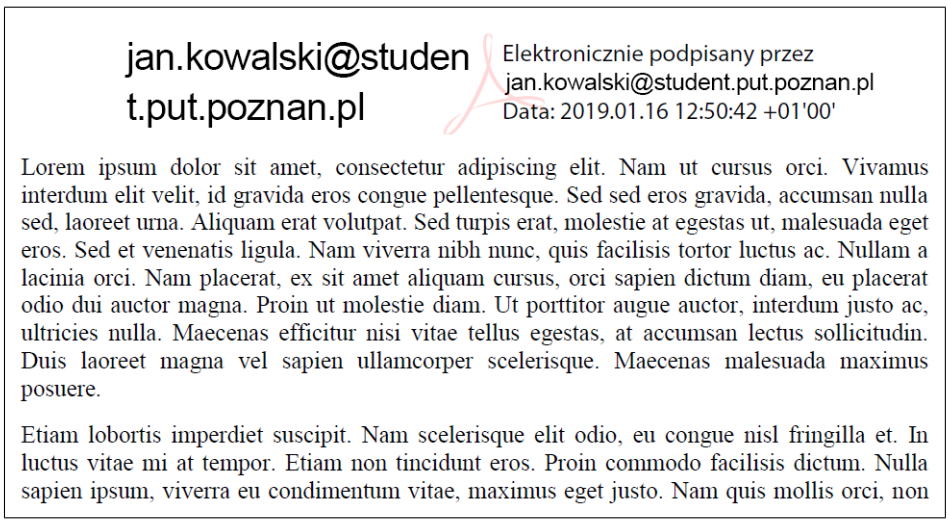


Rysunek 5.41: Adobe Reader DC - zatwierdzenie wyboru odpowiedniej karty elektronicznej



Rysunek 5.42: Adobe - podanie kodu PIN do karty elektronicznej

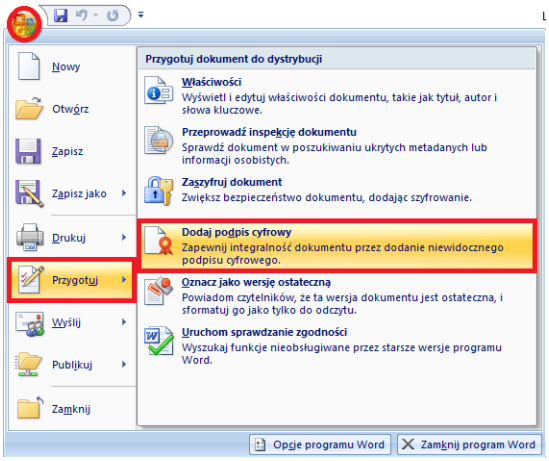
W wyniku wykonania powyższych kroków, we wcześniej zaznaczonym polu, powinien pokazać się odpowiedni podpis cyfrowy (rysunek 5.43).



Rysunek 5.43: Adobe Reader DC - dokument wraz z podpisem

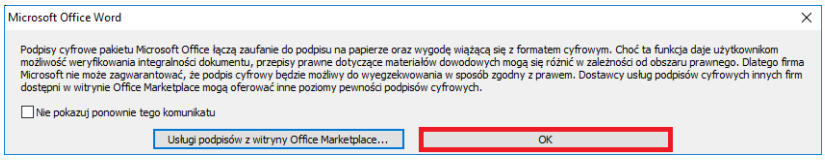
Microsoft Word

Kolejny przykład dotyczy użycia zasobnika certyfikatów systemu operacyjnego wraz z kartą elektroniczną, na której znajduje się zainstalowany aplet PKI oraz certyfikat użytkownika. Instrukcja została przygotowana dla programu Microsoft Word 2007 i przedstawia podpisanie dokumentu w formacie *docx*.
Zanim plik zostanie podpisany, konieczny jest jego zapis. Następnie należy nacisnąć *Przycisk pakietu Office* i wybrać opcję *Przygotuj* oraz *Dodaj podpis elektroniczny* (rysunek 5.44).



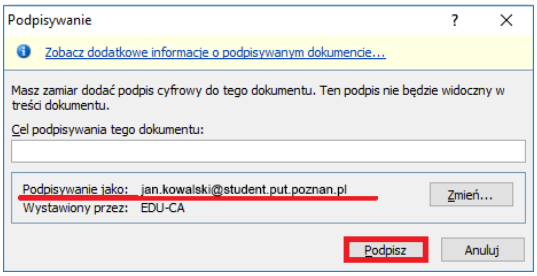
Rysunek 5.44: Microsoft Word - Wybieranie opcji *Dodaj podpis cyfrowy*

Następnie użytkownik zostaje poinformowany przez firmę Microsoft, że nie zapewnia ona mocy prawnej podpisu elektronicznego. Aby przejść do następnego kroku, należy nacisnąć przycisk *OK* (rysunek 5.45).

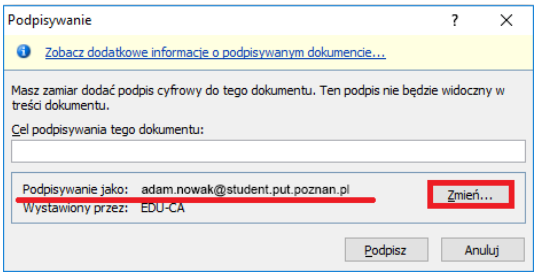


Rysunek 5.45: Microsoft Word - informacja od firmy Microsoft dotycząca mocy prawnej podpisu cyfrowego

W następnym kroku użytkownik powinien sprawdzić, czy w polu *Podpisywanie jako* znajdują się odpowiednie dane, jeżeli tak, należy wybrać opcję *Podpisz* (rysunek 5.46). W przeciwnym wypadku, użytkownik jest zmuszony wybrać opcję *Zmień...* (rysunek 5.47).

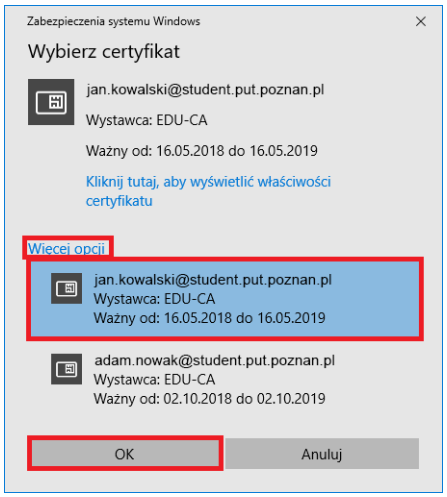


Rysunek 5.46: Microsoft Word - okno w przypadku gdy dane są odpowiednie



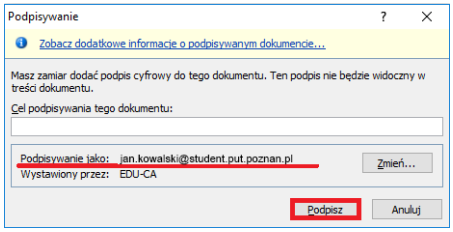
Rysunek 5.47: Microsoft Word - okno w przypadku gdy dane **nie** są odpowiednie

Następnie w oknie *Zabezpieczenia systemu Windows* należy nacisnąć *Więcej opcji*, wybrać odpowiedni certyfikat i zatwierdzić przyciskiem *OK* (rysunek 5.48).



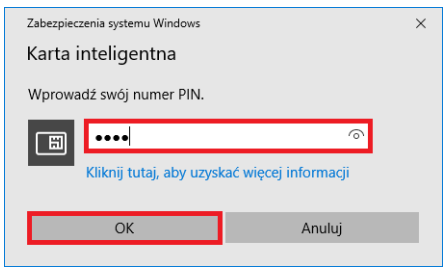
Rysunek 5.48: Microsoft Word - wybór odpowiedniego certyfikatu

Jak widać na rysunku 5.49 pole *Podpisywanie jako* zostało zaktualizowane. W celu podpisania dokumentu, użytkownik powinien wybrać opcję *Podpisz* (rysunek 5.49).



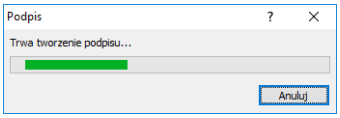
Rysunek 5.49: Microsoft Word - podpisywanie dokumentu po wybraniu właściwego certyfikatu

Następnie użytkownik zostanie poproszony o podanie kodu PIN karty. Po jego wpisaniu, należy zatwierdzić go przyciskiem *OK* (rysunek 5.50).



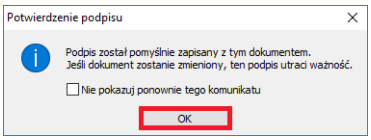
Rysunek 5.50: Microsoft Word - wprowadzanie kodu PIN

W następnym kroku generowany jest podpis elektroniczny i następuje podpisanie dokumentu (rysunek 5.51).



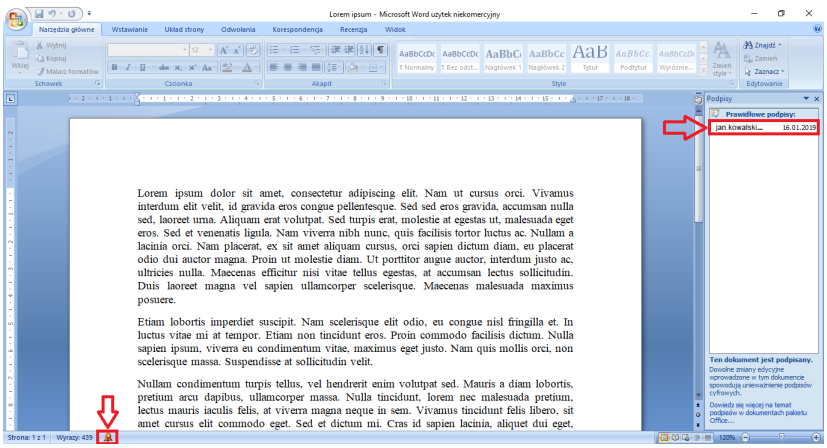
Rysunek 5.51: Microsoft Word - podpisywanie dokumentu po wybraniu właściwego certyfikatu

W ostatnim kroku użytkownik zostaje poinformowany, o tym że w przypadku edycji podpisanego pliku podpis elektroniczny utraci ważność. Okno należy zamknąć przyciskiem *OK*.



Rysunek 5.52: Microsoft Word - informacja od firmy Microsoft dotycząca ważności podpisu cyfrowego

Po wykonaniu przedstawionych powyżej kroków dokument zostaje podpisany, o czym zapewnia nas *Okienko zadań: Podpisy* lub/oraz odpowiedni symbol znajdujący się na *Pasku stanu* (rysunek 5.53).



Rysunek 5.53: Microsoft Word - podpisany dokument

Warto zaznaczyć, iż części dotyczące obsługi programów klienckich są ważnym elementem pracy, ponieważ ich wykorzystanie pozwala na pełne sprawdzenie poprawności działania apletu. W następnym rozdziale przedstawione zostało rozwiązanie umożliwiające korzystanie z oprogramowania pośredniczącego za pomocą aplikacji SmartCard Suite.

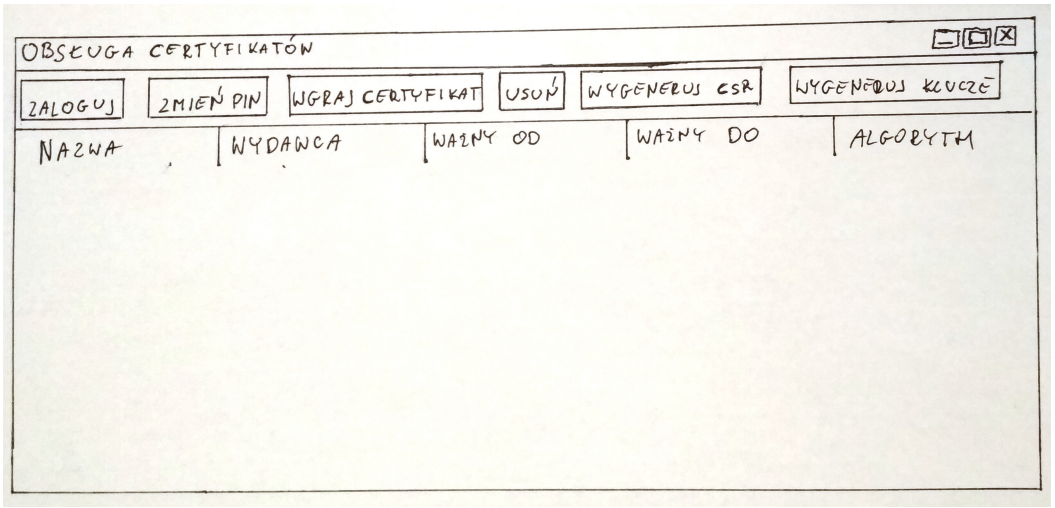
Rozdział 6

Wtyczka dla systemu SmartCard Suite

SmartCard Suite jest oprogramowaniem dla systemu Windows służącym do obsługi kart elektronicznych. Pierwsza wersja oprogramowania powstała w ramach pracy dyplomowej Mateusza Lesznera [7]. Kolejne wtyczki powstawały jako projekty w ramach zajęć laboratoryjnych przedmiotu Programowanie Kart Elektronicznych [55], który jest wykładany na Wydziale Informatyki Politechniki Poznańskiej. Wtyczka, która jest efektem tej pracy ma za zadanie dodawanie, usuwanie oraz podgląd certyfikatów a także zmianę kodu PIN karty.

6.1 Etap projektowania

Projektowanie wtyczki zostało rozpoczęte od stworzenia wstępnego szkicu dotyczącego interfejsu użytkownika. Musiały być w nim zawarte funkcje, które były realizowane przez oprogramowanie OpenSC, takie jak: łączenie z kartą, zmiana kodu PIN, odblokowywanie kodu PIN, wgrywanie i usuwanie certyfikatu, generowanie żądania podpisu certyfikatu oraz generowanie pary kluczy. Kolejnym założeniem była prostota w korzystaniu i przejrzystość aplikacji. Zostało to wykonane poprzez rozdzielenie każdej z funkcjonalności do osobnych, odpowiednio opisanych przycisków. Podgląd ma wyświetlać tylko najważniejsze informacje.



Rysunek 6.1: Projekt wtyczki

1. Menu (rysunek 6.1)

   a) Zaloguj
   Ustanawia połączenie z kartą podłączoną do wybranego terminala.

   b) Zmień PIN
   Pozwala na zmianę kodu PIN karty.

   c) Wgraj certyfikat
   Pozwala na wgranie nowego certyfikatu na kartę.

d) Usuń
Pozwala na usunięcie certyfikatu wraz z przypisaną parą kluczy i/lub usunięcie samej pary kluczy, jeśli nie jest przypisana do żadnego certyfikatu.

e) Wygeneruj CSR
Generowanie żądania podpisania certyfikatu.

f) Wygeneruj klucze
Generuje parę kluczy.

2. Okno wyświetlania

a) Nazwa
Wyświetla nazwę certyfikatu i/lub kluczy.

b) Wydawca
Organ autoryzujący podany certyfikat.

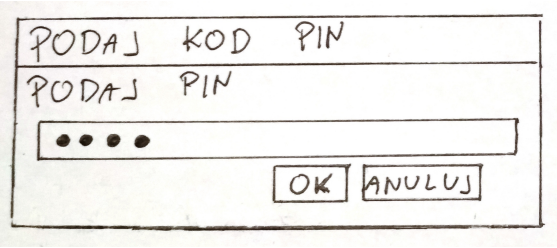
c) Ważny od
Data od której jest ważny podany certyfikat.

d) Ważny do
Data do której jest ważny podany certyfikat.

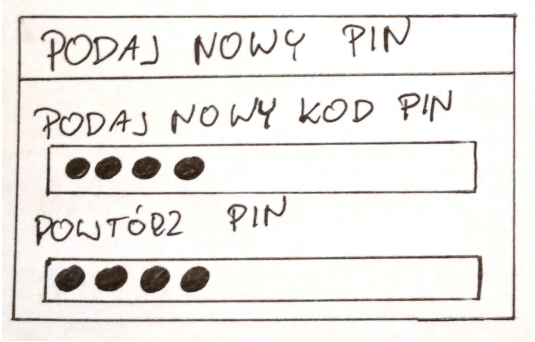
e) Algorytm
Algorytm, według którego został stworzony podany certyfikat i/lub klucz

3. Okna pomocnicze

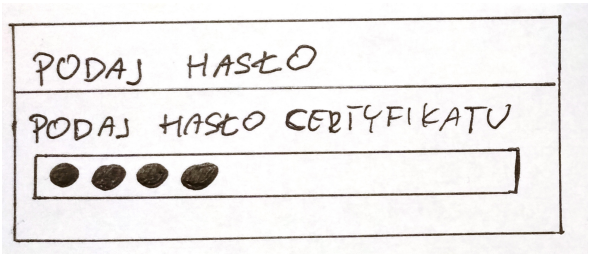
a) Okno wprowadzenia/zmiany kodu PIN/hasła (rysunek 6.2, 6.3, 6.4)
Okno służące do wprowadzania hasła do certyfikatu lub kodu PIN karty lub zmiany kodu PIN na nowy. W przypadku wprowadzania kodu PIN/hasła widoczne jest tylko pole tekstowe.



Rysunek 6.2: Okno wprowadzania kodu PIN



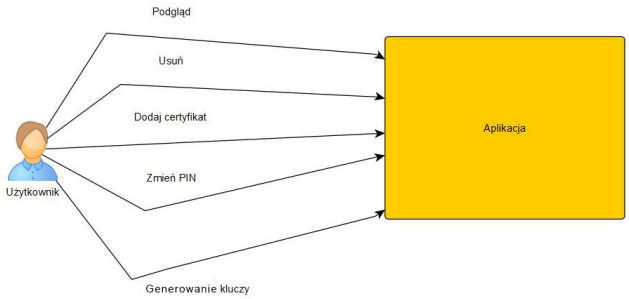
Rysunek 6.3: Okno zmiany kodu PIN



Rysunek 6.4: Okno wprowadzania hasła certyfikatu

6.2 Przypadki użycia

Głównym aktorem w korzystaniu z aplikacji SmartCard Suite jest użytkownik umieszczający kartę w terminalu. Schemat interakcji użytkownika i aplikacji przedstawia rysunek 6.5. Natomiast przypadki użycia zostały umieszczone w tabelach 6.1 i 6.2.



Rysunek 6.5: Interakcje aktorów w systemie

UC01: Zalogowanie do karty
Aktorzy: Użytkownik/Aplikacja
Scenariusz główny: 1. Aplikacja wyświetla listę dostępnych terminali. 2. Użytkownik wskazuje terminal, w którym znajduje się karta. 3. Użytkownik ustanawia połączenie z kartą znajdującą się w terminalu. 4. Użytkownik loguje się na kartę. 5. Aplikacja wyświetla prośbę o podanie kodu PIN/hasła karty. 6. Użytkownik wprowadza kod PIN/hasło. 7. Aplikacja wyświetla zawartość karty.
Rozszerzenia: 1a. Brak podłączonych czytników. 1a1. Aplikacja wyświetla komunikat o konieczności podłączenia czytnika. 2a. We wskazanym terminalu nie znajduje się karta. 3a. Podany PIN jest niepoprawny. 3b. Karta jest zablokowana.

Tabela 6.1: Przypadek użycia UC01: Odczyt i wyświetlenie danych z karty przez użytkownika

UC02: Zmiana kodu PIN
Aktorzy: Użytkownik/Aplikacja
Scenariusz główny: 1. Aplikacja wyświetla listę dostępnych terminali. 2. Użytkownik wskazuje terminal, w którym znajduje się karta. 3. Użytkownik ustanawia połączenie z kartą znajdującą się w terminalu. 4. Użytkownik wciska przycisk zmiany kodu PIN. 5. Aplikacja prosi o podanie obecnego kodu PIN karty. 6. Aplikacja wyświetla podpowiedź dotyczącą nowego hasła. 7. Aplikacja prosi o wpisanie nowego kodu PIN karty oraz o jego powtórzenie. 8. Aplikacja zmienia hasło.
Rozszerzenia: 1a. Brak podłączonych czytników. 1b. Aplikacja wyświetla komunikat o konieczności podłączenia czytnika. 2a. We wskazanym terminalu nie znajduje się karta. 5a. Podany kod PIN jest niepoprawny. 7a. Podany kod PIN jest niepoprawny. 7a1. Aplikacja przerywa zmianę kodu PIN. 7a2. Aplikacja wyświetla błąd zmiany kodu PIN. 7a3. Aplikacja zezwala na ponowne wprowadzenie nowego kodu PIN.

Tabela 6.2: Przypadek użycia UC02: Zmiana kodu PIN

UC03: Wgrywanie nowego certyfikatu na kartę
Aktorzy: Użytkownik/Aplikacja Warunki początkowe: UC01: Zalogowanie do karty
Scenariusz główny: 1. Użytkownik wciska przycisk wgrania nowego certyfikatu. 2. Aplikacja prosi o wybranie certyfikatu z systemu. 3. Aplikacja prosi o podanie hasła do certyfikatu. 4. Aplikacja dodaje nowy certyfikat. 5. Aplikacja wyświetla odświeżone dane z karty.
Rozszerzenia: 3a. Podany kod PIN jest niepoprawny. 3a1. Aplikacja przerywa wgrywanie certyfikatu.

Tabela 6.3: Przypadek użycia UC03: Wgrywanie certyfikatu na kartę

UC04: Usuwanie certyfikatu znajdującego się na karcie
Aktorzy: Użytkownik/Aplikacja Warunki początkowe: UC01: Zalogowanie do karty
Scenariusz główny: 1. Użytkownik wskazuje certyfikat do usunięcia 2. Użytkownik wciska przycisk usunięcia certyfikatu. 3. Aplikacja prosi o potwierdzenie usunięcia certyfikatu. 4. Aplikacja usuwa certyfikat i powiązane z nim klucze publiczne i prywatne. 5. Aplikacja wyświetla odświeżone dane z karty.
Rozszerzenia: 1a. Brak certyfikatów na karcie 1a1. Aplikacja wyświetla komunikat 4a. Nie udaje się usunąć klucza 4a1. Aplikacja wyświetla komunikat

Tabela 6.4: Przypadek użycia UC04: Usuwanie certyfikatu z karty

UC05: Generowanie nowej pary kluczy na karcie
Aktorzy: Użytkownik/Aplikacja Warunki początkowe: UC01: Zalogowanie do karty
Scenariusz główny: 1. Użytkownik wciska przycisk generowania nowej pary kluczy. 2. Aplikacja prosi o potwierdzenie wygenerowania nowej pary kluczy. 3. Aplikacja generuje nową parę kluczy. 4. Aplikacja wyświetla odświeżone dane z karty.
Rozszerzenia: 3a. Nie udaje się wygenerować pary kluczy 3a1. Aplikacja wyświetla komunikat

Tabela 6.5: Przypadek użycia UC05: Generowanie nowej pary kluczy

6.3 Opis implementacji

Wtyczka do obsługi certyfikatów został napisany w języku programowania C# oraz .Net Framework 4. Te technologie zostały wybrane, aby wtyczka była stworzona w tej samej technologii co reszta aplikacji SCS. Została również wykorzystana biblioteka OpenSC odpowiadająca za zmianę kodu PIN karty oraz SecureBlackBox do wgrywania certyfikatu na kartę i generowania nowej pary kluczy. Aby metody zawarte w bibliotece OpenSC mogły być wykorzystane w programie napisanym w języku C# (sam OpenSC napisany jest w języku C++), niezbędne było wykorzystanie dodatkowej biblioteki umożliwiającej mapowanie metod napisanych w innym języku na C#. W tym celu została wykorzystana biblioteka Pkcs11Interop.Net.

1. Ustanowienie połączenia z kartą znajdującą się w terminalu

   Ustanowienie połączenia z kartą znajdującą się w terminalu następuje przez wywołanie metody:

   ```
   private bool OpenSession(bool login = true)
   ```

   Powyższa metoda wywołuje metodę z biblioteki SecureBlackBox odpowiedzialną za faktyczne połączenie z kartą. Pierwszym argumentem jest typ użytkownika korzystającego z aplikacji, drugim PIN karty:

   ```
   Session.Login((int) SBPKCS11Base.Unit.utUser, Pin);
   ```

   Aby operacja połączenia z kartą się powiodła, należy poprawnie wpisać hasło, za co odpowiedzialna jest metoda:

   ```
   private Boolean RequestPassword(string Caption, string Prompt, out string Pass,
   bool showSecondPass = false)
   ```

2. Wprowadzanie kodu PIN

   Metodą odpowiedzialną za wprowadzanie kodu PIN jest:

   ```
   private Boolean RequestPassword(string Caption, string Prompt, out string Pass)
   ```

   Metoda tworzy też nowe okno, w którym wprowadza się PIN.

3. Zmiana kodu PIN

   Aby operacja zmiany kodu PIN była możliwa, niezbędne jest wykorzystanie bibliotek Pkcs11interop i OpenSC. Przy zmianie bibliotek zamykamy poprzednią sesję, wykorzystującą bibliotekę SecureBlackBox i otwieramy nową za pomocą procedury:

   ```
   OpenSessionInterop(out oldPin, CKU.CKU_USER);
   ```

   Poniższa metoda wymaga podania nowego hasła. W celu przejścia dalej niezbędne jest wprowadzenie poprawnego nowego hasła.

```
private Boolean ChangePassword(string Caption, string Prompt, out string Pass,
bool showSecondPass = false, string oldPass = null)
```

4. Sprawdzanie poprawności nowego kodu PIN

   Poprawność nowego kodu PIN jest sprawdzana poprzez metodę:

   ```
   private bool CheckPassword(string pass, string oldPass = null)
   ```

   Korzysta ona z wyrażeń warunkowych w celu sprawdzenia występowania poszczególnych znaków w nowym kodzie PIN. W przypadku braku któregoś ze znaków wyświetlany jest komunikat o jego braku. Wyrażenia warunkowe ustawiane są na dwa sposoby. Pierwszy - poprzez plik tekstowy odpowiadający za konfigurację hasła "passConfig.txt". Drugi jest domyślnie zaimplementowany w kodzie aplikacji i znajduje się w pliku "App.config".

5. Wgrywanie nowego certyfikatu

   Za wgrywanie nowego certyfikatu odpowiada metoda:

   ```
   private void AddCertificate()
   ```

   Wywołuje ona metodę (z biblioteki SecureBlackBox):

   ```
   Storage.Add(Cert, true);
   ```

   gdzie Storage to zasobnik karty.

6. Usuń

   Funkcja usuń dzieli się na dwie części. Pierwsza usuwa certyfikat wraz z przypisaną do niego parą kluczy, druga odpowiada za usunięcie wybranej pary kluczy.

   ```
   private void RemoveObject()
   {
           (...)
               RemoveCertificate(tag, out TElX509Certificate cert);
               RemoveKeyPairAfterCertificateRemove(cert);
           (...)
   }
   ```

   Usunięcie certyfikatu odbywa się poprzez metodę:

   ```
   private void RemoveCertificate(object tag, out TElX509Certificate cert)
   ```

   Usunięcie przypisania pary kluczy jest możliwe za pomocą metody:

   ```
   private void RemoveKeyPairAfterCertificateRemove(TElX509Certificate cert)
   ```

   Druga część metody odpowiada za usunięcie wybranej pary kluczy i odbywa się poprzez wywołanie metody:

   ```
   private void RemoveKeyPair(object tag, TElX509Certificate cert = null)
   ```

7. Tworzenie nowej pary kluczy

   Za wygenerowanie nowej pary kluczy odpowiedzialna jest metoda

   ```
   private KeyPair GenerateKeyPair()
   ```

6.4 Problemy podczas implementacji

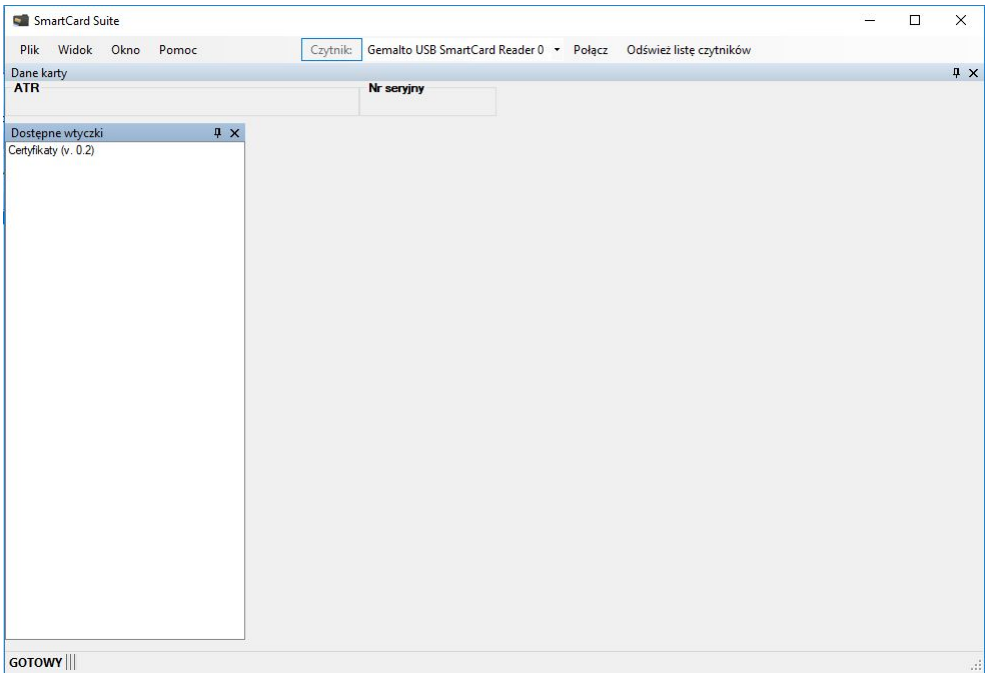
Głównym problemem podczas procesu implementacji wyżej opisanych metod były braki w bibliotece mapującej jak i samej bibliotece OpenSC. Niemożliwym było stworzenie funkcjonalności odblokowywania karty poprzez kod PUK. Było to spowodowane brakiem odpowiedniej funkcji mapującej w bibliotece Pkcs11Interop.Net. Brak odpowiedniej funkcjonalności w OpenSC spowodował też konieczność skorzystania z dodatkowej biblioteki SecureBlackbox w celu dodawania certyfikatu na kartę.

6.5 Opis kompilacji

Do kompilacji programu SmartCard Suite niezbędne jest zainstalowanie programu Visual Studio z pluginami odpowiadającymi za język C#. Po zainstalowaniu Visual Studio należy włączyć plik solucji "SmartCard Suite 2.0.sln" Następnie z górnego menu należy wybrać "Build, Build Solution". Po zbudowaniu plik .exe będzie się znajdować w "./SmartCard Suite 2.0/bin/Debug".

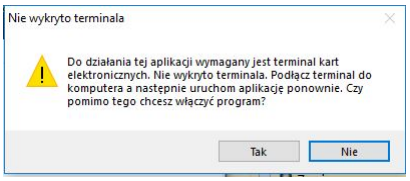
6.6 Instrukcja

Aby uruchomić aplikację SmartCard Suite, należy uruchomić plik wykonywalny otrzymany w trakcie kompilacji kodu źródłowego. Po uruchomieniu aplikacji pojawia się główne okno (rysunek 6.6).



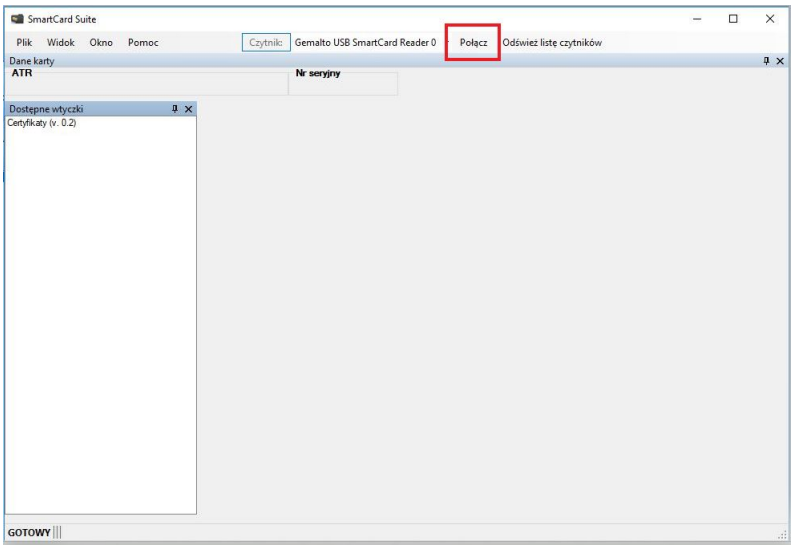
Rysunek 6.6: Główne okno programu SmartCard Suite

W przypadku pojawienia się powiadomienia o braku podłączonego terminala (rysunek 6.7) należy do komputera podłączyć odpowiedni czytnik kart elektronicznych (terminal).



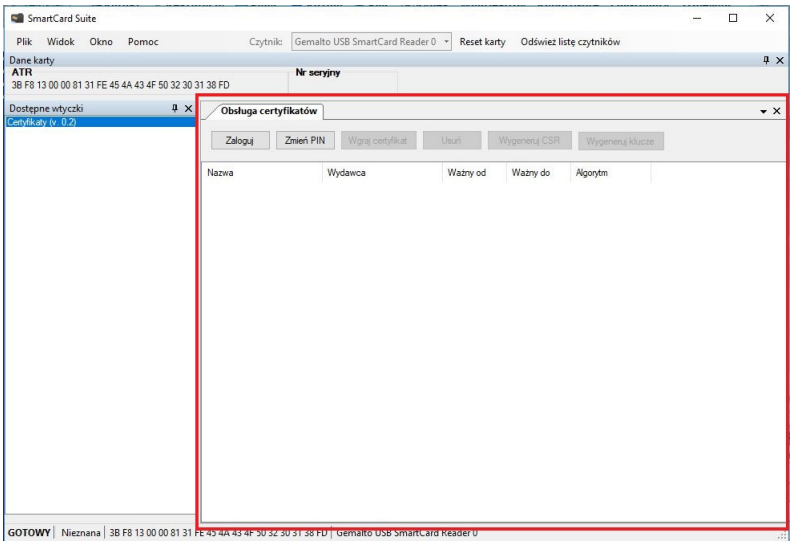
Rysunek 6.7: Powiadomienie o braku podłączonego terminala

Następnie należy połączyć się z terminalem. Aby to zrobić, należy kliknąć przycisk "Połącz"(rysunek 6.8).



Rysunek 6.8: Umiejscowienie przycisku "Połącz"

Po połączeniu z terminalem można przejść do właściwej pracy z wtyczką "Certyfikaty". Po dwukrotnym kliknięciu na nazwę wtyczki w menu dostępnych wtyczek, pojawi się okno Obsługi certyfikatów (rysunek 6.9).

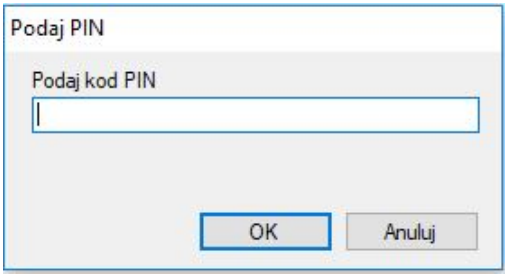


Rysunek 6.9: Wygląd wtyczki "Certyfikaty"

Na tym etapie można połączyć się z kartą lub zmienić jej kod PIN.

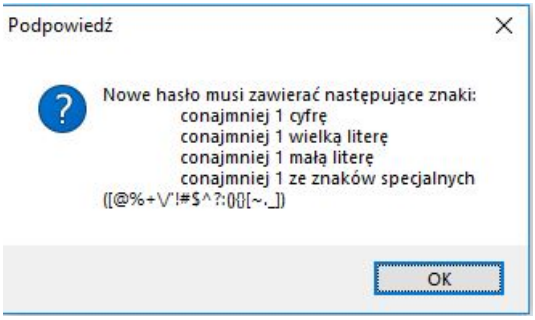
1. **Zmiana kodu PIN**

   W celu zmiany kodu PIN, należy nacisnąć przycisk "Zmień PIN" w górnym menu wtyczki. Użytkownik zostanie poproszony o podanie biblioteki, z jakiej ma korzystać aplikacja do obsługi potrzebnych nam metod. Dla isoApplet jest to plik opensc-pkcs11.dll. Spowoduje to wyświetlenie okna do podania obecnego kodu PIN (rysunek 6.10).



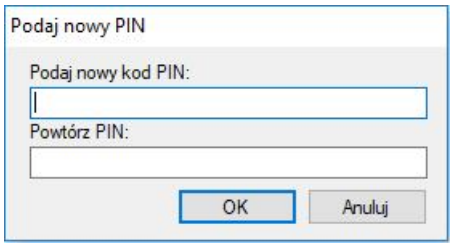
Rysunek 6.10: Okno wprowadzania kodu PIN

Po wprowadzeniu poprawnego kodu PIN, zostanie wyświetlona podpowiedź dotycząca wymaganej złożoności nowego kodu PIN (rysunek 6.11).



Rysunek 6.11: Okno podpowiedzi nowego hasła

Następnym krokiem jest podanie nowego kodu PIN, zgodnie z zaleceniami pokazanymi na wcześniejszej podpowiedzi. Nowy kod PIN należy potwierdzić podając go ponownie w polu "Powtórz PIN" (rysunek 6.12).

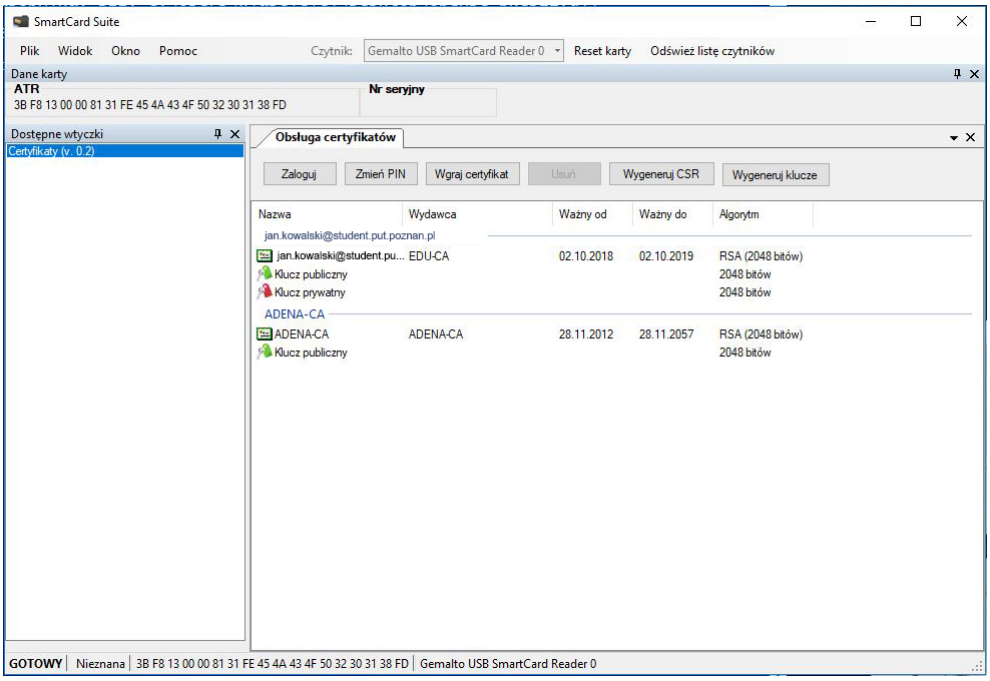


Rysunek 6.12: Okno wprowadzania nowego hasła

Użytkownik zostanie poinformowany o poprawnie zrealizowanej operacji zmiany kodu PIN w oknie potwierdzającym.

2. **Połączenie z kartą**

Aby połączyć się z kartą należy nacisnąć przycisk "Zaloguj" w górnym menu wtyczki. Użytkownik zostanie poproszony o wybranie biblioteki. Dla IsoApplet należy wybrać opensc-pkcs11.dll. Następnie użytkownik zostanie poproszony o podanie kodu PIN. Po jego poprawnym podaniu wyświetli się lista certyfikatów znajdujących się w pamięci karty (rysunek 6.13).



Rysunek 6.13: Wygląd okna po połączeniu z kartą.

Po połączeniu z kartą zostają odblokowane kolejne funkcje programu.

1. **Wgrywanie certyfikatu na kartę**

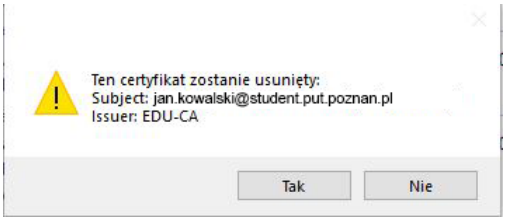
   Aby wgrać certyfikat na kartę użytkownik powinien nacisnąć przycisk "Wgraj certyfikat". Następnie należy wybrać certyfikat, który chcemy wgrać na kartę. W kolejnym kroku użytkownik zostanie poproszony o podanie hasła zabezpieczającego wybrany certyfikat (rysunek 6.14).

Rysunek 6.14: Okno wpisywania hasła certyfikatu.

   Po poprawnym podaniu hasła, lista certyfikatów zostanie zaktualizowana.

2. **Usuń**
   Z funkcji usuwania można skorzystać na dwa sposoby, usuwając certyfikat z przypisaną do niego parą kluczy lub usuwając parę kluczy nieprzypisaną do żadnego certyfikatu.

   a) Usuwanie certyfikatu
   W celu usunięcia certyfikatu należy zaznaczyć certyfikat z listy oraz nacisnąć przycisk "Usuń". Następnie w nowo otwartym oknie (rysunek 6.15), użytkownik zostanie poproszony o potwierdzenie usunięcia wybranego obiektu.



Rysunek 6.15: Okno potwierdzenia usunięcia certyfikatu.

b) **Usunięcie pary kluczy**
Aby usunąć parę kluczy nieprzypisaną do certyfikatu należy zaznaczyć klucz prywatny lub publiczny i nacisnąć przycisk "Usuń". Zostanie wtedy usunięta podana para kluczy i odświeżony widok certyfikatów.

3. **Wygenerowanie CSR**
Aby wygenerować żądanie podpisania certyfikatu należy kliknąć przycisk "Wygeneruj CSR". Użytkownik zostanie poproszony o podanie nazwy żądania. Po jej podaniu para kluczy zostanie wygenerowana na karcie, a żądanie zostanie zapisane na komputerze.

4. **Wygeneruj parę kluczy**
Aby wygenerować parę kluczy (klucz prywatny i publiczny) należy kliknąć przycisk "Wygeneruj parę kluczy". Ukaże się okno potwierdzające wygenerowanie pary kluczy.

5. **Tworzenie pliku konfiguracyjnego dla poprawności hasła**
Plik konfiguracyjny zawierający wyrażenie regularne, służące do sprawdzania poprawności hasła, musi znajdować się w tym samym katalogu co plik wykonywalny aplikacji SmartCard Suite. Składa się on z wierszy, w których są zapisane wyrażenia odczytywane później z aplikacji. Znaki, z których ma korzystać aplikacja, muszą być zapisane pomiędzy znakami "[" i "]". Zaleca się pisanie więcej bardziej specyficznych wyrażeń niż mniej bardziej ogólnych, zapewni to większą kontrolę nad warunkami hasła.

6.7 Przypadki testowe

Tworząc nową aplikację konieczne jest jej przetestowanie w celu sprawdzenia jakości i niezawodności. Każda z funkcjonalności wtyczki została poddana osobnym testom. Podczas testowania zostały wykryte błędy w szczególności podczas próby zmiany kodu PIN, było to spowodowane brakiem odpowiednich metod w bibliotece SecureBlackbox co spowodowało konieczność skorzystania z biblioteki OpenSC i dodatkowej biblioteki mapującej. Dalej problemem okazała się konieczność przełączania pomiędzy sesjami SecureBlackbox i OpenSC. Przeprowadzone testy przedstawiono w kompaktowej postaci w tabelach 6.6, 6.7, 6.8, 6.9, 6.10, 6.11.

Nazwa systemu/modułu	Certyfikaty
Nazwa przypadku testowego	Odczyt
Sygnatura przypadku testowego	P1
Testowana funkcjonalność	Wyświetlanie danych z karty
Sposób dostępu	Wybranie opcji "Zaloguj" w menu wtyczki
Dane wejściowe	Karta podłączona do terminala
Dane wyjściowe	Wyświetlone dane z pamięci karty

Tabela 6.6: Przypadek testowy P1: Wyświetlenie danych z karty.

Nazwa systemu/modułu	Certyfikaty
Nazwa przypadku testowego	Zmiana kodu PIN
Sygnatura przypadku testowego	P2
Testowana funkcjonalność	Zmiana kodu PIN karty
Sposób dostępu	Wybranie opcji "Zmień PIN" w menu wtyczki
Dane wejściowe	Karta podłączona do terminala, aktualny PIN i nowy PIN
Dane wyjściowe	Zmiana kodu PIN karty

Tabela 6.7: Przypadek testowy P2: Zmiana kodu PIN karty.

Nazwa systemu/modułu	Certyfikaty
Nazwa przypadku testowego	Wgranie certyfikatu
Sygnatura przypadku testowego	P3
Testowana funkcjonalność	Wgranie certyfikatu na kartę
Sposób dostępu	Wybranie opcji "Wgraj certyfikat"w menu wtyczki
Dane wejściowe	Karta podłączona do terminala, certyfikat znajdujący się na komputerze użytkownika, hasło certyfikatu
Dane wyjściowe	Wgranie certyfikatu na kartę

Tabela 6.8: Przypadek testowy P3: Wgranie certyfikatu na kartę.

Nazwa systemu/modułu	Certyfikaty
Nazwa przypadku testowego	Usunięcie certyfikatu
Sygnatura przypadku testowego	P4
Testowana funkcjonalność	Usunięcie certyfikatu razem z przypisaną parą kluczy
Sposób dostępu	Wybranie opcji "Usuń"w menu wtyczki
Dane wejściowe	Karta podłączona do terminala, wybrany certyfikat znajdujący się na karcie
Dane wyjściowe	Certyfikat usunięty z karty

Tabela 6.9: Przypadek testowy P4: Usunięcie certyfikatu.

Nazwa systemu/modułu	Certyfikaty
Nazwa przypadku testowego	Wygenerowanie CSR
Sygnatura przypadku testowego	P5
Testowana funkcjonalność	Generowanie CSR
Sposób dostępu	Wybranie opcji "Wygeneruj CSR"w menu wtyczki
Dane wejściowe	Karta podłączona do terminala
Dane wyjściowe	Utworzenie pary kluczy na karcie i żądania certyfikatu na komputerze użytkownika

Tabela 6.10: Przypadek testowy P5: Generowanie CSR.

Nazwa systemu/modułu	Certyfikaty
Nazwa przypadku testowego	Generowanie pary kluczy
Sygnatura przypadku testowego	P6
Testowana funkcjonalność	Generowanie pary kluczy na karcie
Sposób dostępu	Wybranie opcji "Wygeneruj parę kluczy"w menu wtyczki
Dane wejściowe	Karta podłączona do terminala
Dane wyjściowe	Utworzenie pary kluczy na karcie

Tabela 6.11: Przypadek testowy P6: Generowanie pary kluczy.

Rozdział 7

Podsumowanie

Głównym celem niniejszej pracy było znalezienie rozwiązania umożliwiającego wykorzystanie Infrastruktury Klucza Publicznego na Elektronicznej Legitymacji Studenckiej i Elektronicznej Legitymacji Doktoranckiej. Cel został osiągnięty przy wykorzystaniu istniejącego oprogramowania isoApplet. W ramach tego zadania dokonano dostosowania apletu do wymagań uczelni poprzez umożliwienie przechowywania na karcie kluczy prywatnych wygenerowanych poza nią. Ponadto została dokonana modyfikacja, która zapewnia wymóg zdefiniowania kodu PUK przez użytkownika karty. Dodatkowo zostały zawarte wskazówki pozwalające na wprowadzenie kolejnych poprawek celem szerszego dostosowania polityki bezpieczeństwa. W pracy zostały zawarte wszelkie potrzebne informacje, które pozwalają na odtworzenie niezbędnych zmian. Zostały dołączone pliki źródłowe apletu wraz z produktem kompilacji i narzędziem pozwalającym na instalację na kartach elektronicznych.

Kolejnym celem było przygotowanie oprogramowania pośredniczącego dla wybranego apletu. Cel został zrealizowany w oparciu o zmodyfikowaną bibliotekę OpenSC. Praca zawiera informacje pozwalające na dostosowanie jej dla kart użytkowanych przez uczelnię, a także pliki źródłowe oprogramowania middleware. Niezbędne poprawki biblioteki pozwalają na wykrywanie zdefiniowanych kart przez system Microsoft Windows. Oprogramowanie pośredniczące zostało przygotowane w postaci skryptów powłoki realizujących funkcjonalności testowania poprawnej instalacji apletu na karcie, zmiany oraz odblokowania kodu PIN, usunięcia kluczy, generacji kluczy i ładowania certyfikatów na kartę. W ramach tego celu zostały również przygotowane instrukcje pozwalające na wykorzystanie infrastruktury PKI oferowanej przez aplet z aplikacjami klienckimi i zasobnikiem systemu Microsoft Windows.

Ostatnim celem pracy było dostosowanie aplikacji SmartCard Suite do korzystania z apletu wybranego do realizacji pracy. Cel został osiągnięty poprzez stworzenie nowej wtyczki zawierającej niezbędne funkcjonalności. Korzystają one z bibliotek zgodnych z wybranym apletem. Aplikacja zezwala na dostęp do informacji zawartych na karcie, zmianę kodu PIN karty, import certyfikatów i generowanie pary kluczy. Opcja zmiany kodu PIN została rozszerzona o weryfikację zaawansowanej polityki bezpieczeństwa kodu PIN, opartej o plik konfiguracyjny.

Literatura

[1] N. P. Smart *A comparison of different finite fields for use in elliptic curve cryptosystems*, Department of Computer Science, University of Bristol, 2000

[2] M. Szychowiak, *Bezpieczeństwo systemów informatycznych: ćwiczenia z wykorzystaniem systemów Windows i Linux*, Wydawnictwo Politechniki Poznańskiej, 2017

[3] I. Setiadi, A. Miyaji, A. I. Kistijantoro *Elliptic Curve Cryptography: Algorithms and Implementation Analysis over Coordinate Systems*, 2014

[4] P. Nazimek, *Inżynieria programowania kart inteligentnych*, Politechnika Warszawska, 2005

[5] C. Adams, S.Lloyd, przekład W. Sikorski *PKI Podstawy i zasady działania: Koncepcje, standardy i wdrażanie infrastruktury kluczy publicznych*, Wydawnictwo Naukowe PWN SA, 2007

[6] M. Karbowski *Podstawy Kryptografii*, Helion, 2006

[7] *Praca magisterska: Projekt i implementacja uniwersalnej aplikacji do obsługi kart elektronicznych* M. Leszner, 2012

[8] P. Olszewski, *Przegląd modeli zaufania PKI*, Przegląd Teleinformatyczny, 3-4/2014

[9] F.L. Bauer *Sekrety Kryptografii: Metody i zasady kryptologii*, Helion, 2002

[10] P. Opitek *Skiming, Aspekty kryminalistyczne, Cyberprzestępczość w bankowości elektronicznej*, C.H.Beck, 2017

[11] W. Rankl, Effing W. *Smart Card Handbook, Fourth Edition*, Wiley, 2010

[12] L. Kohnfelder *Towards a Practical Public-Key Cryptosystem*, MIT, praca inżynierska, 1978

[13] Rozporządzenie Ministra Nauki i Szkolnictwa Wyższego z dnia 27 października 2018r. w sprawie studiów

[14] ISO/IEC 7816, International Organization for Standardization, Geneva, Switzerland.

[15] ISO/IEC 14443, International Organization for Standardization, Geneva, Switzerland.

[16] A. Jivsov, *ECC in OpenPGP*, Network Workign Group, InternetDraft, 2011

[17] ITU-T Recommendation X.509 *Information technology Open systems interconnection – The Directory: Public-key and attribute Certificate frameworks*, 2008

[18] RSA Laboratories, *PKCS#15 v1.1: Cryptographic Token Information Syntax Standard*, 2000

[19] D. Atkins, W. Stallings, P. Zimmermann *PGP Message Exchange Formats*, RFC1991, IETF, 1996

[20] B. Kaliski *PKCS#7: Cryptographic Message Syntax Version 1.5*, RFC2315, IETF, 1998

[21] B. Ramsdell *S/MIME Version 3 Certificate Handling*, RFC2632, IETF, 1999

[22] B. Ramsdell *S/MIME Version 3 Message Specification*, RFC2633, IETF, 1999

[23] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen *SPKI Certificate Theory*, RFC2693, IETF, 1996

[24] T. Dierks, E. Rescorla *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC5246, IETF, 2008

[25] S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC5280, IETF, 2008

[26] S. Frankel, S. Krishnan, *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*, RFC6071, IETF, 2011

[27] A. Freier, P. Karlton, P. Kocher *The Secure Sockets Layer (SSL) Protocol Version 3.0*, RFC6101, IETF, 2011

[28] K. Moriarty, M. Nystrom, S. Parkinson, A. Rusch, M. Scott, *PKCS #12: Personal Information Exchange Syntax v1.1*, RFC7292, IETF, 2014

[29] J. Pechanec, D. Moffat *The PKCS #11 URI Scheme*, RFC7512, IETF, 2015

[30] J. Buchmann *The Digital Signature Algorithm (DSA)*, Information-technology Promotion Agency, Japan, 2001

[31] `https://ant.apache.org/` [dostęp 10.11.2018]

[32] ByB [CC BY-SA 4.0 (`https://creativecommons.org/licenses/by-sa/4.0`)], from Wikimedia Commons

[33] `https://crocs.fi.muni.cz/` [dostęp 29.12.2018]

[34] `https://docs.microsoft.com/en-us/windows/desktop/seccrypto/certificate-trust-list-overview` [dostęp 23.11.2018]

[35] `https://github.com/martinpaljak/GlobalPlatformPro` [dostęp 10.01.2019]

[36] `https://github.com/martinpaljak/MuscleApplet` [dostęp 02.01.2019]

[37] `https://github.com/OpenSC/OpenSC` [dostęp 10.01.2019]

[38] `https://github.com/philipWendland/IsoApplet` [dostęp 11.11.2018]

[39] `https://globalplatform.org/about-us/` [dostęp 23.11.2018]

[40] `https://maths.ucd.ie/modules/math10040/Chapter2_13.pdf` [dostęp 08.01.2019]

[41] `https://searchsecurity.techtarget.com/definition/Public-Key-Cryptography-Standards` [dostęp 23.11.2018]

[42] `https://searchsecurity.techtarget.com/definition/Rijndael` [dostęp 08.01.2019]

[43] `https://searchsecurity.techtarget.com/definition/soft-token` [dostęp 25.11.2018]

[44] `https://www.appveyor.com/` [dostęp 10.01.2019]

[45] M. Szychowiak *Podstawy Kryptografii cz.1*, `https://www.cs.put.poznan.pl/mszychowiak/dydaktyka/student/Security_slajdy03_Kryptografia1.pdf` [dostęp 08.01.2019]

[46] `https://www.di-mgt.com.au/crt_rsa.html` [dostęp 08.01.2019]

[47] *Oficjalna strona EMVCo*, `https://www.emvco.com/`, [dostęp 11.11.2018]

[48] `http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-15-cryptographic-token-information-format.htm` [dostęp archiwalny 01.08.2015]

[49] `https://www.fi.muni.cz/~xsvenda/jcalgtest/` [dostęp 02.01.2019]

[50] `https://www.globalsign.com/en/blog/cryptographic-key-management-and-storage-best-practice/` [dostęp 02.01.2019]

[51] `https://www.gnupg.org/` [dostęp 10.11.2018]

[52] https://www.gov.pl/web/cyfryzacja/e-dowod-dowod-z-warstwa-elektroniczna [dostęp 23.11.2018]

[53] https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html [dostęp 23.11.2018]

[54] https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1003_DSA.pdf [dostęp 11.01.2019]

[55] http://www.mcp.poznan.pl/ [dostęp 20.11.2018]

[56] http://www.pi.gov.pl/parp/chapter_86197.asp?soid=D1547EEB6C0A40B8804E897CFDED2A7E [dostęp 23.11.2018]

[57] https://www.techopedia.com/definition/16132/public-key-cryptography-standards-pkcs [dostęp 23.11.2018]