UNIVERZITET SINGIDUNUM

DEPARTMAN ZA POSLEDIPLOMSKE STUDIJE

-MASTER STUDIJSKI PROGRAM-

Savremene informaciono komunikacione tehnologije

Pametne kartice u web okruženju

Mentor:

Prof. dr Mladen Veinović

Student:

Darko Dragić

Univerzitet Singidunum

Beograd 2010

Sadržaj

UVOD 7

CILJEVI I METODOLOGIJA *RADA* 9

Ciljevi rada 9

Metodologija rada 9

1. KRIPTOGRAFIJA 10

1.1 Simetrični algoritmi 11

1.2 Asimetrični algoritmi 13

1.3 Jednosmerne hash funkcije 14

2. DIGITALNI POTPIS 15

3. ALGORITMI ZA RAZMENU KLJUČEVA 17

3.1. RSA 17

3.1.1. Enkripcija 17

3.1.2. Dekripcija 18

3.1.3. Digitalno potpisivanje 18

3.1.4. Verificikacija digitalnog potpisa 18

4. PKI INFRASTRUKTURA 19

4.1 Uvod 19

4.2 PKI komponente 21

4.2.1 Sertifikacijski centar 21

4.2.2 Registracijski centar 22

4.2.3 Repozitorij 22

4.2.4 Krajnji entiteti 22

4.3 PKI funkcije upravljanja 23

4.3.1 Registracija 23
4.3.2 Inicijalizacija 23
4.3.3 Sertifikacija 23
4.3.4 Oporavak para ključeva 23
4.3.5 Ažuriranje ključeva 23
4.3.6 Zahtev za opoziv 24
4.3.7 Međusobno sertifikovanje 24
5. X.509 SERTIFIKAT JAVNOG KLJUČA 25
5.1 Uvod 25
5.2 Komponente sertifikata 26
5.2.1 Tamper-evident omotnica 26
5.2.2 Osnovni sadržaj sertifikata 27
5.2.3 Proširenja sertifikata 28
6. PAMETNE KARTICE (Smart Card) 30
6.1. Šta je to pametna kartica 31
6.2. Vrste pametnih kartica 31
6.2.1. Memorijske kartice 31
6.2.2. Kontaktne mikroprocesorske kartice 31
6.2.3. Bezkontaktne mikroprocesorske kartice 32
6.2.4. Kombinovane mikroprocesorske kartice 32
6.3 Standardi 33
6.3.1. ISO-7816 standard 33
6.3.2. EMV standard 35
6.4 Životni ciklus pametne kartice 36
6.4.1 Faza proizvodnje 36

6.4.2 Pripremna faza personalizacije 36
6.4.3 Faza personalizacije 36
6.4.4. Faza korištenja 37
6.4.5. Završna faza (faza poništavanja kartice) 37
6.5 Primena pametnih kartica 37
6.5.1. Elektronsko plaćanje 37
6.5.2. Računarska sigurnost 38
6.5.3. Telekomunikacije 38
6.6. Komunikacijski model 39
6.6.1. Protokoli za komuniciranje 39
6.7 APDU (eng. Application Protocol Data Units) 42
6.7.1. APDU naredba 42
6.7.2. APDU odgovor 43
7. JAVA CARD TEHNOLOGIJA 45
7.1. Java Card Virtual Machine 45
7.2. Java Card izvršna okolina 46
7.2.1. Životni ciklus Java Card virtualnog uređaja 47
7.2.2. Životni ciklus Java Card appleta 47
7.3. Java Card programsko okruženje 48
8. OPENCARD TEHNOLOGIJA 49
8.1. Arhitektura OpenCard tehnologije 49
8.1.1. Sloj terminala 50
8.1.2. Servisni sloj 51
8.2. Rad sa OpenCard tehnologijom 51
9. PRAKTIČAN DEO 53

9.1 Personalizacija kartice 53
9.2 Java Applet za digitalno potpisivanje upotrebom pametnih kartica..... 54
9.3 Problemi digitalnog potpisivanja dokumenata pomoću pametnih kartica u web okruženju 54
9.4 Kreiranje Smart Card Applet-a. 55
9.5 Standardi za pristup pametnim karticama..... 55
9.6 Pristup pametnim karticama iz Java-e 56
9.7 Konfigurisanje Sun PKCS#11 Privider 57
9.8 Statična registracija Sun PKCS#11 Provider 57
9.9 Dinamička registracija Sun PKCS#11 Provider 57
9.10 Konfiguracijski fajl za Sun PKCS#11 Provider..... 57
9.11 Upotreba Sun PKCS#11 Privider bez konfiguracijskog fajla 58
9.12 Odjavljivanje Sun PKCS#11 Provider 59
9.13 Pristup Keystore na pametnoj kartici..... 59
9.14 Preuzimanje sertifikata i privatnih ključeva sa pametne kartice 59
9.15 Potpisivanje podataka pomoću pametne kartice 60
9.16 Sistemski zahtevi za pristup pametnim karticama pomoću Java appleta 61
9.17 Implementacija Applet-a za potpisivanje pomoću pametnih kartica 61
9.18 Kako funkcioniše applet za potpisivanje pomoću pametne kartice 62
9.19 Kopmajliranje i potpisivanje Applet-a..... 62
9.20 Testiranje applet-a pomoću jednostavne HTML forme..... 64
PRIMER..... 64
ZAKLJUČAK 67
LITERATURA..... 69
DODATAK..... 70

Sažetak

U ovom radu prikazane su kriptografske osnove savremenih sistema za zaštitu podataka. Razmatrani su simetrični i asimetrični šifarski sistemi, tehnike digitalnog potpisivanja i tehnike za distribuciju šifarskih ključeva. Posebna pažnja je posvećena PKI sistemima i standardu X.509 za izdavanje javnih digitalnih sertifikata. Prikazane tehnike danas se široko primenjuju na Internetu za ostvarivanje funkcija poverljivosti, identifikacije i autentifikacije, koje su osnova za realizaciju elektronskog poslovanja na Internetu.

U radu su predstavljene karakteristike pametnih kartica kao i razlozi za njihovu sve veću i širu upotrebu kao jednog od vitalnih elemenata zaštite informacionih sistema. Pametne kartice poboljšavaju pogodnosti upotrebe on-line transakcija i obezbeđuju njihovu sigurnost. Koriste se u aplikacijama koje zahtevaju visok nivo bezbednosti kao i za potvrdu autentičnosti.

Ključne reči: zaštita podataka, kriptografija, šifarski sitemi, ključevi, digitalni sertifikati, pametne kartice, protokoli, bezbednost, zaštita, potvrda autentičnosti

Abstract

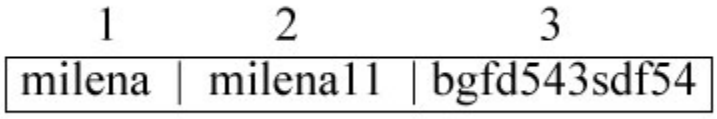
The cryptographic bases of contemporary systems of data protection are shown in this paper. Various symmetric and asymmetric cipher systems are discussed, as well as digital signature techniques and the key distribution techniques with the special reference to the PKI systems and X.509 standard. The presented techniques are nowadays widely applied on the Internet to ensure the functions of confidentiality, identification and authentication, which are fundamental for e-commerce realisation on the Internet.

In this paper are explained the characteristics of smart cards and the reasons for their growing usage as one of the vital part of the information systems protection. Smart cards greatly improve the convenience of on-line transactions as well as their security. Smart Cards are used in applications which require strong security protection and authentication.

Key words: data protection, cryptography, cipher systems, keys, digital certificates, smart cards, protocols, security, protection, authentication

UVOD

Sa ubrzanim svakodnevnim napretkom tehnologije dolazi i do sve većeg zaostajanja različitih do tada efikasnih rešenja u oblasti zaštite sistema, tako da uspešna rešenja vrlo brzo gube na svojoj efikasnosti. Proučavanjem trenutne situacije shvatio sam da je na internetu najrasprostranjenije logovanje putem formi gde korisnik unosi svoje podatke (najčešće username i password). Ova tehnika i dalje funkcioniše međutim polako se pokazuju i njene slabosti. Sagledavanjem samog Interneta odnosno aktivnosti korisnika na Internetu zaključio sam da svaki pojedinac koji aktivno ili manje aktivno funkcioniše na internetu u proseku poseduje minimum 4-5 online naloga koji su zaštićeni lozinkama. Analiziranjem ponašanja na internetu nekolicine prosečnih korisnika došao sam do zaključka da oni najčešće koriste identične podatke za autorizaciju na svim online servisima koje koriste. Razlog ovoga trenda jeste to što se podaci za logovanje najčešće pamte i zbog toga je lakše zapamtiti jednu ili dve lozinke koji će se primenjivati u svim ostalim aktivnim online servisima gde je potrebno da se korisnik loguje. S obzirom da se ovi podaci (prvenstveno lozinke) pamte, one često budu neka logična rešenja ali ne i suviše kompleksne reči. Ovo dovodi do prve slabosti samih lozinki jer napadač vremenom može da upozna žrtvu i na osnovu podataka koje je prikupio može da pretpostavi i lozinku koju žrtva koristi. Druga slabost jeste i sama dužina lozinke kao i raznolikost karaktera od kojih se lozinka sastoji. Pored mnogobrojnih upozorenja od strane online servisa prilikom registrovanja novih korisnika da lozinku koju izaberu što više zakomplikuju ,na žalost one ipak na kraju ostaju ne toliko zaštićene. Slika 1 prikazuje raznolikost lozinki i težinu pamćenja istih.



Slika 1 prikaz različitih oblika lozinki.

Kao što vidimo na slici 1.1 lozinke mogu biti jednostavne koje se vrlo lako pamte kao i mnogo kompleksnije ali teške za pamćenje.Problem kod jednostavnih lozinki je što se veoma lako mogu provaliti i ne predstavljaju baš veliku prepreku napadačima. Na slici 1 pod rednim brojem 2 prikazan je najčešći oblik lozinke gde korisnici ubacivanjem brojeva na kraj komplikuju svoju lozinku a opet lako pamte tu kombinaciju. Na žalost kako tehnologija napreduje ove šifre se sve lakše i lakše provaljuju i postoji mogućnost da provaljivanjem lozinke napadač može da ukrade online identitet svoje žrtve i nanese veliku štetu. U ovom radu neću detaljno objašnjavati postupak provaljivanja samih lozinki ali ću samo navesti neke od postojećih metoda kojim se napadači služe :

- **Brute Force Attack** - napadač uz pomoć alata napada nalog žrtve tako što isprobava sve moguće kombinacije izabranih karaktera. (Ovaj vid napada je

veoma spor pogotovu ukoliko žrtva koristi dovoljno dugačku i kompleksnu lozinku)

- **Word List Scanner** – napadač koristi unapred formiranu listu mogućih lozinkih i isprobava svaku ponaosob dok prijava ne uspe.
- **KeyLog alati** – mali program koji se na razne načine ubacuje na računar žrtve i prati kompletan unos podataka na tastaturi žrtvinog kompjutera i taj sadržaj potom prosleđuju putem interneta napadaču.
- **Men in the middle** – Prisluškivanje predstavlja praćenje protoka mreže između žrtve i servisa koji nudi uslugu. Napadač može da prisluškuje odn. da presretne sve podatke koji se razmenjuju u procesu i samim tim može da ih preuzme , otkrije ili čak izmeni i načini veliku štetu žrtvi. Zbog ove pretnje, uvodimo sigurnosne protokole koji drže napadače na distanci.

Postoji još načina kako da napadač dođe do žrtvine lozinke i samim tim preuzme kontrolu nad tuđim nalozima ali zbog obimnosti teme odlučio sam samo da napomenem neke od njih kako bih prikazao postojeće probleme koje želim da zaobiđem idejnim rešenjem koje ću objasniti u ovom radu.

Najveći nedostatak upotrebe više online servisa gde je potrebno logovati se jeste taj što korisnici najčešće koriste iste podatke (korisničko ime , e-mail adresu i lozinku) . Upotrebna vrednost ovih servisa na internetu varira kao i njihova tehnička zaštita. Postoje servisi koji su napravljeni od strane jednog ne baš iskusnog programera koji nije primenio dovoljnu zaštitu podataka i oni servisi koji su u svoju zaštitu uložili i na desetine miliona dolara. Problem nastaje kada korisnik ne obraća pažnju na ovo i ostavlja identične podatke na sve servise čak i na one slabo zaštićene čime otvara vrata potencionalnom napadaču da preuzme online identitet korisnika.

Kako bi zaštita jednog sitema uspešno funkcionisala moramo da obratimo pažnju na sve slabe tačke i sprečimo mogućnost otkrivanja odnosno zloupotrebe zaštićenih podataka. Iako mi uložimo veliki napor i trud a samim tim i velika ulaganja a ne zaštitimo u potpunosti naš sistem ili ostavimo ranjivu jednu kariku u lancu zaštite kompletan bezbedonosni sistem postaje ranjiv i daje mogućnost potencijalnom napadaču da izvrši uspešan napad na naš servis. Posledice ovog napada mogu da budu na žalost neprocenjljive.

CILJEVI I METODOLOGIJA *RADA*

Ciljevi rada

Pri realizaciji ciljeva i zadatka rada, pored sve dostupne literature (udžbenici, časopisi i Internet) nastojao sam obuhvatiti podatke o karticama koje se koriste u svetu.

Ovaj rad ima prevashodni cilj da nas upozna sa smart karticama, njihovom standardu, tehnologiji i primeni, naravno i sigurnosti.

Pored toga rad treba da ukaže na prednosti korištenja smart kartica i da ukaže na širok spektar mogućnosti koje se nude krajnjim korisnicima.

Metodologija rada

„Najjednostavnija je definicija „metodologije" (grčki „metodhs + logos" = reč, govor, nauka o metodama znanstvenog istraživanja) da je to nauka o metodama naučnog istraživanja. U širem smislu metodologija je nauka o celokupnosti svih oblika i načina istraživanja pomoću kojih se dolazi do sistemskog i objektivnog naučnog znanja, odn. naučna disciplina u kojoj se kritički ispituju i ekslpicitno izlažu različite opšte i posebne naučne metode, naziva se metodologija ".[1]

U ovom radu, da bi se postigli postavljeni ciljevi kao i što efikasniji rezultat koji se dobijaju povezivanjem i odgovarajućom kombinacijom više naučnih metoda, korištene su sledeće metode:

- metoda deskripcije,
- komparativna metoda,
- matematička metoda,
- metoda analize,
- metoda sinteze

Metoda deskripcije, definisana kao postupak jednostavnog opisivanja ili ocrtavanja činjenica, procesa i predmeta, te njihovih empirijskih potvrđivanja odnosa i veza, ali bez naučnog tumačenja i objašnjavanja, korištena je kroz celi rad.

Komparativna metoda, koja podrazumeva upoređivanje istih ili srodnih činjenica, pojava, procesa i odnosa, odnosno utvrđivanje njihove sličnosti u ponašanju i intenzitetu kao i razlika među njima.

---

[1] *Prof.dr.Čekić Š.: „Osnovi metodologije i tehnologije izrade znanstvenog i stručnog djela“, Sarajevo 1999., str.33*

Matematička metoda, je naučni sistemski postupak, a sastoji se u primeni matematičke logike, simbola, brojnih matematičkih operacija. Kratko rečeno predstavlja matematički način zaključivanja.

Metoda analize, je postupak naučnog istraživanja i objašnjavanja stvarnosti putem raščlanjivanja složenih misaonih tvorevina (pojmova, sudova i zaključaka) na njihove jednostavnije delove i elemente, kao i izračunavanje svakog dela (elementa) za sebe i u odnosu na druge delove, celine.

Metoda sinteze, je suprotna metoda od analize koja se takođe primjenjivala u izradi ovog rada.

1. KRIPTOGRAFIJA

Krajem 20. veka informacija postaje roba, i to vrlo tražena. Samim time raste značaj njenog čuvanja i transporta. Paralelno na značaju dobija i kriptografija, naučna disciplina koja se bavi informacijom i očuvanjem njene tajnosti. Upravo ona postaje glavno sredstvo ostvarenja privatnosti, poverenja, kontrole pristupa, elektronskog plaćanja…

U kriptografskoj terminologiji, poruka se naziva jasni ili otvoreni tekst (***plaintext, cleartext***). Kodiranje poruke u svrhu skrivanja njenog sadržaja od uljeza, naziva se kriptovanje (***encryption***). Kriptovana poruka se naziva šifrirani tekst ili šifrat (***ciphertext***). Obrnuti postupak dešifrovanja naziva se dekriptovanje (***decryption***). Svi moderni kriptografski algoritmi, za kontrolu kodiranja, koriste ključ (***key***) te se oslanjaju na tajnost samoga ključa, a ne na tajnosti samog algoritma. Algoritmi se objavljuju kako bi ih ljudi mogli proučavati, uočiti njihove slabosti te ih poboljšati ili izbaciti iz upotrebe. Bitna je i dužina ključa, jer njenim povećanjem raste broj kombinacija koje uljez mora isprobati da bi zaštitu razbio.

Mnogi algoritmi kodiraju podatke po blokovima. To znači da oni uzimaju blok po blok podataka i transformiraju ga u blok iste dužine koristeći funkciju ovisnu o ključu kriptovanja. Tipična dužina bloka je 64 bita.
Postoje dve vrste algoritama koje se oslanjaju na tajnost ključa, simetrični (***secret-key***) i asimetrični (***public-key***) algoritmi. Razlika je u tome da simetrični algoritmi koriste isti ključ za procese kriptovanja i dekriptovanja, dok asimetrični imaju dva ključa, gdje jedan ključ ne možemo izvesti iz drugoga.

Kriptografski produkti će biti otporni na napade ukoliko se algoritmi pažljivo implementiraju. Loše implementacije će biti ranjive za neke vrste.

Odnos dužine ključa i sigurnostialgoritma prikazana je u tabeli 1-1

.

	dužina ključa (bitovi)		
	nedovoljna	trenutno dovoljna	dovoljna u daljoj budućnosti
simetrični algoritmi	40	80	128
asimetrični algoritmi	< 768	1024	2048

Tabela 1-1. Odnos dužine ključa i sigurnosti algoritma

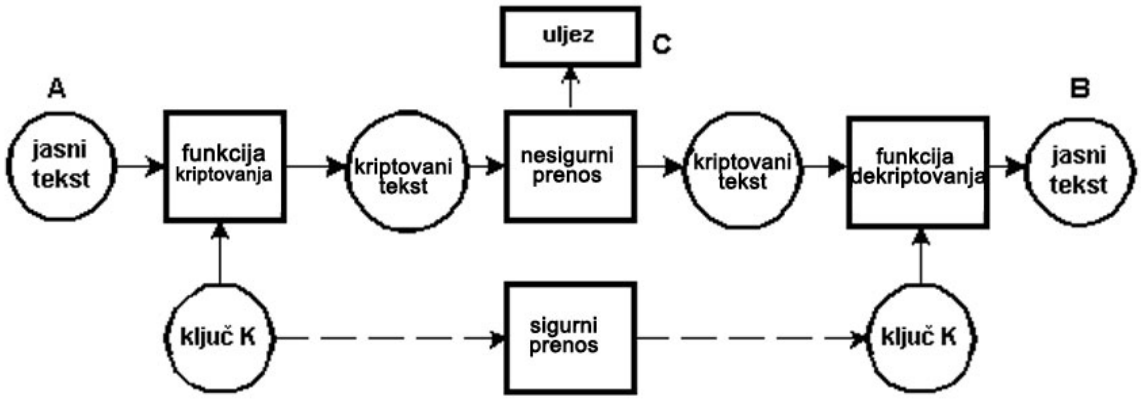
1.1 Simetrični algoritmi

Dve ili više osoba dele jedan te isti ključ kojim šifriraju i dešifriraju poruke. Time se ostvaruje tzv. siguran komunikacijski kanal između grupe ljudi. Glavna prednost ovih sistema je jednostavnost i brzina ali postoji i nekoliko mana:

- kod uspostavljanja sigurnog kanala treba dogovoriti koji će se ključ koristiti, te ga fizički ili na neki drugi siguran način (u tom trenutku siguran komunikacijski kanal još ne postoji) preneti svim korisnicima kanala, što može biti veliki problem ako je fizička udaljenost korisnika velika.
- ukoliko ključ služi za sigurnu komunikaciju između samo dve osobe (tj. može se relativno sigurno odrediti koja je osoba poruku poslala), da bi se takav sistem realizirao za grupu od n osoba potrebno je n*(n-1)/2 ključeva.

Problem velikog broja ključeva se rešava uvođenjem centara za distribuciju ključeva (*key distribution centre – KDC*), kome svi korisnici sistema veruju. Svaki korisnik sada deli tajni ključ samo s *KDC-om*, koji čuva tajne ključeve ostalih korisnika. Pri tome korisnik ne mora znati ključeve drugih korisnika, nego samo njihove identifikatore po kojima će ih *KDC* razlikovati od ostalih korisnika .

Princip rada simetričnog algoritma prikazan je na slici 1-1. Osoba A šifruje poruku ključem K, koji se mora unapred dogovoriti, i šifrovanu poruku šalje osobi B. Osoba B poruku prima i dešifruje je istim ključem. Ukoliko ključ padne u ruke osobi C, nastaju veliki problemi. Osoba C je tada u mogućnosti čitati ne samo sve buduće poruke, već i one stare, ukoliko su kriptovane istim ključem.



Slika 1-1. šema delovanja simetričnog algoritma kriptovanja

U praksi se trenutno koristi više simetričnih algoritama, a razvijaju se i novi budući da je cilj ostvariti što brži i sigurniji algoritam. Brzina izvođenja algoritama poboljšava se korišćenjem novih metoda kriptovanja, dok se sigurnost povećava uz korišćenje većih ključeva. Valja napomenuti da se brzina izvođenja algoritma može znatno ubrzati hardware-om .

Najpoznatiji simetrični algoritmi su:

- ***DES** (Data Encryption Standard)* – algoritam je razvijen 70-ih godina. U Americi je proglašen standardom, a korišten je širom sveta. Široku primenu je našao u financijskoj industriji. DES je algoritam koji kriptuje blokove podataka. Veličina bloka je 64 bita. Ključ je veličine 56 bita, što algoritam čini prilično nesigurnim. Naime, vlade, kriminalne organizacije ili velike kompanije imaju dovoljno sredstava da osiguraju moderne računare, odnosno hardware posebne namene i tako relativno lako otkriju ključ. Zbog toga ovaj algoritam ne bi trebalo koristiti za razvoj novih sistema.
  U novije se vreme pojavio ***Triple-DES*** (***3DES***), baziran na upotrebi DES algoritma tri puta (kriptuj-dekriptuj-kriptuj) s tri različita ključa. Zato ga mnogi smatraju mnogo sigurnijim od običnog DES-a .

- ***Blowfish*** – algoritam je razvio Bruce Schneier. Algoritam kriptuje blokove podataka dužine 64 bita. Dužina ključa je varijabilna i kreće se od 64 do 448 bita. Primenu je našao u mnogim aplikacijama. Još nije poznato da je neko uspešno razbio ovaj algoritam.

- ***IDEA** (International Data Encryption Algorithm)* – razvijen u Švajcarskoj, algoritam koristi ključ dužine 128 bita. Smatra se veoma sigurnim. U nekoliko

godina korištenja, nije objavljen niti jedan uspešan napad, iako je bilo mnogo pokušaja.
Algoritam je patentiran u Americi i većini evropskih zemalja, a upotreba u nekomercijalne svrhe je neograničena.

- ***RC4*** – algoritam je razvila RSA Data Security kompanija s namerom da ostane tajan. Tako je i bilo sve dok mu neko nije objavio izvorni kod [4] . Algoritam je vrlo brz. Pouzdanost mu još nije utvrđena, ali razbijanje se ne čini trivijalnim. Ključevi mogu biti proizvoljne dužine. RC4 je u suštini generator pseudo-slučajnih brojeva. Podatke kriptuje tako da ih dovodi na izlaz generatora gdje se vrši XOR (isključivo-ili) logička operacija. Zbog toga se ne sme isti ključ koristiti za kriptovanje različitih podataka.

1.2 Asimetrični algoritmi

Asimetrični algoritmi koriste različite ključeve u postupcima kriptovanja i dekriptovanja. Vrlo su važni jer se mogu koristiti za prenos podataka čak i kada sudionici komunikacije nemaju mogućnost dogovaranja tajnog ključa. Sve poznate metode su prilično spore, te se stoga nastoje pronaći nova praktičnija rešenja. Zbog svoje sporosti, asimetrični se algoritmi uglavnom koriste za prenos ključa sesije (*session key*), koji tada koristimo za brzo kriptovanje podataka simetričnim postupcima.

Svaki korisnik koji se služi asimetričnim algoritmima mora imati dva ključa: javni i tajni. Javni ključ je javno objavljen i može biti poznat svima onima koji to žele. Tajni ključ zna samo ona osoba koja ga poseduje. Da bi asimetrični algoritmi bili funkcionalni, taj ključ niko drugi ne sme saznati. Princip rada algoritama je sledeći: osoba A šalje poruku osobi B, tako da poruku kriptuje javnim ključem osobe B. Kriptovanu poruku može videti bilo ko, ali od toga neće imati koristi budući da samo osoba B zna tajni ključ kojim će kriptovanu poruku moći dekriptovai, a samim time i čitati.

Problem distribucije ključeva se javlja i kod asimetričnih algoritama. Problem nije u čuvanju tajnosti ključa, jer se ovde distribuišu samo javni ključevi, nego korisnik koji primi nečiji javni ključ mora biti siguran da je taj ključ, javni ključ baš toga korisnika, a ne nekog drugog. Naime, uljez može generisati javni i tajni ključ, objaviti tajni ključ u nečije ime i primati i čitati poruke koje nisu njemu namenjene. Rešenje problema je slično rješenju simetričnih algoritama uz korištenje Menadžera javnih ključeva (*public-key manager – PKM*) koji održava bazu javnih ključeva svih korisnika.

Najpoznatiji asimetrični algoritmi su:

- ***RSA*** *(Rivest-Shamir-Adelman)* – najrašireniji je asimetrični algoritam, a koristi se za kriptovanje i za potpisivanje. Sigurnost mu leži u dužini ključa i težini faktorizacije velikih prostih brojeva. Uz pažljivu upotrebu algoritma, sigurnost je zagarantovana. Ranjiv može postati jedino uz veliki napredak postupaka faktorizacije velikih prostih brojeva.

- ***Diffie-Hellman*** – obično se koristi za razmenu ključeva. Generalno se smatra sigurnim uz korištenje prikladnih generatora slučajnih brojeva i dovoljno dugih ključeva. Sigurnost algoritma leži u problemu diskretnih logaritma (ekvivalentno faktorizaciji velikih prostih brojeva).

1.3 Jednosmerne hash funkcije

Kao što i samo ime govori, jednosmerne funkcije su funkcije kod kojih je relativno jednostavno izračunati $y = f(x)$ ali je vrlo teško (praktično ne moguće u nekom razumnom vremenu) izračunati $x = f^{-1}(y)$.

Razbijanje tanjira čini dobru analogiju s jednosmernom funkcijom: tanjir je jednostavno razbiti, ali je malo teže ponovno ga sastaviti.

Za *hash* funkcije upotrebljava se više naziva: funkcija kompresije, funkcija kontrakcije, sažetak poruke, otisak, kriptografski podatak, kontrola integriteta poruke,…

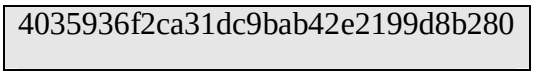
Hash funkcija je funkcija koja kao ulaz prima niz promenljive dužine, a kao izlaz daje niz fiksne dužine. Jednostavna *hash* funkcija je XOR funkcija: obavljanje XOR operacije nad svim članovima niza daje broj fiksne dužine. Primena *hash* funkcija je u proveri identičnosti dveju poruka upoređivanjem njihovih sažetaka. Pomoću jednosmernih *hash* funkcija jednostavno je izračunati sažetak poruke, ali je gotovo nemoguće rekonstruisati poruku iz njenog sažetka. Dobre jednosmerne *hash* funkcije nemaju dva ili više ulaznih nizova za koje daju isti izlaz.

Najrašireniji *hash* funkcije su: ***MD2, MD4*** – stare verzije algoritama iz RSA. Budući da poseduje određene nedostatke, njihova se primena ne preporučuje.

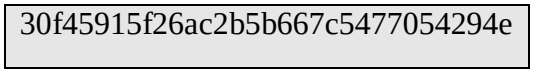
- ***MD5*** *(Message Digest Algorithm 5)* – ovaj sigurni *hash* algoritam je razvijen u RSA Data Security. Nizove proizvoljne dužine sažima u podatak dužine 128 bita. U širokoj je upotrebi.

- ***SHA*** *(Secure Hash Algorithm)* – ovaj algoritam je prilično nedavno objavila vlada SAD-a. Nizove proizvoljne dužine, algoritam sažima u podatak dužine 160 bita. Smatra se prilično dobrim algoritmom i zbog toga ću ga upotrebiti u ovom radu.

- ***RIPEMD-160*** – prilično novi algoritam, dizajniran da zameni MD5. Nizove proizvoljne dužine sažima u podatak dužine 20 bajta. Brzina rada mu je oko 40 Mb/s na 90 MHz pentiumu.



Slika 1-2. MD5 hash reči "beograd"



Slika 1-3. MD5 hash reči "Beograd"

Na slici 1-2. i 1-3. je prikazana hash funkcija reči beograd i Beograd. Iz date slike može se primetiti da i ukoliko nad izvornim tekstom napravimo i najmanju izmenu (u ovom slučaju reči se razlikuju u prvom slovu odn. po veličini prvog slova) rezultat hash funkcije će se drastično promeniti.

2. DIGITALNI POTPIS

Digitalni potpis je kombinacija sažetka poruke i asimetričnog algoritma, a pruža funkcionalnosti autentifikovanja pošiljaoca, očuvanje integriteta poruke, te nemogućnost krivotvorenja i ponovnog korištenja. Digitalni potpis se dobija izračunavanjem sažetka poruke, a zatim kriptovanjem tog sažetka privatnim ključem pošiljaoca. Kriptovani sažetak može svako dekriptovati uzimanjem javnog ključa pošiljaoca, ali je samo on mogao stvoriti svoj digitalni potpis jer samo on poseduje svoj privatni ključ. Primaoc poruke može proveriti digitalni potpis tako da javnim ključem pošiljaoca dekriptuje sažetak poruke, a zatim izračuna svoj sažetak primljene poruke i uporedi ga dobijenim dektiptovanjem sažetka. Ako su sažeci identični, poruka je autentična jer je jedino pošiljaoc mogao kriptovati sažetak svojim privatnim ključem. Digitalni potpis ne osigurava tajnost pa je poruku potrebno zaštititi kriptovanjem, npr. javnim ključem primaoca nakon potpisivanja.

"Primenjena kriptografija – Bruce Schneier str.37"

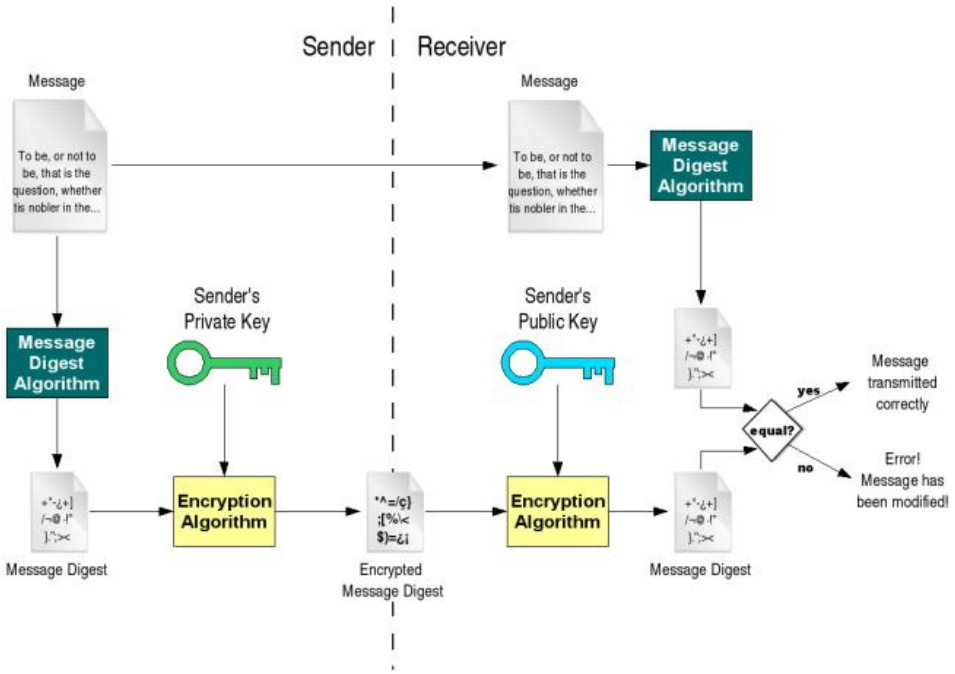
Primer protokola digitalnog potpisa

1. Alisa šifruje dokument pomoću svog javnog ključa i pri tome ga potpisuje.
2. Alisa šalje Bobu potpisani dokument.
3. Bob dešifruje dokument pomoću Alisinog javnog ključa, pri tome proveravajući potpis.

Ovaj primer prikazuje tok digitalnog potpisivanja bez posrednika odnosno trećeg lica kome veruju obe strane. Ovim poostupkom obezbedili smo pet osnovnih zahteva digitalnog potpisa:

1. Potpis je autentičan: kada Bob proveri poruku pomoću Alisinog javnog ključa, on zna da je ona potpisala tu poruku.
2. Potpis se ne može falsifikovati; samo Alisa zna svoj privatni ključ.
3. Potpis se ne može ponovo koristiti: potpis je funkcija konkretnog dokumenta i ne može da se prenese na drugi dokument.
4. Potpisani dokument se ne može menjati: ukoliko se dokument izmeni na bilo koji način, potpis se više ne može potvrditi pomoću Alisinog javnog ključa.
5. Potpis ne može da se porekne. Bobu ne treba Alisina pomoć da bi proverio njen potpis.

Na slici 2.1 grafički je prikazan proces digitalnog potpisivanja.



Slika 2.1 (grafički prikaz procesa digitalnog potpisivanja poruke)

3. ALGORITMI ZA RAZMENU KLJUČEVA

Algoritmi za razmenu ključa služe za sigurnu i poverljivu komunikaciju između dva ili više subjekta. Poverljivost informacije se temelji na paru ključeva kojim se poruka kriptuje, odnosno dekriptuje. Svaki subjekt u komunikaciji poseduje svoj privatni ključ koji niko osim njega ne zna, a ostali subjekti imaju javni ključ subjekta s kojim komuniciraju. Tajnost poslane poruke se temelji na tome što niko osim primaoca ne može procitati poruku jer nema pravi ključ.

3.1. RSA

RSA algoritam je dobio ime po početnim slovima prezimena triju autora, Rona **R**ivesta, Adia **S**hamira i Lena **A**dlemana, a izumljen je 1977. godine. Algoritam se može koristiti za enkripciju javnim ključem, te kod digitalnog potpisivanja. Njegova sigurnost se temelji na težini faktoriziranja velikih prostih brojeva. Algoritam se sastoji u sledećim koracima:

1. Generiši dva velika prosta broja, **p** i **q**, približno iste veličine i takve da njihov proizvod daje broj određene dužine u bitovima (npr. 1024 ili 2048).
2. Izračunaj **n** = **p** * **q** i **f** = (**p**-1)(**q**-1).
3. Izaberi prost broj **e**, 1 < **e** < f, takav da najveci zajednicki delitelj od (**e**, **f**) iznosi 1 (**e** i **f** su relativno prosti brojevi).
4. Izracunaj tajni eksponent **d**, 1 < **d** < **f**, takav da je **e***d **f** 1 (mod **f**).
5. Javni ključ je (**n**, **e**), a privatni ključ je (**n**, **d**). Vrednosti **p**, **q, f** bi takođe morale biti tajne.

3.1.1. Enkripcija

Pošiljalac A kriptuje poruku na sledeći način:

1. Preuzmi javni ključ primaoca B (n, e).
2. Izračunaj prost broj m < n koji će predstavljati poruku.
3. Izračunaj kriptovani tekst c = me mod n.
4. Pošalji kriptovani tekst primaocu.

3.1.2. Dekripcija

Primaoc dekriptuje poruku na sledeći način:

1. Upotrebi svoj privatni ključ (n, d) da bi mogao izračunati m = cd mod n.
2. Izvrši konvertovanje celobrojne reprezentacije poruke m u njen tekstualni oblik.

3.1.3. Digitalno potpisivanje

Pošiljaoc digitalno potpisuje poruku na sledeći način:

1. Kreira sažetak poruke koja se šalje (npr. SHA-1).
2. Izračuna prost broj m, gde je 0 < m < n - 1, koji ce reprezentovati sažetak.
3. Upotrebi svoj privatni ključ (n, d) da bi izračunao digitalni potpis

   s = md mod n.

4. Pošalji potpis **s** primaocu.

3.1.4. Verificikacija digitalnog potpisa

Primaoc proverava digitalni potpis na sledeći način:

1. Uz pomoć javnog ključa pošiljaoca izračunaj prost broj v = se mod n.
2. Izračunaj sažetak poruke iz broja v.
3. Ponovno izračunaj sažetak primljene poruke.
4. Ako su oba sažetka identična, potpis je valjan.

4. PKI INFRASTRUKTURA

4.1 Uvod

IETF definicija PKI sistema glasi: *PKI je skup hardwer-a, programske opreme, ljudi, pravila i funkcija potrebnih za stvaranje, upravljanje, pohranjivanje, distribuiranje i opozivanje sertifikata baziranih na kriptografiji javnim ključem*[2].

PKI pruža osnovnu sigurnost potrebnu za sigurnu komunikaciju takvu da korisnici, koji se međusobno ne poznaju ili su dosta udaljeni, mogu sigurno komunicirati kroz lanac poverenja. PKI se stoga naziva i hijerarhijom poverenja. Poverenje je građevni blok infrastrukture javnog ključa i ono se u svetu interneta klasifikuje pomoću sledećeg:

- ***Privatnost.*** Ostvaruje se korišćenjem kriptovanja. Poruka se pretvara u drugačiji tekst korišćenjem algoritama enkripcije i samo osobe sa ispravnim ključem za dekriptovanje mogu dobiti originalnu poruku.
- ***Integritet.*** Garantuje da je ono što je primaoc primio upravo ono što je pošiljaoc poslao. Nema gubitka informacija.
- ***Neporecivost.*** Osigurava da korisnik ne može negirati da je poslao poruku ili sudjelovao u transakciji.
- ***Autentifikacija.*** Utvrđuje da je entitet ono što tvrdi da jest. Digitalni sertifikat povezuje identitet s jedinstvenim ključem.

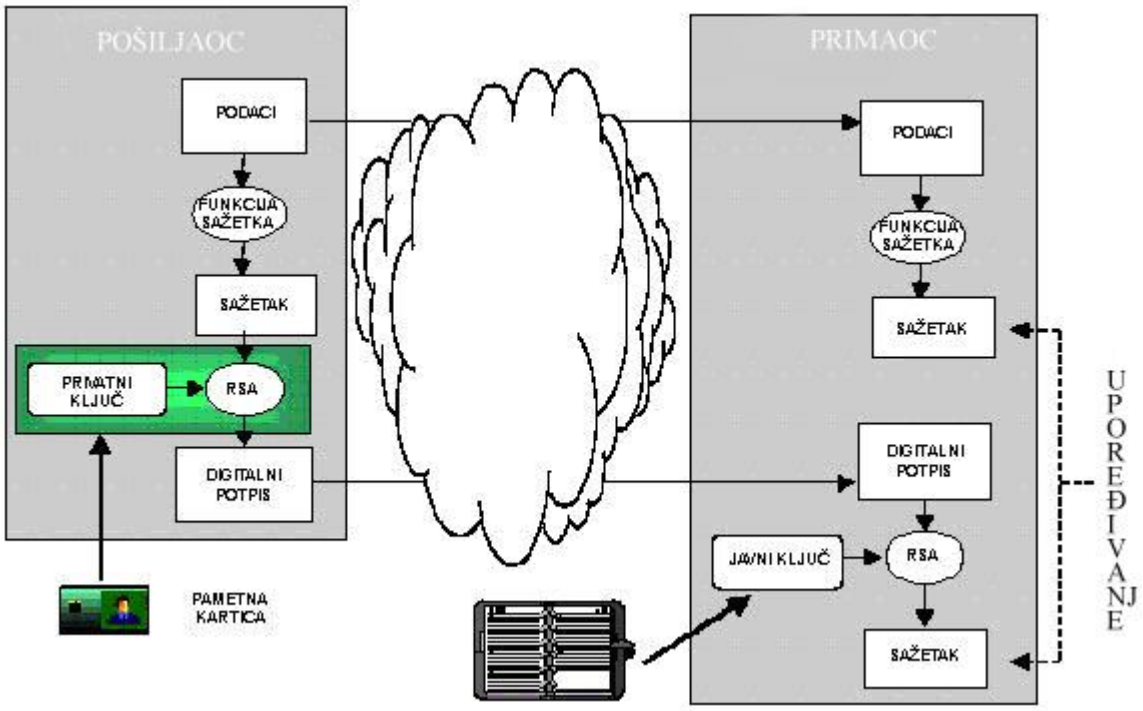
Temeljni deo kriptovanja kod PKI-a čini par ključeva; javni i privatni ključ. Svaki korisnik poseduje svoj jedinstveni par ključeva. Jedan ključ iz para je privatni i njime se koristi samo njegov vlasnik, a drugi je ključ javan i dostupan svima. Ključevi u paru vezani su složenim matematičkim algoritmom (*RSA* algoritam) koji garantuje slijedeće:

- posedovanjem javnog ključa nije moguće otkriti privatni ključ,
- sve što je šifrirano privatnim ključem moguće je dekriptovati samo javnim ključem i obratno.

Kriptografija javnim ključem manifestuje se kroz kriptovanje i digitalno potpisivanje. Poruka se kriptuje pomoću javnog ključa osobe kojoj poruku šaljemo. Na taj način samo primaoc može dekriptovati poruku s odgovarajućim privatnim ključem. Kod digitalnog potpisivanja izračunava se sažetak (engl. *hash*) poruke, koji je jedinstven za svaku poruku, a čime se onemogućava kopiranje već korištenog potpisa. Pošiljalac potpisuje sažetak poruke svojim privatnim ključem, a primalac utvrđuje autentičnost javnim ključem pošiljaoca. Tj. iz primljene poruke se vadi kriptovani sažetak, dekriptuje se javnim ključem pošiljaoca, te se iz primljene izvorne poruke računa novi sažetak i upoređuje sa dobijenim. Ukoliko su isti, dokazuje se identitet pošiljaoca i integritet

[2] *RUSS HOUSLEY, TIM POLK, 2001. Planning for PKI, Wiley Computer Publishing, Canada.*

poruke (funkcija sažetka je jednosmerna, te je nemoguće iz dobijenog sažetka dobiti izvornu.poruku).



Slika 4.1: Razmena poruka i digitalnih potpisa

Privatni ključ nije dostupan svima i stoga je zaštićen lozinkom. U tu se svrhu dodatna sigurnost PKI sustema ostvaruje upotrebom pametnih kartica. Pored pohrane privatnog ključa, digitalni potpis se generiše unutar sigurnog okruženja čipa. Na taj način privatni ključ nikada ne napušta karticu i nije dostupan aplikacijama izvan čipa. Za korištenje pametne kartice potreban je lični broj ili PIN (engl. *Personal Identification Number*), koji poseduje samo korisnik - vlasnik pametne kartice i čitač kartice koji se priključi na računar. Zaštita i čuvanje pametne kartice i PIN-a obaveza su svakog korisnika usluge[3].

Primeri korištenja PKI-a:

- Osiguravanje transakcija između poslovnih partnera i krajnjih klijenata preko javne mreže, kao što je Internet, upotrebom SSL-a,
- Internet bankarstvo,
- Autentifikacija korisnika i kontrola pristupa kroz identifikaciju baziranu na sertifikatima,

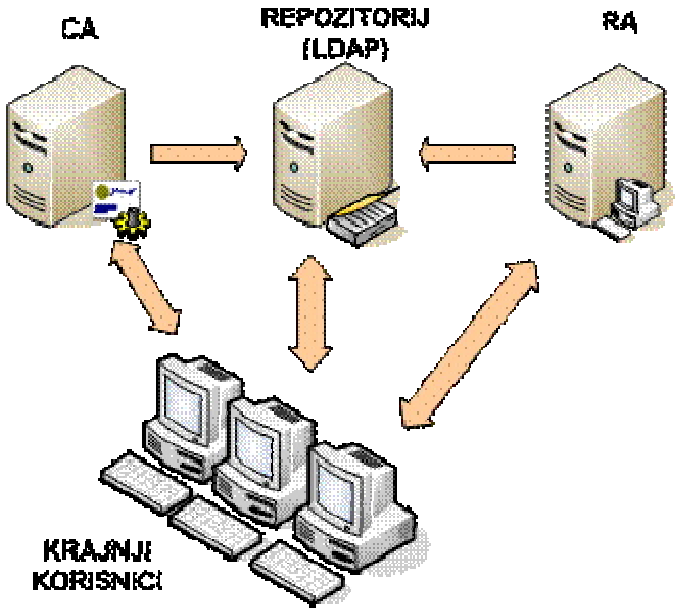
Neporecivost; digitalni sertifikat potvrđuje korisnikov identitet, tako da je posle nemoguće poricati digitalno potpisanu transakciju (npr. kupovanje na Web stranicama).

[3] *ALAN YOUNG, Public-Key Infrastructure: The road to E-Business security*

4.2 PKI komponente

Delovi PKI sistema:

- ***Sertifikacijski centar (CA):*** izdaje i povlači sertifikate, održava informacije o stanju sertifikata, objavljuje valjane sertifikate.
- ***Registracijski centar (RA):*** proverava sadržaj sertifikata za CA.
- ***Repozitorij:*** distribuiše sertifikate i liste opozvnih sertifikata u realnom vremenu.
- ***Krajnji entiteti (korisnici):*** osobe ili oprema.



Slika 4.2: PKI infrastruktura

4.2.1 Sertifikacijski centar

Sertifikacijski centar je treća strana poverenja (engl. *Trusted Third Party*). Javni ključevi se distribuiraju u formi sertifikata javnog ključa. CA je temeljni deo PKI-a s obzirom da je jedina komponenta koja može izdavati sertifikate. CA koji je izdao određeni sertifikat digitalno ga potpisuje (time se povezuje ime subjekta sa javnim ključem). CA je također odgovoran za izdavanje liste opozvanih sertifikata (CRL) osim u slučaju kad je ta funkcija dodijeljena odvojenom izdavatelju CRL-a.

4.2.2 Registracijski centar

RA obrađuje zahteve korisnika za izdavanje sertifikata, registruje korisnike i sarađuje sa CA u poslovima izdavanja sertifikata. Cilj RA je identifikacija korisnika i osiguravanje neporecivosti – da li je digitalni identitet izdan pravoj osobi? RA predstavlja okružemje između krajnjeg korisnika i CA, te obavlja slijedeće funkcije:

- Potvrđuje atribute subjekta koji zahteva sertifikat,
- Potvrđuje da subjekt ima vlasništvo nad registrovanim privatnim ključem,
- Generiše par privatni/javni ključ,
- Upravlja interakcijom sa CA (ili nekoliko njih) kao posrednik krajnjeg entiteta, uključujući obaveštavanje o kompromitiranju ključa i zahtev za oporavak ključa.

4.2.3 Repozitorij

Repozitorij se koristi za distribuiranje sertifikata i statusa izdanih sertifikata. Najčešće upotrebljavana usluga repozitorija je LDAP servera. CA objavljuje sertifikate u repozitorij, a klijenti ih preuzimaju pomoću LDAP bazirane klijentske aplikacije. LDAP implementacija može biti jednostavan LDAP server ili potpuni X.500 proizvod.

Status sertifikata može imati dve forme. Ukoliko je to CRL ili CDP (CRL Distribution Point), koristi se LDAP server. CRL se objavljuje na LDAP server, te klijent preuzima najnoviji CRL. Druga se forma zove On-line Certificate Status Protocol (OCSP). U tom slučaju, keš (engl. *cache*) (forma repozitorija), pohranjuje status sertifikata. Svaki put kada klijentova aplikacija odluči koristiti sertifikat, OCSP će postaviti upit o validnosti sertifikata.

4.2.4 Krajnji entiteti

Krajnji entiteti najčešće predstavljaju krajnje korisnike-ljude. No, to mogu biti i razni uređaji kao što su usmerivači, serveri, aplikacije/servisi ili bilo što drugo što se može identifikovati u imenu subjekta sertifikata javnog ključa. Kao krajnji entiteti mogu biti predstavljeni i davaoci usluga (podrške) vezanih uz PKI. Tako se iz perspektive CA, RA smatra krajnjim entitetom.

4.3 PKI funkcije upravljanja

4.3.1 Registracija

Krajnji entiteti se moraju učlaniti u PKI pre nego što počnu koristiti njegove usluge. Registracija je prvi korak u tom procesu i predstavlja obradu zahteva za sertifikacijom. Ovaj korak je najčešće povezan s inicijalnom proverom identiteta krajnjih entiteta. Proces registracije se može postići direktno sa CA ili preko RA. Isto tako može biti on-line ili off-line (ili kombinacija oba). U registracijskom postupku, u skladu s sertifikacijskom politikom i pravilima o izdavanju sertifikata, potrebno je odrediti u koje će se svrhe sertifikat koristiti i koliki će biti period validnosti sertifikata.

4.3.2 Inicijalizacija

Inicijalizacija sledi nakon registracije. Ovaj se korak najčešće odnosi na povezivanje krajnjeg entiteta s parom ključeva. Generiše se par ključeva što uključuje stvaranje para javni/privatni ključ, koji je povezan s krajnjim entitetom. Ključeve može generisati RA, CA ili neka druga komponenta.

4.3.3 Sertifikacija

Sertifikacija je zaključni deo učlanjivanja krajnjeg entiteta u PKI. Ovaj korak uključuje izdavanje sertifikata javnog ključa od strane CA. Ukoliko je zahtev za sertifikacijom odobren, CA će potpisati podatke i javni ključ iz zahteva. Kada je generisan, sertifikat se daje krajnjem entitetu i/ili objavljuje u repozitoriju.

4.3.4 Oporavak para ključeva

Parovi ključeva se mogu koristiti kao potpora pri stvaranju digitalnog potpisa, za proveru, kriptovanje i dekriptovanje ili oboje. Kada se koriste za kriptovanje/dekriptovanje, važno je omogućiti mehanizam oporavka ključeva potrebnih za dekriptovanje, jer se može dogoditi da "normalan" pristup sadržaju ključeva nije moguć. U tom slučaju nije moguće obnoviti kriptovane podatke. Normalan pristup ključevima dešifriranja može rezultirati zaboravljenom lozinkom, štetom na hardware-u i sl. Oporavak para ključeva omogućava krajnjim entitetima povratak ključeva od autoriziranog sistema za pohranu (engl. *backup*) ključeva.

4.3.5 Ažuriranje ključeva

Sertifikati se izdaju s fiksnim vremenom trajanja (dve do pet godina) i nakon toga ističu. Ažuriranje para ključeva uključuje generisanje novog para ključeva i izdavanje novog sertifikata, a može se pojaviti s unapred zadatim istekom vremena. Tako se legitimni par

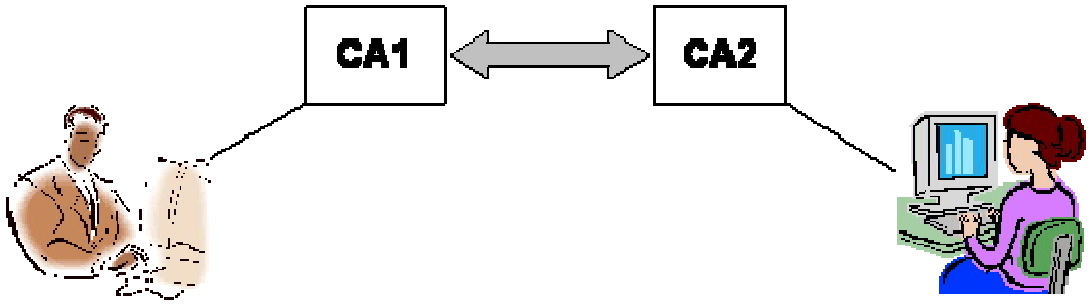
ključeva uvek nalazi u vlasništvu krajnjeg entiteta. Moguće je utemeljiti različite periode valjanosti za ključeve koji se koriste za digitalno potpisivanje i proveru.

4.3.6 Zahtev za opoziv

Sertifikati se izdaju s dugim vremenom trajanja, ali sertifikat može isteći i pre, zbog raznih okolnosti kao što su promena imena, kompromitovanje ključa i sl. Zbog toga je potrebno opozvati sertifikat pre isteka vremena trajanja sertifikata. Zahtev za opoziv može doći i od krajnjeg entiteta (ili RA). Informacije o opozivu moraju biti dostupne od CA koji je izdao sertifikat ili od izdavatelja CRL-a kojemu je CA poverila tu funkciju. Krajnji entiteti moraju proveravati status opoziva svih sertifikata u sertifikacijskoj putanji. To uključuje informacije o opozivu krajnjih entiteta i prelaznih CA.

4.3.7 Međusobno sertifikovanje

Međusobno sertifikovanje se pojavljuje između CA centra. *Cross certifikat* je zapravo sertifikat javnog ključa koji je jedan CA izdao drugom. Drugim rečima, to je sertifikat koji sadrži javni ključ jednog CA, a digitalno ga je potpisao drugi CA. Budući da PKI sistem može imati više CA, moguće je da dva korisnika PKI sistema imaju javne ključeve sertifikovane kod različitih CA, a koji nisu podređeni istom vršnom CA. Međusobno sertifikovanje CA korisnicima omogućava međusobnu autentifikaciju u takvoj situaciji.



Slika 4.3: Međusobna sertifikovanje

5. X.509 SERTIFIKAT JAVNOG KLJUČA

5.1 Uvod

Sertifikat je najvažniji deo celokupnog PKI sistema. Na njemu počiva uzajamno poverenje vlasnika sertifikata i onoga ko sertifikat proverava. Sertifikat nedvosmisleno povezuje osobu uz njen javni ključ - potvrđuje identitet osobe. Sprečava se lažno predstavljanje učesnika u elektronskim transakcijama, kao i otkrivanje tajnih informacija osobama koje za to nisu ovlaštene. U javnom životu sertifikat bi predstavljao ekvivalent ličnoj karti i u tom smislu, u komunikaciji putem interneta, osigurava međusobnu identifikaciju strana koje komuniciraju.

Kako bi podržavao automatsko obrađivanje i da bi se mogao distribuisati putem interneta, sertifikat mora biti digitalni objekat i mora biti kodiran u standardnom formatu. Široko prihvaćen standardni format sertifikata javnog ključa je X.509 sertifikat javnog ključa. X.509 je prvi put objavljen 1988., a u međuvremenu su se pojavile još dve verzije. Sve moderne implementacije PKI-a generišu i obrađuju sertifikate X.509 verzije 3 (X.509 v3), koja je uvela proširenja sertifikata. Proširenja se koriste kada izdavač sertifikata želi uključiti informacije koje nisu podržane osnovnim poljima sertifikata. Nedavno je dovršen i sertifikat X.509 v4 ali on ne menja format sertifikata nego definiše neka nova proširenja.

Vrste sertifikata su:

- ***Sertifikati krajnjih korisnika.*** Ovi sertifikati se izdaju subjektima koji nisu CA. Moraju sadržati pravu informaciju da bi korisnici sertifikata ustanovili da li je javni ključ prikladan za određene aplikacije. Subjekt sertifikata krajnjeg entiteta može biti čovek ili sistem (npr. web server).

- ***CA sertifikati.*** Ovi sertifikati su izdani subjektima koji jesu CA. Oni su deo sertifikacijskog puta. Javni ključevi iz ovih sertifikata se koriste za proveru digitalnog potpisa na sertifikatima i listi opozvanih sertifikata. Moraju sadržavati pravu informaciju za korisnike, za konstruisanje sertifikacijskog puta i lociranje CRL-a. Subjekt CA sertifikata može biti drugi CA unutar istog PKI sistema, CA iz drugog PKI sistema.

- ***Samoizdani sertifikati.*** To je posebna vrsta CA sertifikata kod kojega su izdavač i subjekt isti. Samoizdani sertifikati se koriste za utvrđivanje tačaka poverenja, distribuisanje novih javnih ključeva i modifikovanje skupa sertifikacijskih pravila podržanih u PKI-u.

Digitalni sertifikat se koristi u dve svrhe:

- Javni ključ dostupan je drugim osobama koje mogu koristiti taj ključ kako bi poslali šifrirane poruke, poruke koje može čitati samo vlasnik privatnog ključa. Druge osobe ne mogu čitati te šifrirane poruke.

- Potvrđuje identitet kada se šalju poruke drugim osobama. Drugim rečima, služi kao dokaz da ste vi osoba za koju tvrdite da jeste.

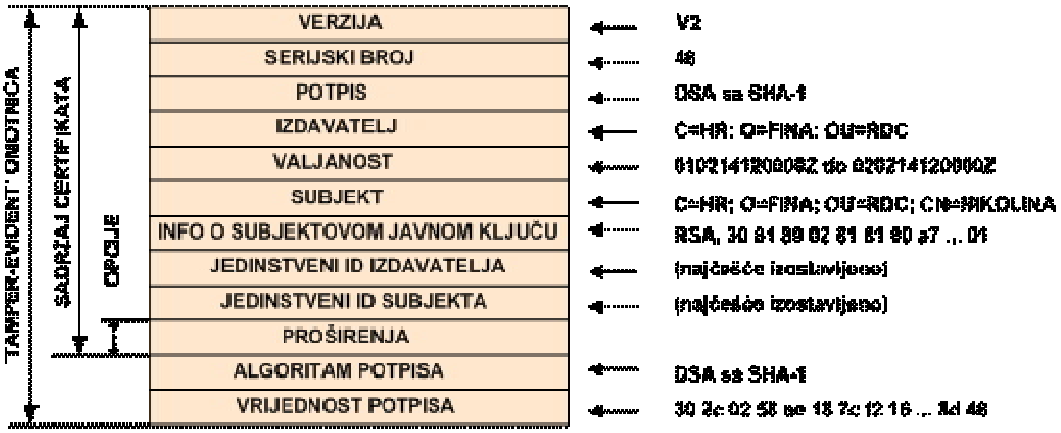
5.2 Komponente sertifikata

X.509 sertifikat možemo promatrati kao tri ugniježđene komponente. Prva komponenta je tzv. *tamper-evident* omotnica na kojoj su jasno vidljivi znaci promena i koju osigurava digitalni potpis. Unutar same omotnice nalazi se osnovni sadržaj koji uključuje informacije obavezne za svaki sertifikat. Osnovni sadržaj može uključivati skup proširenja po izboru. Većina sertifikata, danas generisanih, uključuje sve tri komponente; omotnicu, osnovni sadržaj i proširenja. Slika 5.1 prikazuje strukturu X.509 sertifikata.

5.2.1 Tamper-evident omotnica

"Tamper-evident" omotnicu možemo posmatrati kao prozirnu, plastičnu omotnicu oko sadržaja sertifikata. Poruka se može lako pročitati, ali se ne može menjati bez oštećivanja omotnice. Na ovoj razini sertifikat ima samo tri polja:

- ***Potpisani sertifikat***
- ***Algoritam potpisa***: sadrži identifikator algoritma i identifikuje algoritam digitalnog potpisa koji je izdavač koristio pri potpisivanju sertifikata.
- ***Vrednost potpisa***: sadrži digitalni potpis koji je kodiran kao bit string.



Slika 5.1: Sadržaj digitalnog sertifikata

5.2.2 Osnovni sadržaj sertifikata

Sertifikat koji treba biti potpisan (engl. *to-be-signed certificate*) sadrži sve osnovne informacije sertifikata. Sadrži minimalno šest polja:

- ***serijski broj*** Celi broj koji izdavač dodeljuje svakom sertifikatu. Serijski broj mora biti jedinstven za svaki sertifikat koji je generisao određeni izdavač. Kombinacija imena izdavača i serijskog broja jedinstveno određuje svaki sertifikat.
- ***potpis*** Polje potpisa sadrži identifikator algoritma. To je kopija algoritma potpisa sadržanog u polju algoritma potpisa. Digitalni potpis štiti ovu vrednost.
- ***ime izdavača*** Sadrži X.500 jedinstveno ime (engl. *distinguished name*) izdavača. Ovo polje mora biti popunjeno. Jedinstveno ime je strukturisano tako da podržava hijerarhijski sistem imenovanja. Atributi imenovanja su: C=država (engl. *country*), O=organizacija (engl. *organization*), OU= organizacijska jedinica (engl. *organizational unit*), L=mesto (engl. *locality*), CN=ime (engl. *commom name*) Npr. C=SR; O=DRADA; OU=RDC; CN=MILENA
- ***period validnosti*** Ovo polje ima dve komponente. Datum kada je sertifikat postao validan, ne pre (engl. *notBefore)*, te datum kada sertifikat ističe, ne kasnije (engl. *notAfter*).
- ***javni ključ*** Ovo polje sadrži subjektov javni ključ i identifikator algoritma. Parametri unutar ovog polja sadrže važne informacije o javnom ključu.
- ***subjekt*** Sadrži jedinstveno ime (dn) onoga ko poseduje privatni ključ uparen s javnim ključem sertifikata. Subjekt može biti CA ili krajnji entitet. Zbog sertifikacijske putanje, ime subjekta u CA sertifikatu mora se poklapati s imenom izdavača sertifikata koji sledi.

Postoje četiri izborna polja:

- ***verzija*** Ovo polje opisuje sintaksu sertifikata. Ukoliko se izostavi, sertifikat se kodira po verziji 1 koja ne uključuje jedinstvene identifikatore i proširenja. Verzija 2 uključuje jedinstvene identifikatore bez proširenja, dok verzija 3 uključuje i identifikatore i proširenja.
- ***jedinstveni ID izdavača i subjekta*** Svrha jedinstvenog ID-a izdavača i subjekta je ponovno korišćenje imena izdavača i subjekta nakon nekog vremena. Međutim, preporuka je da se ova polja izostave.

proširenja Ovo se polje pojavljuje samo u verziji 3. Svako proširenje uključuje identifikator proširenja, kritičnu zastavicu i vrednost proširenja.

5.2.3 Proširenja sertifikata

Raniji razvoj PKI-a je pokazao da je osnovni sadržaj sertifikata nedovoljan. Korisnici sertifikata nisu bili u mogućnosti odrediti važne informacije o izdavaču, subjektu i javnom ključu. Proširenja sertifikata dopuštaju da CA uključi informacije koje nisu podržane osnovnim sadržajem sertifikata. Svaka organizacija može definisati privatna proširenja, no većina uslova može biti zadovoljena upotrebom standardnih proširenja, koja su široko podržana komercijalnim proizvodima.

Proširenja imaju tri komponente: *identifikator proširenja*, *kritičnu zastavicu* i *vrednost proširenja*. Identifikator proširenja je OID (engl. *object identifier),* sekvenca celih brojeva koja jedinstveno identifikuje objekat i ukazuje na format i semantiku vrednosti proširenja. Kritična zastavica ukazuje na važnost proširenja. Kada je postavljena, informacija je neophodna za upotrebu sertifikata. Zato se sertifikat ne sme koristiti ukoliko se naiđe na nepoznato kritično proširenje. Isto tako, nepoznato nekritično proširenje može se ignorisati.

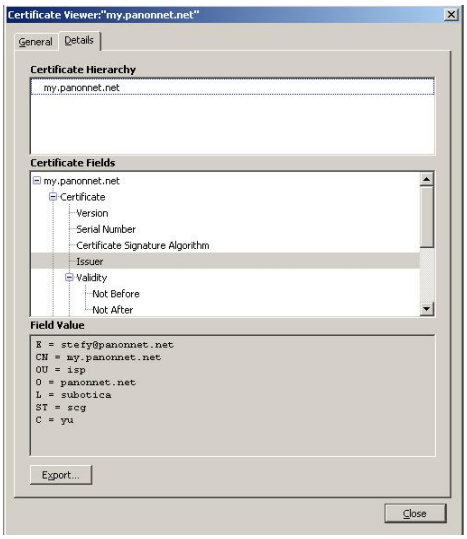
Proširenje tipa subjekta. Nesposobnost određivanja da li sertifikat pripada CA-u ili krajnjem entitetu otežava konstrukciju sertifikacijske putanje. Proširenje osnovnim ograničenjima rešava ovaj problem. Osnovna ograničenja takođe uključuju ograničenje dužine putanje. To izborno polje sadrži celi broj koji pokazuje maksimalan broj sledećih CA sertifikata u sertifikacijskoj putanji. Ako je dužina putanje definisana kao nula, u obzir se uzimaju samo sertifikati krajnjih entiteta.

Proširenje imena. Alternativno proširenje imena, subjektova i izdavačeva alternativna imena su dodatna imena koja bi bila osigurana za CA i subjekt sertifikata. Ta alternativna imena mogu preuzeti bilo koju formu (e-mail adresa i DNS ime u sertifikatima za ljude, URL i DNS imena za računare(naročito servere) te IP adrese i DNS imena za usmrivače i servere).

Atributi ključa. Većina CA centara i krajnjih entiteta imaju više od jednog para javni/privatni ključ i ne koriste isti par ključeva za implementaciju različitih sigurnosnih usluga. Kako bi razlikovali javne ključeve u različitim sertifikatima, CA koristi četiri različita proširenja sertifikata: *namena ključa* (opisuje sigurnosne usluge za koje se javni ključ koristi), *proširena namena ključa* (ukazuje na aplikacije specifične za javni ključ), *identifikator ključa autoriteta* (ako CA ima nekoliko ključeva za potpisivanje sertifikata, ovo proširenje daje ispravan ključ za proveru određenog potpisa) i *identifikator subjektovog ključa* (ako subjekt ima više sertifikata, omogućuje brzo identifikovanje sertifikata s ključem koji nas zanima).

Informacije o pravilima. Postoje dva standardna proširenja ovog tipa: proširenje *Certificate Policies* i *Policy Mapping*. *Certificate Policies* se odnosi na pravila pod kojima je sertifikat izdat, kod CA sertifikata se odnosi na pravilo pod kojim CA radi. *Policy mapping* se koristi kod CA sertifikata za prevođenje informacija o pravilima između dve domena pravila (između dva CA centra).

Dodatne informacije. Kako bi proverio sertifikat, korisnik sertifikata mora dobiti CA sertifikat sa kojeg uzima javni ključ, te pomoću njega proverava potpis sertifikata. Nakon toga konsusltuje najnoviji CRL radi utvrđivanja validnosti sertifikata. Kako bi dobio sertifikate ili CRL iz direktorija, korisnik mora znati samo jedinstveno ime iz sertifikata. Ako se pristupa nekom drugom repozitoriju, korisnik mora znati jedinstveno ime, adresu repozitorija koji sadrži informaciju i protokol pristupa. Svaki CA u sertifikacijskoj putanji može koristiti drugi repozitorij. Postoji nekoliko standardnih proširenja koja korisnicima daju dodatne informacije: CRL tačka distribucije, najsvežiji CRL, pristup informacijama centra, pristup informacijama subjekta i drugo. CRL tačka distribucuje govori korisniku gde i kako dobiti CRL potreban za utvrđivanje da li je sertifikat opozvan. Ovo proširenje sadrži jednu ili više distribucijskih tačaka i svaka opisuje lokaciju odgovarajućeg CRL-a.



Slika 5.2: Digitalni sertifikatu web browseru

6. PAMETNE KARTICE (Smart Card)

Pametna kartica (engl. smart card) je plastična kartica koja u sebi ima ugrađen mikročip (engl. integrated circuit chip), a dimenzije su joj iste kao i kod kreditnih kartica. Glavne funkcije kartice su autentifikacija, pohranjivanje podataka, vrednovanje (engl. validation), te mehanizam samozaključavanja. Otporna je na vanjske napade i ne ovisi o potencijalno ranjivim vanjskim resursima. Baš zbog tih svojstava se pametne kartice često koriste u različitim aplikacijama koje zahtevaju visok nivo sigurnosne zaštite i autentifikacije. Na primer, pametne kartice mogu služiti za identifikaciju vlasnika kartice, kao zdravstvena kartica, ili kao kreditna/debitna bankovna kartica koja omogućava financijske transakcije. Sve te aplikacije koriste osetljive podatke spremljene na kartici kao što su biometrijski podaci o vlasniku, medicinski podaci, kriptografski ključevi za autentifikaciju itd. Neprestanim unapređivanjem sigurnosnih svojstava pametna kartica postaje vrlo pouzdan medij za pohranjivanje najosjetljivijih podataka čime se smanjuje broj argumenata za sumnju u njenu isplativost.

Postojanje niza načina ugrožavanja sigurnosti pametnih kartica ne znači da pametna kartica nije sigurna. Napadi na sigurnosne sisteme bilo koje vrste nisu novost. Nigde ne postoji savršena sigurnost, pa tako ni kod pametnih kartica. Glavna odlika o tome da li je sistem siguran ili ne, ovisi o ispunjavanju sigurnosnih zahteva sistema. Nadalje, većina napada na pametne kartice se mogu klasifikovati u tri klase, što znaci da je cena probijanja sistema puno veća nego njegova vrednost, ili je potreban vrlo dugi vremenski period da bi se probili kriptografski algoritmi. Kako tehnologija vrlo brzo napreduje, proizvođači su primorani stalno ažurirati sigurnosne sisteme svojih proizvoda.

Postoji nekoliko vrsta pametnih kartica, od običnih memorijskih kartica koje nemaju mogućnost procesiranja podataka, do kartica s procesorom i kriptografskim procesorom koje proširuju područje upotrebe kartice, ali i nivo sigurnosti. Najnovija vrsta kartica su beskontaktne kartice koje, osim izbegavanja problema s kontaktima na prihvatnim uredajima (terminalima), donose i niz prednosti kojima postižu veću popularnost kod krajnjih korisnika.

Sigurnosni mehanizmi su osnovna sredstva kojima se postiže vrlo visok nivo sigurnosti kod pametnih kartica. Oni se temelje na vrlo kompleksnim matematičkim proračunima koje vrlo teško mogu izračunavati standardni računari bez specijalnog hardware-a. U mehanizme se ubrajaju algoritmi kriptovanja, funkcije sažimanja i algoritmi za razmenu ključeva.

6.1. Šta je to pametna kartica

Pametnu karticu (eng. *Smart Card*) možemo opisati kao plastičnu karticu s instaliranim integrisanim kolom koji sadrži mikroprocesor. Pametne kartice ne sadrže u sebi nikakav uređaj za napajanje već napajanje dobijaju od uređaja za prihvat kartica CAD (eng. *Card Acceptance Device*). Za razliku od kartica s magnetnom trakom, pametne kartice pružaju puno veći nivo sigurnosti za podatke koji su na kartici, a takođe se jedna kartica može koristiti za više različitih usluga. Jednim od najvažnijih svojstava pametnih kartica smatra se činjenica da podaci koji se nalaze na njima mogu biti zaštićeni od neovlaštenog pristupa i menjanja podataka, a također omogućuju kriptovanje i dekriptovanje podataka.

6.2. Vrste pametnih kartica

Prema karakteristikama pametne kartice možemo podeliti na:

1. memorijske kartice
2. kontaktne mikroprocesorske kartice
3. bezkontaktne mikroprocesorske kartice
4. kombinovane mikroprocesorske kartice

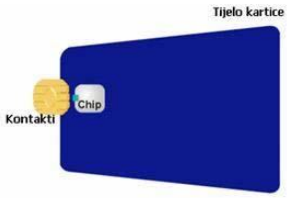
6.2.1. Memorijske kartice

Memorijske kartice su kartice koje na integrisanom kolu imaju samo memoriju i pristupnu logiku. Slično kao i kartice sa magnetnom trakom, memorijske mogu samo spremati podatke. U starijim memorijskim karticama nema ugrađene sigurne kontrolne logike. Zbog toga neovlašten pristup podacima na kartici ne može biti sprečen. Današnje memorijske kartice imaju u integrisanom kolu ugrađenu sigurnu logiku koja omogućuje da pristup sigurnoj zoni imaju samo autorizirani korisnici.

6.2.2. Kontaktne mikroprocesorske kartice

Kontaktne mikroprocesorske kartice imaju ugrađen mikroprocesor na integrisanom kolu, a prenos podataka može biti samo onda kada je kartica postavljena u uredaj za čitanje kartica CAD (eng. *Card Acceptance Device*) u svrhu čega kartica ima osam zlatnih kontakata na sebi kojima se električki spaja s CAD uređajem.

Vanjski izgled ovih kartica zasnovan je na ISO-7816-1 i ISO-7816-2 standardima koji određuju dimenzije kartice, smeštaj kontakata na kartici i njihov razmeštaj. Struktura kontaktne pametne kartice prikazana je na slici 6-1.



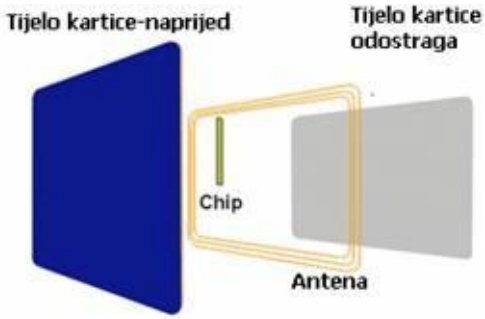
Slika 6-.1. Kontaktna pametna kartica

Jedno od najvažnijih obilježja mikroprocesorskih pametnih kartica je sigurnost. Kontaktne mikroprocesorske pametne kartice su uglavnom prilagođene za sigurni prenos podataka. Ako se korisnik ne autentifikuje uspešno, neće se moći pristupiti podacima koji se nalaze na kartici. Također, ako je kartica izgubljena, podaci na kartici neće biti izloženi neovlaštenom pristupu ako su ti podaci prikladno sačuvani na kartici. Pošto je pametna kartica kao mali računar ona može sigurno obraditi unutrašnje podatke te proslediti rezultat na izlazno okruženje.

6.2.3. Bezkontaktne mikroprocesorske kartice

Kontaktne pametne kartice nisu pogodne za sve tipove aplikacija, posebno kod masovnih transakcija i kada je potrebno da se podaci s kartice pročitaju u vrlo kratkom vremenu. Korišćenjem elektromagnetskih talasa bezkontaktne pametne kartice mogu slati podatke prema CAD uređaju s udaljenosti od nekoliko cm do 50 cm.

Kao i kontaktne kartice, ove također nemaju nikakvo unutrašnje napajanje već koriste tehnologiju koja omogućava CAD uređajima da napaja i komunicira s karticom, bez fizičkog kontakta s karticom, preko elektromagnetskih talasa. Struktura bezkontaktne kartice prikazana je na slici 6-2.



Slika 6-2 Bezkontaktna pametna kartica

Bezkontaktne pametne kartice su pogodne kod velikog broja pristupa kartici i prenosa podataka, ali nisu pogodne kad je potreban prenos velike količine podataka od kartice prema CAD uređaju. Također jedan od nedostataka bezkontaktnih kartica je da može doći do transakcije između CAD uređaja i kartice bez znanja korisnika.

6.2.4. Kombinovane mikroprocesorske kartice

Kontaktne i bezkontaktne kartice koriste dva različita protokola za komuniciranje s CAD uređajem. Obe kartice imaju svoje prednosti i nedostatke. Kontaktne pametne kartice imaju veći nivo sigurnosti, dok bezkontaktne kartice imaju bolji i jednostavniji interfejs za transakcijsko okruženje. U pokušaju da se korisnicima omoguće prednosti obe kartica razvijena su dva načina:

1. Pametna kartica ima dva interfejsa za komunikaciju, jedno je kontaktno, a drugo je bezkontaktno tako da kartica može komunicirati s CAD uređajem preko kontakata ili preko elektromagnetskih talasa.

2. Kontaktna pametna kartica se postavi u poseban uređaj koji ima napajanje za karticu i antenu tako da može komunicirati preko elektromagnetskih talasa s CAD uređajem.

6.3 Standardi

Kroz istoriju razvoja pametnih kartica uspostavljene su različite norme za rešavanje problema kompatibilnosti kartica različitih proizvođača. Prva standard bila je ISO-7816 koju je izdala međunarodna organizacija za norme (eng. *International Standard Organisation* - ISO) 1987. godine. Pre toga proizvođači kartica i CAD uređaja imali su svoje norme kartice i uređaje koji nisu bili kompatibilni s karticama i uređajima ostalih proizvođača. Sa ISO normom pametne kartice različitih proizvođača mogu komunicirati istim protokolom. Fizičke karakteristike i dimenzije su također jednake za sve kartice i one su također propisane normom, također je propisan položaj kontakata na kartici, značenje pojedinog kontakta te protokol i sadržaj poruka kod razmjene poruka. Ove norme osiguravaju da kartice proizvedene od jednog proizvođača mogu biti prihvaćene od CAD uređaja drugog proizvođača. Danas tri najvažnije norme su:

1. ISO-7816 standard
2. EMV (Europay, Mastercard, Visa) standard
3. GSM (Global Standard for Mobile Communications) standard

6.3.1. ISO-7816 standard

ISO-7816 je međunarodni standard za pametne kartice. ISO-7816 standard sastoji se od više delova od kojih su najbitniji:

1. ISO-7816-1
2. ISO-7816-2
3. ISO-7816-3
4. ISO-7816-4
5. ISO-7816-5
6. ISO-7816-6

6.3.1.1. ISO-7816-1

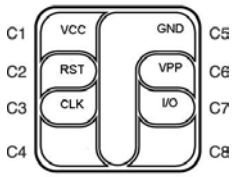
Opisuje fizičke karakteristike pametnih kartica njihove dimenzije, te otpornost kartice na vanjske smetnje.

Dimenzije kartice po ovom standardu iznose 85.60mm × 53.98mm × 0.80mm. Ovaj standard propisuje i granice izlaganja kartice na razne vanjske uticaje kao što su

rendgenske zrake, UV svetlo, elektromagnetska polja, elektrostatička polja, savijanje i rastezanje kartice.

6.3.1.2. ISO-7816-2

Određuje dimenzije i lokaciju kontakata na pametnoj kartici te funkciju i poziciju određenog kontakta. Kontaktna pločica na površini pametne kartice koja je spojena sa izvodima integrisanog kola (čipa) postavljenog u pametnu karticu ima osam kontakata koji su označeni od C1 do C8. Međutim nije svih osam kontakata povezano s integrisanim kolom i u današnje vreme ti kontakti još nemaju svoju specijalnu funkciju već su ti kontakti rezerviasani za buduću upotrebu. Na slici 6-3 prikazan je raspored kontakata, a u tabeli 6-1. oznake i opis pojedinog kontakta.



Slika 6-3 Raspored kontakata pametne kartice

Kontakt	Oznaka	Opis
C1	Vcc	Kontakt za napajanje integrisanog kola – obično se kreće od +3v do +5v
C2	RST	Kontakt za reset preko kojeg se pokreće reset protokol
C3	CLK	Kontakt preko kojeg dovodimo takt signal IC-u
Kontakt	**Oznaka**	**Opis**
C4	RFU	Rezervisano za buduću upotrebu (Reserved for Future Use)
C5	GND	Kontakt el. Mase
C6	Vpp	Napajanje koje se u prošlosti koristilo za programiranje EEPROM memorije
C7	I/O	Ulazno/izlazni port, veza kartice s vanjskim svijetom, omogućuje half-duplex komunikaciju između čitača i CAD uređaja
C8	RFU	Rezervisano za buduću upotrebu (Reserved for Future Use)

Tabela 6-1. Oznake i opis kontakata pametne kartice

6.3.1.3. ISO-7816-3

Opisuje električne signale i protokole prenošenja signala kod pametnih kartica. Većina ovog standarda važna je za proizvođače CAD uređaja i za programere koji žele uspostaviti komunikaciju s pametnom karticom na vrlo niskom nivou, signalnom nivou.

Ovom normom deklarišu se razine napona, jačine struja te perioda trajanja signala, te protokoli (T=0, T=1) komuniciranja između CAD uređaja i kartice.

6.3.1.4. ISO-7816-4

Određuje celu logičku strukturu pametne kartice te naredbe i protokole za komuniciranje, a određuje:

- sadržaj poruka, naredbi i odgovora, usmerenih od CAD uređaja prema kartici i obrnuto
- strukturu i sadržaj bajtova koje šalje pametna kartica kao odgovor na reset
- strukturu datoteka i podataka
- metode pristupa datotekama i podacima na kartici
- metode za sigurno komuniciranje
- metode pristupa algoritmima koji su procesuirani na kartici. Ovaj standard ne opisuje te algoritme

6.3.1.5. ISO-7816-5

Opisuje brojevni sistem za prikaz identifikatora aplikacija AID-a (eng. *Application Identifiers*). Svaki AID ima dva dela. Prvi deo je identifikator registrovanog davaoca aplikacije RID (eng. *Registered Application Provider Identifier*) koji se sastoji od 5 bajta i oni su jedinstveni (eng. *unique*). Drugi dio AID je promenljive dužine do 11 bajtova koji omogućuje RID-u identifikaciju specifične aplikacije.

6.3.1.6. ISO-7816-6

Standard opisuje međuindustrijske elemente podataka, fizički transport uređaja i prenos podataka, odgovor na reset ATR (eng. *Answer To Reset*). Ova specifikacija dozvoljava dva protokola T=0 i T=1. Kartica može podržavati bilo koji, ali ne oba.

6.3.2. EMV standard

EMV standard jedan je od važnijih standarda za pametne kartice kojeg su formirale vodeće financijske kompanije u kartičnom poslovanju Europay, Mastercard i Visa. Standard pokriva elektromehaničke karakteristike, protokole, podatke te instrukcije koje spaja sa bankovnim transakcijama. Cilj EMV specifikacije je da svi sistemi za plaćanje dele iste POS-terminale (eng. *Point of Sales*), kao što to čine magnetne kartice i aplikacije za njih. Zbog toga što će se uskoro sve bankovne magnetne kartice menjati pametnim karticama, ovaj standard osigurava da nove bankovne pametne kartice budu kompatibilne s bankovnim transakcijskim sistemima. Zahvaljujući ovom standardu, sve bankovno orijentirane pametne kartice bit će kompatibilne jedna s drugom isto kao i prijašnje (dosadašnje) magnetne kartice. Fleksibilnošću EMV standarda, bankama je dopušteno da dodaju svoje opcije i specijalne potrebe u platni sistem pametnih kartica.

6.4 Životni ciklus pametne kartice

U svakoj pametnoj kartici postoji operatinni sistem kojeg određuju podaci kao što su identifikacijski broj proizvođača (*eng. Manufacturer identification number*), tip komponente, serijski broj, informacije o profilu itd.Najvažniji su sigurnosni ključevi koje sistemsko područje sadrži, kao što su ključ proizvođača (*eng. Manufacturer/fabrication key*) i ključ personalizacije. Sve te informacije moraju biti sakrivene i nedostupne drugim subjektima. One se najčešće nalaze unutar posebnih datoteka kojima mogu pristupiti samo ovlašćene osobe.

Dakle, od proizvođača, do davaoca aplikacija, pa sve do vlasnika kartice, proizvodnja pametne kartice je podeljena na nekoliko različitih faza. U svakoj od faza su ograničenja kod transfera i pristupa podacima različita, tj. povećavaju se u svrhu zaštite različitih područja na pametnoj kartici. Postoji pet različitih faza kod uobičajenog životnog ciklusa pametne kartice koje su predstavljene u narednom delu rada.

6.4.1 Faza proizvodnje

Ova faza je određena proizvođačima čipova. U ovoj fazi se izrađuje integrisano kolo od silikona koje se kasnije i testira. Ključ proizvođača se dodaje da bi se zaštitio čip od neovlaštene modifikacije sve do ugradnje u plastičnu karticu. Ključ je jedinstven za svaki čip, a dobijen je derivacijom glavnog ključa proizvođača. Ostali podaci proizvođača se zapisuju na kraju ove faze. Nakon toga se integrisano kolo sprema da dostavu proizvođačima zaštićen ključem proizvođača.

6.4.2 Pripremna faza personalizacije

Ova faza je određena proizvođačima kartica, a sastoji se od ugradje integrisanog kola u karticu. Izrađuju se veze između čipa i štampanih veza nakon čega je kartica spremna za testiranje. Dodatna sigurnost se postiže zamenom ključa proizvođača s ključem personalizacije. Nakon toga se upisuje personalizacijska brava Vper (*eng. Personalization lock*) koja označava kraj zapisivanja podataka koji se odnose na pripremnu fazu personalizacije, i koja štiti čip od daljnje modifikacije personalizacijskog ključa. Personalizacijskom bravom se onemogućava izvođenje instrukcija koje pristupaju fizičkoj memoriji, pa je pristup kartici ograničen samo na korištenje logičkog memorijskog adresiranja. Brava je predstavljenja zapisom u datoteci na kartici, odnosno njenom zaglavlju. Logičko adresiranje čuva sistem i štiti rezervisana područja na čipu od neovlaštene modifikacije.

6.4.3 Faza personalizacije

Ova faza je usko vezana s izdavačima kartica, a sastoji se od završavanja strukture logičkih podataka. Sadržaj datoteka podataka i podataka vezanih za aplikaciju se zapisuju na karticu. Osim tih podataka pohranjuju se još informacije o identitetu korisnika kartice, PIN i deblokacijski PIN. Na kraju se zapisuje i brava korištenja Vutil

(*engl. utilisation lock*) u datoteku kojom je završena faza zapisivanja podataka koji se odnose na fazu personalizacije. Zapisivanjem brave korištenja je kartica spremna za korištenje.

6.4.4. Faza korištenja

U ovoj fazi korisnik koristi karticu. Aktiviraju se aplikacijski sistemi, kontrole za pristup logičkoj datoteci i ostale kontrole koje su potrebne. Pristup informacijama spremljenim na kartici je ograničen sigurnosnom politikom aplikacije koja koristi pametnu karticu.

6.4.5. Završna faza (faza poništavanja kartice)

Postoje dva načina kako kartica dolazi u ovo stanje. Prvi se inicira od strane aplikacije koja zapisuje bravu poništavanja (*engl. invalidation lock*) u individualnu datoteku ili glavnu datoteku. Sve operacije, uključujuci pisanje i ažuriranje, se onemogućavaju od strane operativnog sistema. Ostaju samo operacije čitanja koje su predviđene za svrhe analiziranja. Nepostojanje ostalih operacija nakon zapisivanje brave poništavanja kartica postaje velikim delom beskorisna jer više ne može spremati nikakve nove podatke na sebi (na primer nove sertifikate nakon isteka njihove validnosti ili kriptografske ključeve nakon isteka perioda njihove validnosti), ali ni koristiti svoje kriptografske mogućnosti (generirsanje para ključeva, vrednovanja digitalnog potpisa i ostalih kriptografskih operacija). Drugi način dovođenja kartice u ovo stanje se postiže ireverzibilnim blokiranjem pristupa kartici od strane kontrolnog sistema. To nastaje zbog blokiranja PIN-a i blokiranja deblokacijskog PIN-a (obe vrste PIN-a su tri puta krivo unesene). U ovom slucaju je onemogućena i operacija čitanja podataka s kartice.

6.5 Primena pametnih kartica

S velikim rastom internet tehnologije i elektronske trgovine, pametne kartice su postale sve šire prihvaćene i korištene kao kartice sa sigurno sačuvanim sadržajem. U ovom poglavlju opisat će se neke aplikacije gde se koristi tehnologija pametnih kartica. Ove aplikacije možemo podeliti u pet glavnih kategorija:

- Elektronsko plaćanje
- Računarska sigurnost
- Telekomunikacije

6.5.1. Elektronsko plaćanje

Kod elektrosnkog plaćanja pametna kartica služi za spremanje novčanih sredstava korisnika u obliku elektronskog novca ili elektronskih kovanica ili žetona (eng. *tokens*). Elektronski novac može se koristiti za manje kupovine kod kojeg nebi bilo potrebe za autorizaciju preko PIN broja (eng. *Personal Identification Number*). Novčana sredstva na

kartici pokriva korisnikova banka preko njegovog računa u banci ili se ta sredstva pokrivaju na neki drugi način. Kada korisnik plaća pametnom karticom elektronski novac se skida s pametne kartice i šalje na račun davaoca usluge. Korisnik može napuniti svoju karticu elektronskim novcem u banci u bilo koje vreme.

Od 1994. godine došlo je do značajnijeg razvoja aplikacija za rad s elektronskim novcem u Evropi koje su se proširile i izvan Evrope. U vezi pametnih kartica i elektronskog novca bilo je razvijeno nekoliko globalnih projekata kao npr.: ProtonCard firme Banksys, VisaCash od firme Visa International te Mondex firme Mastercard. Ovi projekti bili su širom prihvaćeni u trgovinama širom sveta.

6.5.2. Računarska sigurnost

Pametna kartica našla je veliku primenu u računarskoj sigurnosti gde se koristi za autentifikaciju korisnika u sistemu, za čuvanje digitalnih sertifikata i kriptografskih ključeva (npr. javni i tajni ključ za asimetrične algoritme) te za kriptovanje i dekriptovanje poruka.

Jedan od primera korištenja pametne kartice u računarskoj sigurnosti je korišćenje pametne kartice u PKI sistemu gdje pametna kartica čuva javni i tajni ključ korisnika koji se tada koriste za kriptovanje i dekriptovanje poruka, kreiranje digitalnog potpisa i za čuvanje digitalnih certifikata.

6.5.3. Telekomunikacije

Telekomunikacije su jedan od najvećih korisnika pametnih kartica. Od 1988. godine pametne kartice postale su jedne od najvažnijih komponenata u telefonskim stanicama (mobilni telefoni). Podaci mreže, informacije pretplatnika i svi kritični podaci mobilne mreže snešteni su unutar kartice. S ovom karticom, pretplatnik može obaviti poziv s bilo kojeg mobilnog telefona. Takođe svaki poziv preko telefona može biti kriptovan što osigurava privatnost.

Iz dana u dan primena pametnih kartica je sve veća. Pojavljuju se nove oblasti gde se koriste pametne kartice kako bi se olakšali i ubrzali određeni procesi. U ovom radu prikazana je funkcija pametnih kartica u računarskoj sigurnosti. Pre svega pametna kartica nam može poslužiti kao izvrstan indentifikacioni metod na internetu koji bi povećao sigurnost današnjih poznatih metoda za identifikaciju korisnika na raznim online servisima.

6.6. Komunikacijski model

Komunikacija između CAD uređaja (eng. *Card Acceptance Device*) i pametne kartice odvija se preko jedne linije i zbog toga se kartica i CAD uređaj moraju izmenjivati u slanju poruka, dok jedna strana šalje poruku druga prima i obrnuto. Ovakav protokol zove se poludupleks (eng. *Half-Duplex*) protokol. Pametna kartica i CAD uređaj imaju gospodar-sluga (eng. *master–slave*) odnos gde se kartica ponaša kao sluga, a CAD uređaj gospodar. Ovaj odnos podrazumeva da kartica nikada ne šalje poruku bez vanjskog zahteva tj. komunikaciju uvek pokreće CAD uređaj.

6.6.1. Protokoli za komuniciranje

Standard ISO-7816-3 specificira ukupno 16 protokola. Protokoli su označeni s 'T=' plus sekvencijalni broj od 0 do 15. U praksi nas zanimaju 2 protokola, a to su T=0 i T=1 protokol.

6.6.1.1. T=0 protokol

T=0 protokol je prvi standardizovan protokol za pametne kartice, a pravila kod kreiranja bila su minimalno korištenje memorije i maksimalna jednostavnost. Protokol je bajtovno orentisan i svaki TPDU (eng. *Transmission Protocol Data Unit*) sastoji se dva dela:

1. Zaglavlje
2. Podaci

Zaglavlje sadrži pet bajtova kako je prikazano u tabeli 6-2.

Ime	Opis
CLA	određuje klasu instrukcije
INS	određuje određenu instrukciju
P1	određuje adresu
P2	određuje adresu
P3	određuje broj bajtova prenesenih prema ili od pametne kartice

Tabela 6-2. Elementi zaglavlja TPDU naredbe za T=0 protokol

Mehanizam provere greške je jednostavan i koristi proveru pariteta pojedinog bajta. Kod detekcije greške zahteva se ponovno slanje naredbe.

Kartica odgovara CAD uređaju sa zaglavljem koje sadrži četiri bajta kako je prikazano u tabeli 6-3.

Ime	**Opis**
ACK	Označava prijem [CLA,INS] naredbe
NULL	Koristi se za kontrolu protoka I/O kanala
SW1	Status informacija za nadzor tekuće naredbe
SW2	Status informacija za nadzor tekuće naredbe(opcionalno)

Tabela 6-3. Elementi zaglavlja odgovora TPDU naredbe kod T=0 protokola

6.6.1.2. T=1 protokol

T=1 je blokovski orentisan protokol. Ovo znači da se dobro definisan skup podataka ili blok prenosi kao jedna jedinica podataka između pametne kartice i CAD uređaja. U blok može biti ugrađena APDU (eng. *Application Protocol Data Unit*) naredba namenjenja nekoj specifičnoj naredbi. Ova mogućnost omogućava vrlo dobro uslojavanje između sloja povezivanja i aplikacijskog sloja. Stavljanje informacije u blok zahteva da blok bude prenesen bez greške jer inače protokol može biti jednostavno izgubljen. Detekcija i ispravak grešaka kod T=1 protokola je zbog toga kompleksnije nego kod T=0 protokola.

Detekcija grešaka u T=1 protokolu se obavlja uzdužnom redundancijom znakova LRC (eng. *Longitudinal Redundancy Character*), koji je u suštini puno kompleksniji od provere pariteta koja se obavlja u T=0 protokolu, ili se obavlja korištenjem cikličke redundancije znakova CRC (eng, C*yclic Redundancy Check*). Ove operacije provere grešaka garantuju detektovanje bilo koje jednobitne greške u prenosnom bloku. Kada se detektuje greška koja je nastala tokom prenosa signalizira se pošiljaocu da ponovi slanje bloka u kojem je detektirana greška.

U T=1 protokolu postoje tri osnovna različita tipa blokova koji imaju jednaku strukturu, kako je prikazano na slici 6-4., ali različitu svrhu:

1. Informacijski blok – Ovaj blok služi za prenošenje podataka između aplikacijskog programa u kartici i aplikacijskog programa na strani servera.
2. Blok potvrde i primanja – Ovaj blok služi za prenošenje negativne ili pozitivne potvrde o tome da li je blok prenesen bez greške preko informacijskog kanala.
3. Nadzorni blok – Ovaj blok služi za prenos kontrolnih informacija između pametne kartice i CAD uređaja.

Uvodno polje Prologue field			Informacijsko polje Information field	Zaključno polje Epilogue field
Adresa čvora Node address NAD	Kontrolni bajt protokola Protocol control byte PCB	Duljina Length LEN	APDU	Detekcija pogreške Error detection LRC/CRC
1 bajt	1 bajt	1 bajt	0 do 254 bajta	1 do 2 bajta

Slika 6-4. Struktura T=1 prijenosnog bloka

Svaki T=1 blok kao što je vidljivo iz slike 4.4. ima tri polja:

- **Uvodno polje (eng. *prologue field*)** – Obavezno polje bloka koje ima dužinu od 3 bajta. Sadrži ova tri elementa:
  - **NAD** – Adresa čvora (eng. *node address*)
  - **PCB** – Kontrolni bajt protokola (eng. *Protocol control byte*)
  - **LEN** – dužina
- **Informacijsko polje (eng. *information field*)** – opciono polje bloka koje može biti dužine do 254 bajta.
- **Zaključno polje (eng. *epilogue field*)** – Obavezno polje bloka koje je dužine od jednog do dva bajta.

NAD element se koristi za identifikaciju adrese izvora i odredišta bloka. Ova mogućnost adresiranja je od velike koristi kad se T=1 protokol koristi za podržavanje višestrukih logičkih veza između pametne kartice i višestrukih aplikacijskih spojnih tačaka na strani CAD uređaja. NAD ima dva pod-elementa:

- SAD (eng. *Source Address*) – Adresa izvora označava se sa tri najmanje značajna bita NAD bajta.
- DAD (eng. *Destination Address*) – Adresa odredišta označava se od petog do sedmog bita NAD bajta

U situacijama gde se ne koriste višestruki logički kanali NAD element postavljen je na nulu. Druga dva bita, koji se ne koriste za SAD i DAD podelemente, služe za prenos informacija namenjenih kontrolisanju V_{PP} napajanja(napajanje za programiranje EEPROM memorije).

PCB element koristi se za identifikovanje tipa bloka(informacijskog, bloka potvrde i primanja te nadzornog bloka). Dva najznačajnija bita PCB bajta koriste se za označavanje različitih tipova:

- Najznačajniji bit postavljen u 0 označava da je blok informacijski
- Dva najznačajnija bita postavljena u 1 označavaju da je blok nadzorni
- Najznačajniji bit postavljen u 1, a sledeći postavljen u 0 označava da je reč o bloku potvrde i primanja

6.7 APDU (eng. Application Protocol Data Units)

APDU (eng. *Application Protocol Data Units*) je internacionalno standardizovana jedinica podataka za razmenu podataka između pametne kartice i CAD uređaja. U prenosnom sloju se APDU naziva TPDU koji je u aplikacijskom sloju podeljen na dva APDU dela koji su: APDU naredba i APDU odgovor. APDU možemo opisati kućicama gde svaka kućica sadrži naredbu koju šalje CAD uređaj pametnoj kartici ili pametna kartica šalje odgovor CAD uređaju.

6.7.1. APDU naredba

Svaka APDU naredba ima dva elementa zaglavlje i telo. Veličina zaglavlja je fiksna i iznosi 4 bajta, dok veličina tela varira ovisno o količini podataka koji se šalju. Slika 6-5. prikazuje strukturu APDU naredbe.

APDU naredba						
Zaglavlje (obavezno)				Tijelo (opcionalno)		
CLA	INS	P1	P2	Lc	Podaci	Le

Slika 6-5. Struktura APDU naredbe

• **CLA** (1 bajt) – Ovo obavezno polje identifikuje aplikacijsko-specifičnu klasu instrukcije. Odgovarajuće CLA vrednosti definisane su ISO-7816-4 standardom. Tabela 6.4. prikazuje vrednosti CLA polja i njihovu klasu instrukcija kojoj pripadaju

CLA vrednost	**Klasa instrukcije**
0x0n, 0x1n	ISO-7816-4 instrukcije kartice, npr. za pristup datotekama i sigurne operacije
0x20 do 0x7F	Rezervisano
0x8n do 0x9n	ISO-7816-4 format koji se koristi za lične aplikacijski-specifične instrukcije

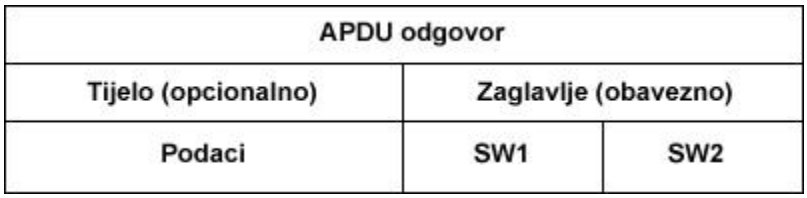
0xAn	Aplikacijsko-specifične instrukcije
0xB0 do 0xCF	ISO-7816-4 format za korištenje aplikacijsko specifičnih instrukcija
0xD0 do 0xFE	Aplikacijsko-specifične instrukcije
FF	Rezervisano za izbor vrste protokola

Tabela 6-4. Vrijednosti i klasa CLA polja

• **INS** (1 bajt) – Ovo obavezno polje predstavlja specifičnu instrukciju sa klasom instrukcije definisanom u CLA polju. ISO-7816-4 standard definiše osnovne instrukcije za pristup podacima na pametnoj kartici. Korisnik može definisati svoje aplikacijsko-specifične instrukcije jedino kad koristi odgovarajuću vrednost CLA polja.
• **P1** (1 bajt) – Ovo polje definiše prvi parametar instrukcije. Može se koristiti za kvalifikaciju INS polja ili za ulazni podatak.
• **P2** (1 bajt) – Ovo polje definiše drugi parametar instrukcije. Može se koristiti za kvalifikaciju INS polja ili za ulazni podatak.
• **Lc** (1 bajt) – Ovo opciono polje sadrži broj bajtova koji se nalaze u polju podataka
• **Podaci** (promenljiva veličina, Lc bajtova) – Opciono polje koje sadrži podatke naredbe.
• **Le** (1 bajt) – Opciono polje koje definiše maksimalni broj bajtova u polju podataka očekivanog odgovora.

6.7.2. APDU odgovor

Struktura APDU odgovora koji vraća kartica je mnogo jednostavnija i prikazana je na slici 6-6.

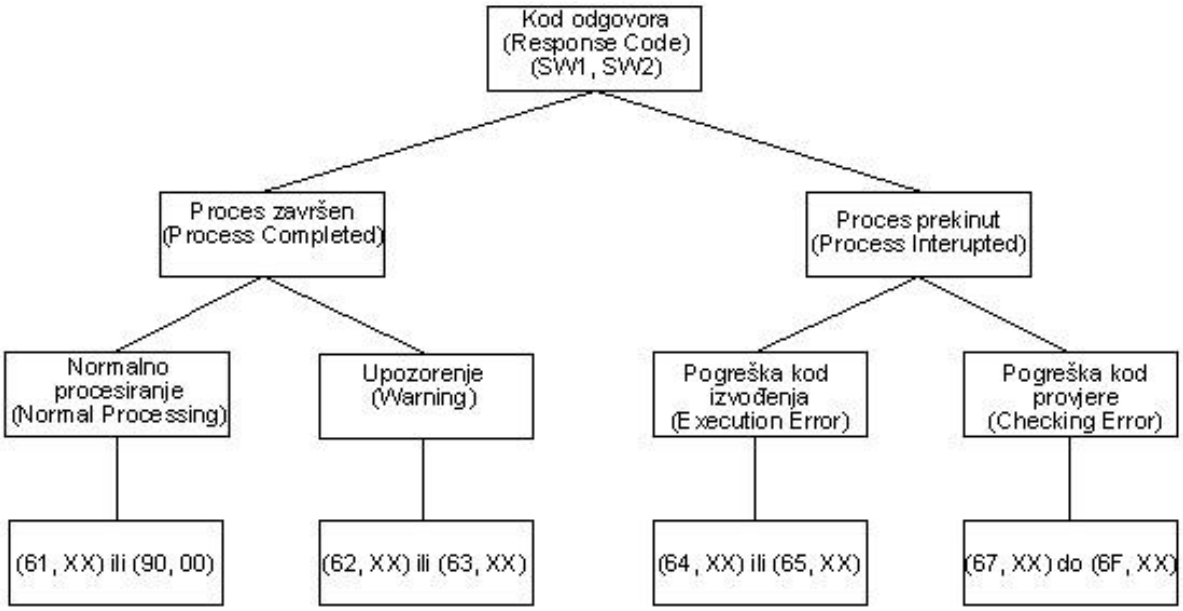


Slika 6-6. Struktura APDU odgovora

Takođe kao i APDU naredba i APDU odgovor ima opciona i obavezna polja.

• **Podaci** (promenljiva veličina, određene Le poljem u APDU naredbi) – Opciono polje koje sadrži podatke koje vraća aplikacija na kartici.
• **SW1** (1 bajt) – Obavezno polje sadrži statusnu reč 1.
• **SW2** (1 bajt) – Obavezno polje sadrži statusnu reč 2.

Vrednosti statusne reči definisane su ISO-7816-4 standardom. Moguće vrednosti statusne reči SW1 i SW2 mogu se prikazati blok dijagramom prema načinu izvođenja procesa kako je prikazano na slici 6-7.



Slika 6-7. Blok dijagram statusnih riječi

7. JAVA CARD TEHNOLOGIJA

Java Card tehnologija predstavlja najmanju Java platformu koja je namenjena memorijski ograničenim uređajima poput pametnih kartica, te omogućava da se programi pisani u Java programskom jeziku mogu izvršavati na takvim uređajima. Ova tehnologija je ustvari podskup Java tehnologije koji omogućava kreiranje aplikacija za pametne kartice tzv. applete koji se izvršavaju na kartici.

Tipični Java Card uređaj ima mikroprocesor koji može biti 8-bitni, 16-bitni ili 32-bitni te sadrži 1KB RAM memorije i više te najmanje 16KB trajne memorije (EEPROM). Većina kartica također ima i posebne koprocesore te RAM memoriju za kriptografske algoritme.

Java Card tehnologiju možemo podijeliti na tri dijela:

1. Java Card virtualni uredjaj (eng. *Java Card Virtual Machine, JCVM*
2. Java Card izvršna okolina (eng. *Java Card Runtime Environment, JCRE*)
3. Java Card programski interfejs(eng. *Application Programming Interface, API)*

7.1. Java Card Virtual Machine

Java Card virtualni uredjaj *(Java Card Virtual Machine*) specifikacija definira podskup Java programskog jezika i Java kompatibilnog virtualnog uredjaja (eng. *Virtual Machine, VM*) za pametne kartice uključujući prikaz binarnih podataka i formate datoteka te JCVM set instrukcija.

VM za Java Card platformu implementiran je u dva diela. Prvi deo je razvojni alat, tipično prikazan kao Java Card Converter tool, koji učitava, verifikuje te dalje priprema Java klase u Java paketu za izvršavanje na pametnoj kartici. Izlaz iz konvertera je konvertirana applet datoteka ili CAP (eng. *Converted Applet*) datoteka koja se učitava na karticu. CAP datoteka sadrži sve klase iz Java paketa u binarnom obliku za izvršavanje na kartici. Drugi dio VM-a je implementiran na kartici i on interpretira bajtkod, rukovodi klasama i objektima itd.

JCVM podržava samo ograničen podskup Java programskog jezika, ali sadrži mnoge poznate mogućnosti uključujući objekte, nasleđivanje, pakete, dinamičko kreiranje objekata, virtualne metode, interfejs i izuzetke. JCVM specifikacija izbacuje potporu za mnoge elemente Java jezika koji bi koristili previše memorije na ionako ograničenoj memoriji pametne kartice. Tabela 7.1. prikazuje pregled ograničenja Java Card programskog jezika dok tabela 7.2. prikazuje pregled ograničenja Java Card virtualnog uredjaja (JCVM).

Mogućnosti jezika	Dinamičko učitavanje klasa, sigurnosni manager, kloniranje objekata te neki aspekti kontrole pristupa paketima nisu podržani.
Ključne riječi (eng. *keywords*)	native, synchronized, transient, volatile, strictfp nisu podržane
Tipovi podataka	Nisu podržani char, double, float i long tipovi te nisu podržana višedimenzionalna polja. Podrška za int tip je opcionalna.
Klase i interfejsi (eng. *classes and interfaces*)	Klase i interfejsi jezgre Java API-a (java.io, java.lang, java.util) nisu podržani sa izuzetkom klase Object i Throwable, ali većina njihovih metoda nije dostupna.
Izuzeci (eng. *Exceptions*)	Neke podklase izuzetaka (Exception) i grešaka (Error) su izostavljene jer ne mogu proizlaziti iz Java Card platforme.

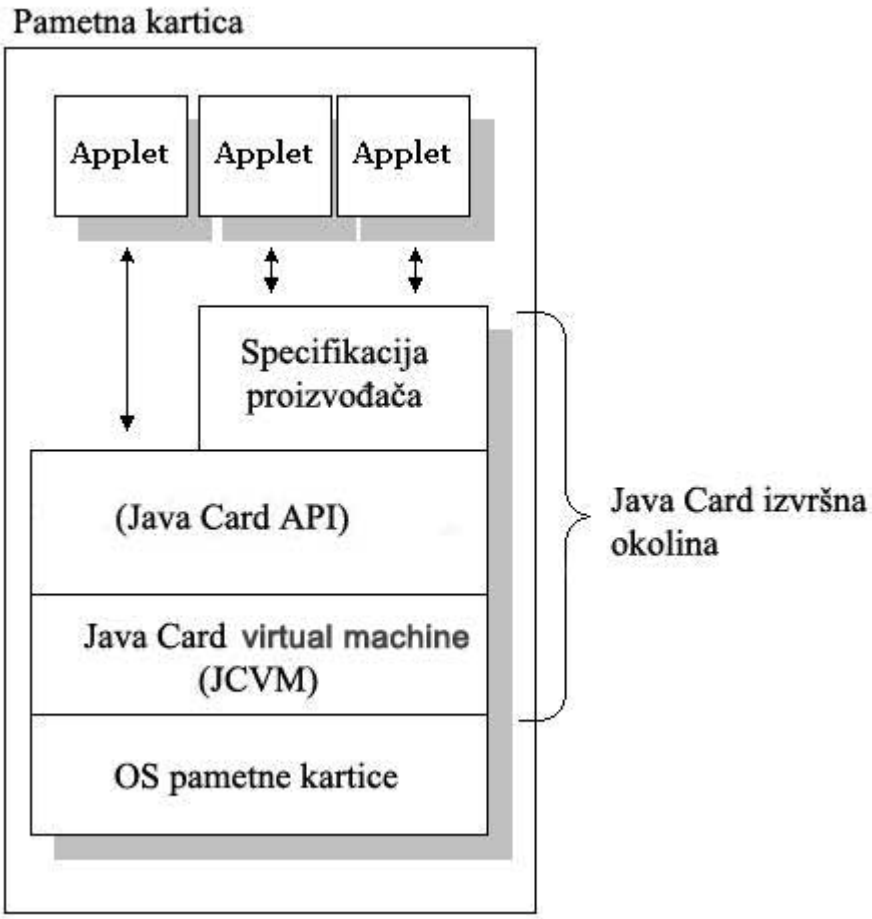
Tablica 7.1. Ograničenja Java Card programskog jezika

Paketi	Paket može sadržavati do 128 drugih paketa.
	Puno ime paketa je ograničeno na 255 bajta.
	Paket može imati do 255 razreda.
Razredi	Razred može direktno ili indirektno implementirati do 15 interfejsa.
	Interfejs može naslijediti do 14 interfejsa.
	Paket može imati do 26 statičkih metoda ako sadrži applet ili 255 ako ne sadrži.
	Razred može implementirati do 128 javnih ili zaštićenih metoda.

Tablica 7.2. Pregled ograničenja Java Card virtualnog uredjaja

7.2. Java Card izvršna okolina

Java Card izvršna okolina (eng. *Java Card Runtime Environment, JCRE*) specifikacija definiše životni ciklus Java Card virtualnog uredjaja (JCVM), životni ciklus appleta, selektiranje appleta, transakcije te deljenje i postojanost objekta. JCRE osigurava interfejs neovisno o platformi tj. neovisnost o operativnom sistemu kartice. JCRE se sastoji od Java Card virtualnog uređaja (JCVM), Java Card programskog interfejsa (Java Card API) te bilo koje specifikacije isporučitelja ili proizvođača pametne kartice. Slika 7.1. prikazuje arhitekturu Java pametne kartice.



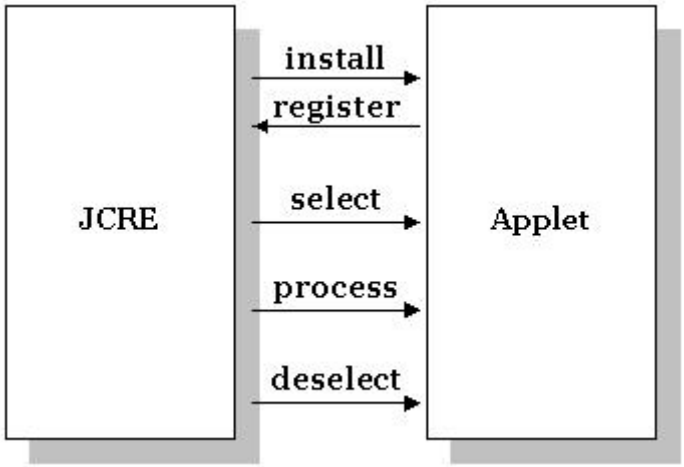
Slika 7.1. Arhitektura java pametne kartice

7.2.1. Životni ciklus Java Card virtualnog uređaja

Životni ciklus Java Card virtualnog uredjaja (JCVM) podudara se s životnim ciklusom pametne kartice. Započinje nakon što je kartica proizvedena i testirana, pre nego je izdata korisniku, završava kada je odbačena ili uništena. JCVM se ne zaustavlja kad kartica nema napajanja zbog toga što je stanje JCVM-a spremljeno u trajnu memoriju pametne kartice. Nakon što je JCVM pokrenut sve interakcije s karticom su u principu kontrolirane jednim od appleta na pametnoj kartici. Kad se napajanje ukloni sa kartice svi podaci koji su bili u RAM memoriji su izgubljeni, međutim bilo koje stanje spremljeno u trajnu memoriju je sačuvano.

7.2.2. Životni ciklus Java Card appleta

Svaki aplet na kartici je jedinstveno identificiran aplikacijskim identfikacijskim brojem (eng. *Application ID, AID*). AID je sekvenca od 5 do 16 bajtova. Svaki aplet mora nasleđivati Applet abstraktnu klasu koji definiše metode koje koristi Java Card izvršna okolina (JCRE) za kontrolisanje životnog ciklusa apleta kako je prikazano na slici 7.2.



Slika 7.2. Metode životnog ciklusa Java Card appleta

Životni ciklus Java Card appleta započinje kad je applet učitan na karticu i kad JCRE poziva statičku metodu appleta Applet.install() i kad se applet registruje sa JCRE-om pozivanjem Applet.register() metode. Jednom kad je applet instaliran i registrovan on se nalazi u neselektiranom stanju, ali dostupan za selekciju i APDU procesiranje. U neselektiranom stanju applet je neaktivan. Applet dolazi u selektirano stanje za APDU procesiranje kada poslužiteljska aplikacija zahtijeva od JCRE-a da selektuje određeni applet na kartici. Da JCRE obavesti applet da je selektiran, od poslužiteljske aplikacije, JCRE poziva select() metodu. Applet tipično izvodi odgovarajuću inicijalizaciju kao pripremu za APDU procesiranje. Jednom kad je selekcija obavljena JCRE preusmerava dolazeće APDU naredbe selektiranom appletu pozivajući metodu process(). Deselekcija appleta se izvodi kad poslužiteljska aplikacija zahteva od JCRE-a da selektira neki drugi applet. JCRE obaveštava aktivni applet da će biti deselektiran pozivajući deselect() metodu, koja tipično izvodi bilo koje logičko čišćenje memorije i vraća applet u neaktivno tj. neselektirano stanje.

7.3. Java Card programsko okruženje

Java Card programski interfejs (API) definiše mali podskup standardnog Java programskog interfejsa. Ovo je najmanji programski interfejs Java platforme.

U zamenu za mali podskup sličnih klasa sa standardnom jezgrom Java programskog interfejsa, Java Card API definiše svoj skup klasa koje podržavaju Java Card aplikacije. Nalaze se u sledećim klasama:

- java.IO definiše jednu klasu izuzetaka, IOException klasu, u svrhu da se kompletira hijerarhija poziva udaljenih procedura (eng. *Remote Method Invocation*, *RMI*) iznimaka.

- java.lang definiše Object i Throwable klase koje imaju velik nedostatak metoda naspram klasa u standardnom Java jeziku. Ovaj paket takođe definiše nekoliko klasa izuzetaka, osnovnu klasu za izuzetke Exception te različite izuzetke kod izvođenja (eng. *Runtime Exception*).
- java.rmi definiše Remote interfejs i klasu RemoteException. Podrška za udaljeni poziv procedura (eng. *Remote Method Invocation, RMI*) je uključena da se pojednostavi migracija s uređajima koji koriste Java Card tehnologiju.
- javacard.framework definiše interfejs, klase i izuzetke koji sačinjavaju jezgru Java Card interfejs. Definiše važne koncepte poput ličnog identifikacionog broja (eng. *Personal Identification Number, PIN*), APDU razred, Java Card Applet klasu, te Java Card sistem (eng. Java Card System) JCSystem klasa. Takođe definiše razne konstante ISO7816 norme u razne specifične izuzetke.
- javacard.security definiše klase i interfejs za Java Card sigurnosni okvir. Java Card specifikacija definiše snažnan sigurnosni interfejs koji sadrži različite tipove javnih i privatnih ključeva te kriptografske algoritme, algoritme za računanje sažetka poruka i algoritme za kreiranje digitalnog potpisa.
- javacardx.crypto je dodatni paket koji definiše KeyEncryption interfejs I klasu Cipher. Cipher klasa je osnovna abstraktna klasa koju moraju naslediti svi kriptografski algoritmi.

8. OPENCARD TEHNOLOGIJA

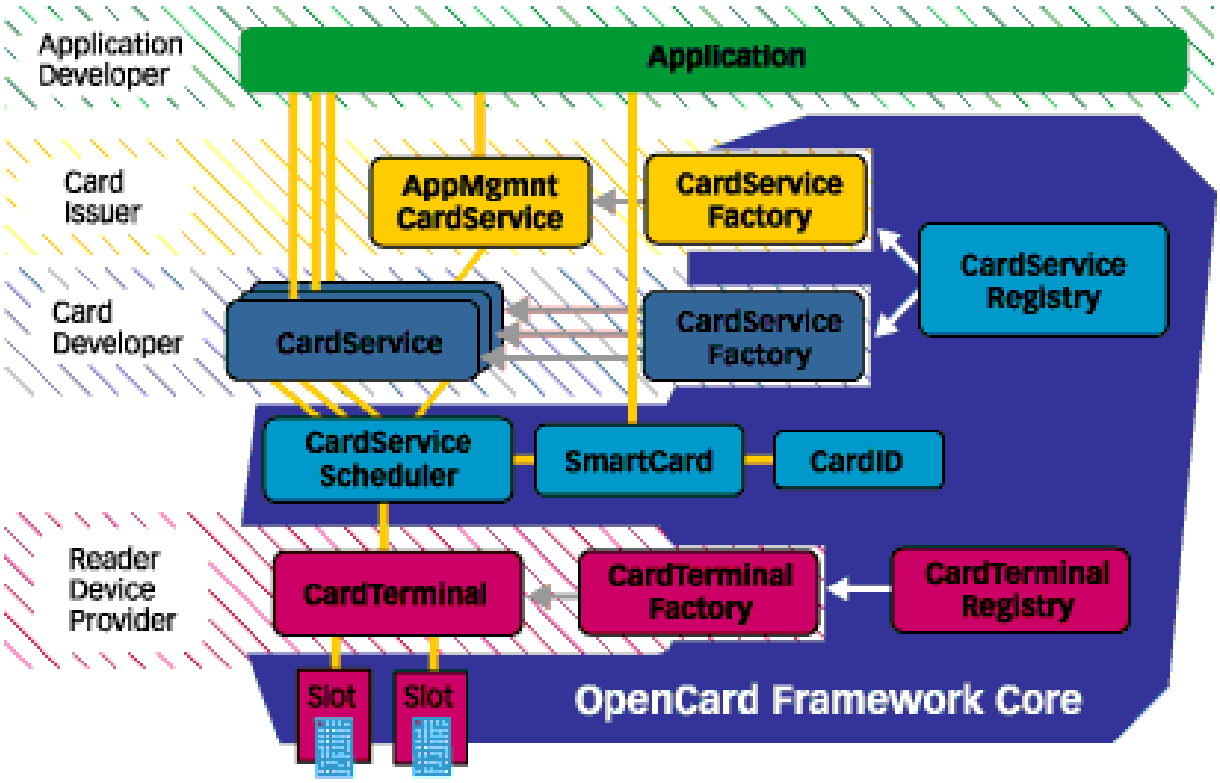
OpenCard tehnologija je otvorena arhitektura napravljena u programskom jeziku Java koja daje programski interfejs koji omogućava komunikaciju s bilo kojom karticom i bilo kojim čitačem neovisno o proizvođaču. Velika prednost OpenCard tehnologije je u tome da se kod implementacije sistema za rad s pametnim karticama ne treba poznavati sklopove pametne kartice i čitača. Deo na višem sloju može se implementirati neovisno o nižim slojevima koji implementiraju komunikaciju s pametnom karticom.

8.1. Arhitektura OpenCard tehnologije

OpenCard implementacija ima skup klasa i paketa koji su napravljeni da olakšaju i ubrzaju izradu sistema koji su temeljeni na pametnim karticama. Ova arhitektura omogućava da:

- davalac aplikacijskih i servisnih usluga
- davalac kartičnog operativnog sistema
- izdavač kartice
- proizvođač čitača kartice

rade zajedno.



Slika 8.1. Arhitektura OpenCard tehnologije (izvor. Gemalto.com)

Davalac aplikacijskih usluga izrađuje kartične i terminalske aplikacije koje vidi korisnik pametne kartice. Terminal koristi interfejs i kartične usluge OpenCard arhitekture kako bi komunicirao s kartičnom aplikacijom. Izdavač kartičnog operativnog sistema izrađuje operativni sistem koji upravlja pametnom karticom i izvodi programe na kartici. Izdavač kartice inicijalizira i izdaje pametnu karticu. Proizvođač čitača stvara uređaj koji komunicira s pametnom karticom. Slika 8.1. prikazuje arhitekturu OpenCard tehnologije.

Jezgra arhitekture OpenCard tehnologije sadrži dva glavna dela:

1. Sloj terminala (klasa CardTerminal)
2. Servisni sloj (klasa CardService)

8.1.1. Sloj terminala

Sloj terminala (CardTerminal) sadrži klase i interfejs koji programeru aplikacije omogućavaju pristup terminalima i njihovim priključcima (eng. *slots*). Na tržištu postoji velik broj čitača pametnih kartica s različitim funkcijama. Vrlo jednostavni čitači samo daju osnovne funkcije ulaza i izlaza (eng. *input-output*) preko jednog priključka, dok su drugi čitači sofisticiraniji i omogućuju više priključaka te mogućnost unosa PIN broja.

Klase sloja terminala imaju dvostruku ulogu. Jedna od tih uloga je da omogućava pristup fizičkom čitaču pametne kartice i kartici umetnutoj u taj čitač. Ove funkcije su enkapsulirane u CardTerminal klasu preko priključaka (eng. *slots*) te preko CardID klase.

CardTerminal klasa je abstraktna klasa koju nasleđuju konkretne klase za određene terminale (čitači pametnih kartice). Pristup umetnutoj kartici dobiva se preko SlotChannel objekta. CardTerminal klasa osigurava da je u svakom trenutku vremena instaliran maksimalno jedan SlotChannel objekt. To znači kad neki objekt dobije SlotChannel objekt nijedan drugi objekt ne može pristupiti pametnoj kartici tako dugo dok taj SlotChannel objekt nije oslobođen.

Terminalski sloj takođe ima mehanizam za dodavanje i odstranjivanje terminala. CardTerminalFactory razred i CardTerminalRegistry objekt implementiraju te funkcionalnosti. Svaki proizvođač čitača pametnih kartica koji podržava OpenCard tehnologiju implementira CardTerminalFactory koji zna za određenu porodicu čitača, te implementira odgovarajuću CardTerminal klasu. CardTerminalRegistry objekt sadrži zapise o instaliranim čitačima pametnih kartica, te ima metode za prijavljivanje i odjavljivanje terminala.

CardTerminal klasa takođe generiše događaje i obaveštava ostali deo OpenCard tehnologije kad je pametna kartica umetnuta ili izvađena iz terminala.

8.1.2. Servisni sloj

Servisni sloj omogućava OpenCard tehnologiji rad s različitim operativnim sistemima pametne kartice i raznim funkcijama koje one mogu ponuditi. CardService interfejs predstavlja programski interfejs visoke razine prema aplikaciji i brine se o svim komunikacijskim detaljima APDU veze. Za stvaranje objekta CardService zadužena je klasa CardServiceFactory. Podatke o svim registrovanim CardServiceFactory objektima sadrži klasa CardServiceRegistry. CardService komunicira s pametnom karticom preko klase CardServiceScheduler koji služi za sinhronizaciju konkurentnih pristupa pametnoj kartici.

8.2. Rad sa OpenCard tehnologijom

Pre korištenja funkcija iz OpenCard tehnologije mora se obaviti inicijalizacija. Kako bi se obavila inicijalizacija mora se prvo pozvati metoda start() iz klase SmartCard, takođe bi trebalo i pozvati metodu shutdown() prije nego se završi aplikacija. Nakon inicijalizacije treba čekati da se pametna kartica umetne te da se tada može dobiti odgovarajući CardService objekt. Kod čekanja na pametnu karticu postoje dva načina. Jedan način je blokirajući i tada se rad programa zaustavlja tako dugo dok se kartica ne umetne, dok je drugi način neblokirajući te se program izvršava, a dok se kartica umetne generiše se određeni događaj isto se događa kad se izvadi pametna kartica. Nakon što je

pametna kartica umetnuta pristupa se zahtevu servisa (CardService) za rad s pametnom karticom.

Kako bi neka klasa mogla koristiti neblokirajući način čekanja na pametnu karticu ona mora implementirati CTListener interfejs koji sadrži dve metode cardInserted() i cardRemoved(). Takođe treba pre korištenja te funkcionalnosti inicijalizirati generator događaja EventGenerator. Slika 8-2. prikazuje inicijalizaciju OpenCard tehnologije

.

```
//inicijalizacija EventManagera te OpenCard tehnologije
public void start() {
      EventGenerator.getGenerator().addCTListener(this);
      try {
            SmartCard.start();
            EventGenerator.getGenerator().
                  createEventsForPresentCards(this);
      }
      catch (Exception e) {
            …
      }
}
```

Slika 8-2. Inicijalizacija OpenCard tehnologije

Nakon što inicijaliziramo OpenCard tehnologiju i nakon što je umetnuta pametna kartica pristupa se pozivanju CardService objekta. Instanca CardService objekta je PassThruCardService objekt koji omogućava komuniciranje s pametnom karticom na nivou APDU naredbi. Slika 8-3. prikazuje kako se inicijalizuje PassThruCardService.

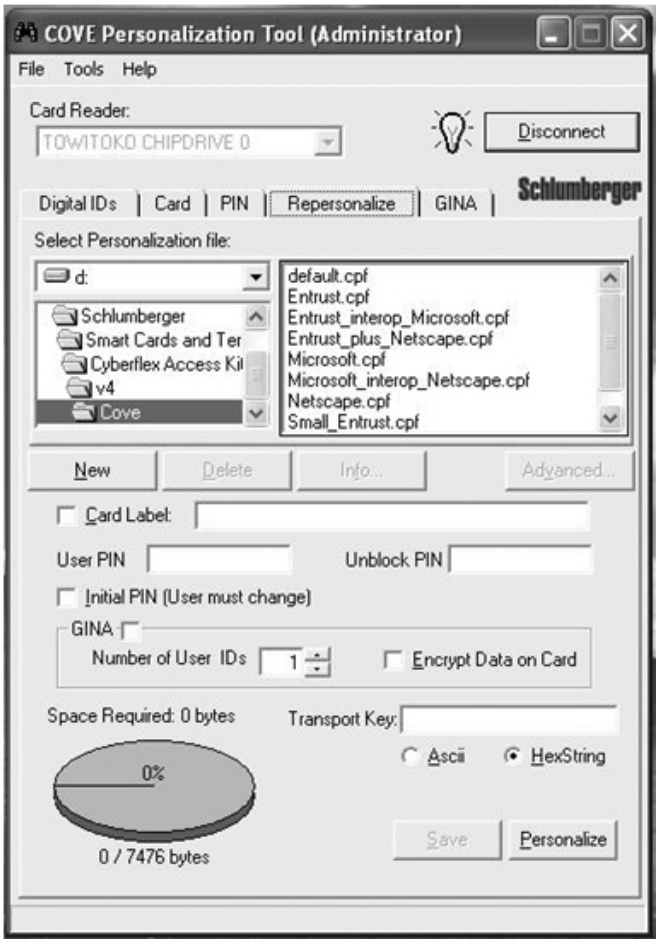
```
…
CardRequest cardRequest = new CardRequest(CardRequest.ANYCARD,
      null, PassThruCardService.class);
SmartCard smartCard = SmartCard.getSmartCard(p_event, cardRequest);
PassThruCardService ptcs = smartCard.getCardService(
      PassThruCardService.class, true);
…
```

Slika 8-3. Pozivanje PassThruCardService klase

9. PRAKTIČAN DEO

9.1 Personalizacija kartice

Programski paket koji predstavlja vezni model (*eng. middleware*) između operativnog sistema , pametne kartice i čitača pametnih kartica pruža mogućnost personalizacije pametne kartice. Personalizacija čini jednu fazu pripreme kartice za njenu upotrebu, opisano u delu 6.4.3 ovoga rada. Za ovu aplikaciju upotrebljen je vezni modul Cyberflex Aceess Software Development Kit release 4.4. proizvođača Schlumberger. Vezni modul sadrži i preglednik COVE (*engl. Cryprographic Object Viewer and Editor*) koji omogućava personalizaciju kartice. Pomoću njega je moguće pregledati sadržaj memorije kartice, promeniti PIN i PIN za deblokiranje, repersonalizovati karicu itd. Sledeća slika (slika 9.1) prikazuje grafički korisnički interfejs preko kojeg ovlaštena osoba može menjati podatke na kartici tj. izvršiti personalizaciju prazne kartice ili repersonalizovati već prethodno personalizovanu karticu.



Slika 9.1 *Korisnično grafičko sučelje za personalizaciju pametne kartice*

Na slici je moguće videti da se nudi nekoliko datoteka kojima je moguće automatsko personalizovanje, moguće je uneti oznaku (eng. label) kartice, PIN i PIN za deblokiranje zaključane kartice, broj korisnika kartice, način čuvanja potataka na kartici (u kriptovanom obliku ili ne) kao i izbor transportnog ključa koji predstavlja autorizacijksi ključ za aplikaciju. On se koristi kod novih kartica da bi se omogućio pristup kartici pre same personalizacije.

9.2 Java Applet za digitalno potpisivanje upotrebom pametnih kartica.

Digitalno potpisivanje dokumenata pomoću pametnih kartica direktno iz web pretraživača moguće je izvesti java applet-om. U ovom delu prezentovaću jedan ovakav java applet baziran na Sun PKCS#11 Provider-u, a kriptografski provajder je deo JSE 5.0.

Digitalno potpisivanje zahteva od korisnika da poseduje digitalni sertifikat sa odgovarajućim privatnim ključem. Ovi sertifikati se izdaju od strane autorizovanog tela i mogu biti sačuvani u fajlu ili na pametnoj kartici.

Pametne kartice se koriste da bi se obezbedila veća zaštita osetljivih informacija kao što su privatni ključevi i sertifikati tako što sprečavaju direktan pristum i mogućnost kopiranja . Privatni ključeva koji se nalaze na pametnoj kartici mogu se koristiti za enkriptovanje i potpisivanje podataka ali se ne mogu direktno pročitiati sa kartice.

9.3 Problemi digitalnog potpisivanja dokumenata pomoću pametnih kartica u web okruženju

Kod potpisivanja dokumenata uz pomoć pametnih kartica unutar web pretraživača (form fields, forms, files i ostalo) pojavljuje se problem jer ne postoje standardna rešenja za ovaj postupak, bez obzira na platformu i verziju web browsera.

Danas web browsers ne poseduju ugrađene funkcije za digitalno potpisivanje dokumenata koji se šalju serveru od strane klijenta. Ova osobina je itekako neophodna i zbog toga se posebno izrađuju aplikacije kada je potrebno potvrditi identitet pošiljaoca odn. sa sigurnošću uvrditi da je fajl koji prima server zaista poslat od strane klijenta za kojeg se i predstavlja. Primeri ovakvih aplikacija mogu se naći u e-bankarstvu , elektronskim vladama kao , finansijskim sistemima i drugim online servisima gde je potreban ovaj stepen zaštite.

Postoje dva pristupa rešenju ovog problema.

- Upotreba posebnih ekstenzija za određeni web browser (npr. CAPICOM za Internet Explorer i crypto.SignText() za Netscape i Mozzilu)

- Upotreba Java applet-a , koji bi radili gotovo u svim browser-ima

Ukoliko bi se odlučili za prvo rešenje , potpisivanje delova HTML formi pomoću JavaScript bi bilo lako, međutim ukoliko je potrebno potpisati fajl sa lokalong diska nailazimo na problem jer JavaScript ne može da čita podatke sa lokalnih diskova iz sigurnosnih razloga. Java applet koji je objašnjen u daljem radu može da pristupa fajlovima na lokalnom disku jer je digitalno potpisan.

Po pravilu, Java applet-I se pokreću sa svim dozvolama, bez sigurnosnih ograničenja, I oni mogu čitati lokalne fajlove.

9.4 Kreiranje Smart Card Applet-a.

Cilj ovog rada je kreirati tehnologiju koja omogućava digitalno potpisivanje dokumenata unutar web okruženja pomoću pametnih kartica. Ova tehnologija bi trebala da bude nazavisna od operativnog sistema na kojem se pokreće kao i od web browser-a koji korisnik poseduje. Digitalno potpisivanje će izvršiti java applet unutar korisnikovog web browser-a.

U prvom delu prikazan je pristup pametnoj kartici uz pomoc Java 5.0 . Kasnije cu prikazati korak po korak kreiranja Java applet-a za digialno potpisivanje. Ovaj applet digitalno potpisuje dokument unutar web browser-a pre slanja web serveru pomoću standardnih HTML formi.

9.5 Standardi za pristup pametnim karticama

Kao što sam naveo u radu pametne kartice poseduju sopstveni kripto processor (cryptoprocessor) **,** sopstveni operativni sitem, zaštićenu memoriju kao i sopstveni file system. Pristup ka njima vrši se na različitim nivoima sa različitim protokolima. Standard ISO 7816 definiše glavne komponente pametne kartice i opisuje pristup ka njima na niskom nivou (low-lewel access).

Za viši nivo pristupa pametnoj kartici koristi se PKCS#11 standard. Ovaj standard definiše interfejs aplikacije za komunikaciju sa cryptographics devices (cryptographics tokens) for instance, smart card, hardware cryptographic accelerators, and others..

Aplikacije koje dolaze uz pametne kartice najčešće ispunjavaju PKCS#11 standarde za određene pametne kartice kao i čitače kartica. Implementacija ovog standarda se najčešće izvršava pomoću posebnih biblioteka koda (.dll fajl u Windows operativnom sistemu ili .so fajl u Linux ili UNIX operativnim sistemima) koji mogu biti učitani dinamički .

Na primer, ako koristimo ISO 7816 kompatibilnu Utimaco Software pametnu karticu, onda je PKCS#11 implementacija za ovu karticu sadržana u programskom paketu "Utimaco SafeGuard Smartcard Provider" koji se isporučuje sa karticom. Za ovaj proizvod, pretpostavlja se da je gore spomenuti programski paket instaliran u Windows XP operativnom sistemu , biblioteka (.dll fajl) koja definiše i implementira PKCS#11 standard može se naći kao fajl :

C:\windows\system32\pkcs201n.dll

PKCS#11 standard ne dozvoljava fizičko izvlačenje privatnog ključa sa pametne kartice , ali omogućava pristup i upotrebu ovog ključa za enkripciju , dekripciju kao i digitalno potpisivanje. Naravno, da bi se ovo moglo izvesti korisnik pre svega treba da unese ispravan PIN kod koji štiti od neovlašćenog pristupa kartici.

PKCS#11 standard nam daje interfejs za pristup zaštićenim ključevima i sertifikatima koji se nalaze na pametnoj kartici. Iz ovog razloga ,funkcionalnost pametne kartice ima dosta sličnosti sa funkcionalnosti PKCS#12 keystores. Ipak pametne kartice su na određeni način mnogo pouzdanije i pružaju više mogućnosti od PKCS#12 keystores ; na primer ugradjenu funkciju enkripcije i potpisivanja.

9.6 Pristup pametnim karticama iz Java-e

Sve novije verzije JAVA platforme imaju ugrađenu podršku za pristup pametnim karticama. Ovaj pristup se vrši preko PKCS#11 interfejsa (Cryptographic Token Interface Standard). Interakcija se uspostavlja preko cryptographic service provider, „Sun PKCS#11 Provider".

9.7 Konfigurisanje Sun PKCS#11 Privider

Važno je napomenuti da ukoliko želimo koristiti Sun PKCS#11 Provider on mora prvo biti registovan kao cryptographic services provider u JCA. Registracija se može izvršiti dinamičkim ili statičkim putem. U daljem tekstu objasniću kako registrovati Sun PKCS#11 Provider na oba načina.

9.8 Statična registracija Sun PKCS#11 Provider

Ovaj tip registracije zahteva određene izmene u sledećem fajlu :

%JAVA_HOME%/lib/security/java.security

Da bi dodali podešavanja za novi security provider , potrebno je dodati sledeći par linija koda :

Configuration for security providers 1..6 omitted

Security.provider.7 = sun.security.pkcs11.SunPKCS11

C:\smartcards\config\pkcs11.cfg

Fajl pkcs11.cfg mora da sadrži Sun PKCS#11 podešavanja . To je tekstualni fajl i opisuje neke od opcija provider-a kao i putanju do PKCS#11 biblioteke.

9.9 Dinamička registracija Sun PKCS#11 Provider

Za dinamčku registraciju Sun PKCS#11 Providera potrebno je izmeniti sun.security.pkcs11.SunPKCS11 klasu registrovanu u JCA (*Java Cryptography Architecture*). Ime konfiguracijskog fajla koji definiše sva potrebna podešavanja potrebno je navesti kao u primeru :

String pkcs11ConfigFile = “c:\\smartcards\\config\\pkcs11.cfg“;

Provider pkcs11Provider = new sun.security.pkcs11.SunPKCS11(pkcs11ConfigFile);

Security.addProvider(pkcs11Provider);

9.10 Konfiguracijski fajl za Sun PKCS#11 Provider

U oba navedena slučaja, za statičko i dinamičko registrovanje Sun PKCS#11 Provider, konfiguracijski fajl je neophodan. On je neophodan kako bi SunPKCS11 klasa mogla

pročitati putanju do matične biblioteke koja omogućava implementaciju PKCS#11 standarda. U nastavku je prikazan primer konfiguracijskog fajla :

pkcs11.cfg

name = SmartCard

libraty = c:\windows\system32\pkcs201n.dll

Fajl sadrži dva parametra : **name** i **library**. Vrednost **name** parametra označava ime objekta PKCS#11 u JCA. Parametar **library** definiše putanju do biblioteke (u ovom slučaju .dll unutar Windows platforme) koja implementira PKCS#11 standard. Ukoliko je potrebno podesiti rad sa više različitih pametnih kartica , onda se mora Sun PKCS#11 treba definisati onoliko puta koliko vrsta kartica želimo da koristimo i pri tome svaki put koristimo drugu vrednost za **name** parametar.

Konfiguracijski fajl može da sadrži i dodatne parametre (koji su navedeni u dokumentaciji), ali parametri pokazani u primeru iznad , **name** i **library** su obavezni.

9.11 Upotreba Sun PKCS#11 Privider bez konfiguracijskog fajla

Ukoliko ne želimo da koristimo eksterni konfiguracijski fajl možemo definisati podešavanja Sun PKCS#11 Provider dinamički putem streaminga. Sledeći primer prikazuje ovaj postupak :

```
String pkcs11config =
                    "name = SmartCardn" +
                    "library =  c:\windows\system32\pkcs201n.dll";
byte[] pkcs11configBytes = pkcs11config.getBytes();
ByteArrayInputStream configStream =
                                        newByteArrayInputStream(pkcs11configBytes);
Provider pkcs11Provider =
                            new sun.security.pkcs11.SunPKCS11(configStream);
  security.addProvider(pkcs11Provider);
```

U ovom primeru , kreirali smo stream koji čita potrebna podešavanja, ne iz tekstualnog fajla nego iz stringa. Prvo konvertujemo string u niz bajtova (byte array) i zatim kreiramo stream od niza.

9.12 Odjavljivanje Sun PKCS#11 Provider

Kada nam više nije potreban PKCS#11 provider , poželjno ga je odjaviti kako bi se izbegla nepotrebna potrošnja resursa. Sledeći primer pokazuje sam postupak.

```
Provider pkcs11Provider = ...;
String pkcs11ProviderName = pkcs11Provider.getName();
Security.removeProvider(pkcs11ProviderName);
```

Takođe aktivna sesija sa pametnom karticom može ostati otvorena i izazvati pojedine probleme prilikom sledećeg potpisivanja dokumenta. Druga mogućnost za odjavljivanje PKCS#11 providera je :

```
Security.removeProvider("SunPKCS11-SmartCard");
```

Ime provider-a se sastoji od prefiksa „SunPKCS11-“ na koji se dodaje vrednost parametra **name** koje je definisano u konfiguracijskom fajlu (pkcs11.cfg)

9.13 Pristup Keystore na pametnoj kartici

Nakon uspešnog registrovanja i podešavanja Sum PKCS#11 Provider, možemo ga koristiti za pristup sertifikatima i ključevima koji se nalaze na pametnoj kartici. Ovaj proces ćemo izvesti uz pomoć standardne Java klase za pristup zaštićenim delovima pametne kartice – keystores.

Java.security.KeyStore

Sledeći kod predstavlja primer uspostavljanja pristupa zaštićenim delovima pametne kartice (keystore) :

```
char [] pin = {'1', '2', '3', '4'};
KeyStore smartCardKeyStore = KeyStore.getInstance("PKCS11");
smartCardKeyStore.load(null, pin);
```

Da bi uspešno pristupili keystore na pametnoj kartici potrebno je uneti ispravan PIN kod.

9.14 Preuzimanje sertifikata i privatnih ključeva sa pametne kartice

Nakon uspešnog uspostavljanja veze sa pametnom karticom kao i keystore na njoj , možemo koristiti tj. dobaviti sertifikate i ključeve koji se nalaze u njemu. Svi ključevi ,

setifikati i sertifikacijski lanci (cerification chains) su smešteni pod odgovarajućim pseudonimom (aliases) u keystore. Imena se mogu dobiti preko iteratora.

Kod koji je prikazan ispod predstavlja primer kako prikazati sve sertifikate iz određenog keystore , zajedno sa informacijama o njihovim privatnim ključevima :

```
KeyStore keyStore = ...;

Enumeration aliasesEnum = keyStore.aliases();
while (aliasesEnum.hasMoreElements()) {
  String alias = (String)aliasesEnum.nextElement();
  System.out.println("Alias: " + alias);
  X509Certificate cert =    (X509Certificate) keyStore.getCertificate(alias);
  System.out.println("Certificate: " + cert);
  PrivateKey privateKey =  (PrivateKey) keyStore.getKey(alias, null);
  System.out.println("Private key: " + privateKey);
}
```

Ovaj primer najčešće funkcioniše sa uobičajnim keystore kao i sa keystore koje se nalaze na pametnim karticama. Lozinka (password) nije potreban u slučaju kada se pristupa keystore koji se nalazi na pametnoj kartici.Ovo je moguće zbog toga što je pre kreiranja KeyStore objekta utvrđena ispravnost PIN koda i zbog toga je vrednost lozinke u kodu predstavljena kao **null**.

Na prvi pogled izgleda da se privatni ključ može direktno pročitati sa pametne kartice, ali u stvarnosti ovo nije moguće. Pametne kartice ne dozvoljavaju direktan pristup privatnim ključevima ali omogućuju indirektan pristup za potpisivanje , proveravanje potpisa , enkripciju i dekripciju. U prikazanom primeru , privatni ključ se ne ispisuje nego se kreira interfejs za pristup ključu.

9.15 Potpisivanje podataka pomoću pametne kartice

Nakon uspešnog kreiranja interfejsa za pristup ključu na pametnoj kartici on se može koristiti za potpisivanje podataka kao bilo koji drugi privatni ključ. Potpisivanje dokumenta se odvija tako što se prvo izračunava hash dokumenta koji treba potpisati , zatim se taj hash šalje pametnoj kartici u kojoj kriptoprocesor vrši potpisivanje sa izabranim ključem.Ukoliko je potpisivanje uspešno , pametna kartica vraća potpisani hash. Na ovaj način kartica čuva privatni ključ od direktnog pristupa i drži ključ u tajnosti čak i od vlasnika kartice. Sledeći kod prikazuje potpisivanje podataka pomoću interfejsa ka određenom ključu :

```
private static byte[] signDocument(byte[] aDocument,  PrivateKey aPrivateKey)
throws GeneralSecurityException {
  Signature signatureAlgorithm =    Signature.getInstance("SHA1withRSA");
  signatureAlgorithm.initSign(aPrivateKey);
  signatureAlgorithm.update(aDocument);
  byte[] digitalSignature = signatureAlgorithm.sign();
  return digitalSignature;
}
```

9.16 Sistemski zahtevi za pristup pametnim karticama pomoću Java appleta

Java applet za digitalno potpisivanje zahtevaju instaliran Java Plug-In, verzija 5.0 ili noviji, na računaru klijenta. Ovo je neophodno jer applet koristi Sun PKCS#11 Provider, koji je podržan u Java verzija 5.0 i novije.

9.17 Implementacija Applet-a za potpisivanje pomoću pametnih kartica

Proučavanjem samog postupka potpisavanja dokumenata najpre sam naišao na potpisivanje dokumenata pomoću PFX fajlova koji u sebi sadrže privatne ključeve a sadržaj ovih fajlova je zaštićen lozinkom (password). Prilikom zahteva za dobijanje ključa korisnik prvo mora da unese lozinku. Treba napomenuti da na ovaj način funkcioniše PKCS#12 standard. PKCS#11 standard koji koristimo za potpisivanje dokumenta pomoću pametnih kartica je dosta sličan sa već pomenutim PKCS#12 standardom. Razlika je u tome što se u PKCS#12 ključevi čuvaju u fajlu na lokalnom disku ili USB flashu i samim tim mogu biti kopirani od strane napadača koji će kasnije iskoristii brute-force napad kako bi dobio vrednosti kljuceva koji se nalaze u PFX fajlu. U PKCS#11 standardu kllućevi se čuvaju na kartici gde su mnogo sigurniji.

Listing 1 (dodatku koji se nalazi na kraju rada) prikazuje izvorni kod *SmartCardSignerApplet* klase. Glavna **main** klasa applet-a koristi drugu dodatnu klasu , *PKCS11LibraryFileAndPINCodeDialog* , koji obezbeđuje korisniku da izabere PKCS#11 biblioteku (.dll fajl pod Windows platformom) kao i PIN kod za pristup pametnoj kartici što je prikazano u Listingu 2.

Applet takođe koristi i klasu Base64Units (Listing 3) za BASE64 kodiranje i dekodiranje.

9.18 Kako funkcioniše applet za potpisivanje pomoću pametne kartice

Pomoću klase netscape.javascript.JSObject preuzimamo ime fajla za potpisivanje iz HTML forme i ovaj fajl se smešta u memoriju računara. Ovo izvodljivo jer smo predhodno potpisali applet što dozvoljava pristup appleta lokalnim fajlovima.

Nakon ovog koraka prikazuje se prozor dijaloga gde se zahteva od korisnika da izabere PKCS#11 biblioteku i unese validan PIN kod za pristup pametnoj kartici. Nakon uspešnog procesa omogućen je pristup sertifikatima na kartici , sertifikacijskim lancima (ukoliko su prikazani) kao i privatnim ključevima. Naravno direktan pristup privatnom ključu nije dozvoljen ali se kreira Java interfejs za pristup ključu.

Na kraju možemo fajl koji je prethodno smešten u memoriju potpisati uz pomoć ključa koji se nalazi na pametnoj kartici. Dobijeni rezultat potpisa se pretvara u tekstalni format (BASE64) i upisuje se u odgovarajuće polje u HTML formi.

Ukoliko se u nekom od navedenih koraka pojavi greška korisnik će biti informisan. Greške mogu biti razne : ukoliko se fajlu za potpisivanje ne može pristupiti ; ukoliko biblioteka za implementaciju PKCS#11 nije odgovarajuća ; ako PIN kod nije validan ; ukoliko se na kartici ne nalazi odgovarajući sertifikat ili privatni ključ ... itd.

Java applet za korisnički interfejs je baziran na biblioteci koja se nalazi u AWT i JFC/Swing , koje dolaze kao standard u JDK 5.0.

Nakon što je uspešno registrovan PKCS#11 Provider, pristup keystore koji se nalazi na pametnoj kartici se odvija kroz klasu java.security.KeyStore. Applet očekuje da se na kartici nalazi samo jedan zapis (alias), u kojem se čuva korisnikov sertifikat i odgovarajući privatni ključ.

Implementacija svih kriptografskih algoritama je obezbeđena pomoću standardnih kriptografskih servisa SunJSSE , koji su sastavni deo JDK 5.0 i noviji.

U navedenom primeru algoritam za digitalno potpisivanje je „SHA1withRSA“ koji je dostupan u Java kroz klasu java.security.Signature i podržan je od većeg broja pametnih kartica.

9.19 Kopmajliranje i potpisivanje Applet-a

Da bi applet u potpunosti funkcionisao prethodno mora da bude potpisan. Ovo je neophodno zbog toga što applet treba da pristupa fajlovima koji se nalaze na lokalnom računaru odn. na kliejntovom disku. Sledeći primer prikazuje skriptu za generisanje za generisanje sertifikata za samo-potpisivanje koji ćemo iskoristiti za potpisivanje applet-a.

Generate-certificate.bat

```
del SmartCardSignerApplet.jks
keytool -genkey -alias signFiles
  -keystore SmartCardSignerApplet.jks
  -keypass !secret -dname "CN=Your Company"
  -storepass !secret
pause
```

Kao rezultat izvršavanja ove skripte dobićemo kezstore fajl, SmartCardSignerApllet.jks, koji sadrži generisani sertifikat i njegov odgovarajući privatni ključ sačuvan pod imenom „signFile“ i lozinka za pristup „!secret“. Format odn. ekstenzija fajla je JKS (Java KeyStore).

Kako bi kompajlirali izvorni kod , kereiamo JAR arhivu i da bi je potpisali korisićemo sledeću skriptu:

build-script.bat

```
set JAVA5_HOME=C:Progra~1Javajdk1.5.0_04

del *.class
%JAVA5_HOME%binjavac -classpath .;
  "%JAVA5_HOME%jrelibplugin.jar" *.java

del *.jar
%JAVA5_HOME%binjar -cvf SmartCardSignerApplet.jar *.class
%JAVA5_HOME%binjarsigner -keystore SmartCardSignerApplet.jks
  -storepass !secret
  -keypass !secret
  SmartCardSignerApplet.jar signFiles

pause
```

Određeni delovi koda brišu sve kompajlirane .class fajlove, kompajliraju sve .java fajlove, smeštaju dobijene .class fajlove u arhivu SmartCardSignerApplet.jar i potpisuju ovu arhivu sa unapred generisanim sertifikatom za samo-potpisivanje (koji se nalazi u SmartCardSignerApplet.jks kezstore fajlu). Za kompajliranje potreban je JDK 5.0 i noviji.

9.20 Testiranje applet-a pomoću jednostavne HTML forme

Na kraju možemo da testiramo applet unutar jednostavnog HTML dokumenta koji sadrži HTML formu koja komunicira sa samim applet-om. (Listing 4)

Važno je napomenuti da u kodu treba koristiti <object> i <embed> tagove umesto starog <applet> taga. Ukoliko koristimo star tag , ne možemo napomenuti web browseru da je potrebna Java verzija 5.0 ili novija.

Jednostavna HTML stranica sadrži HTML formu sa tri polja, applet za potpisivanje i dugme za pokretanje aplikacije koje je sastavni deo applet-a. Tri polja u HTML formi nam služe za izbor fajla za potpisivanje , za prikaz (certification chain) kao i rezultat samog digitalnog potpisivanja dokumenta.

PRIMER

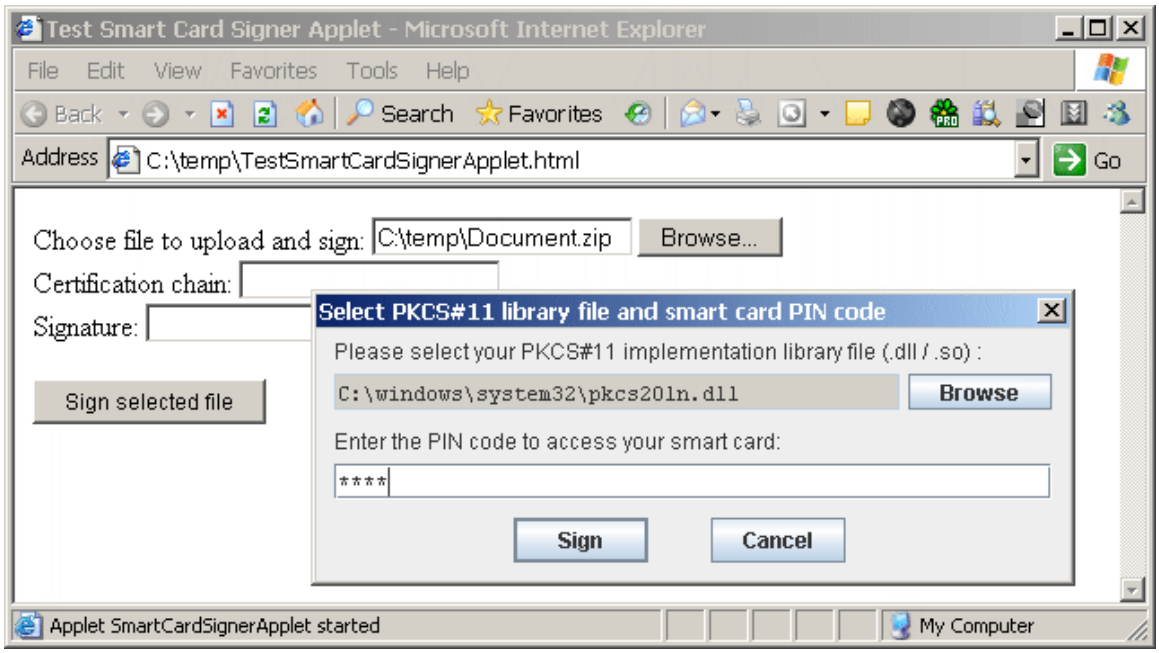
Nakon otvaranja HTML dokumenta , prvo se proverava verzija Java na klijentovom računaru. Ukoliko je verzija starija od verzije 5.0 , korisnik se automatski preusmerava ka web sajtu gde može da se preuzme najnovija verzija Java. Ukoliko korisnik poseduje odgovarajuću verziju, nakon učitavanja HTML dokumenta pojaviće se prozor za dijalog gde se zahteva od korisnika da prihvati izvršavanje appleta. (Slika 9.2)

Slika 9.2 *Java Plug-in 5.0 zahtev za izvršavanje potpisanog applet-a*

Ukoliko korisnik veruje kompaniji koja je navedena u sertifikatu (Your Company) i prihvati izvršavanje appleta klikom na dugme YES , applet će se pokrenuti.

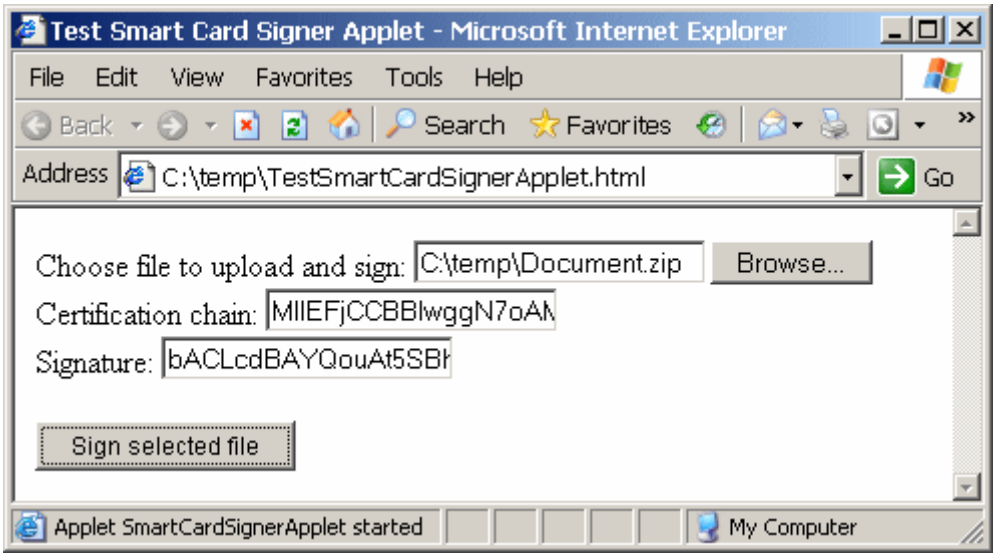
U daljem toku korisnik treba da izabere fajl koji želi da potpiše (izbor fajla će izvršiti pomoću HTML polja koji se nalazi na stranici) i klikne na dugme za potpisivanje (Sign selected file dugme) koje je sastavni deo appleta.

Sada se prikazuje prozor u kome korisnik treba da izabere PKCS#11 biblioteku I da unese odgovarajući PIN kod za pametnu karticu. (Slika 9.3)



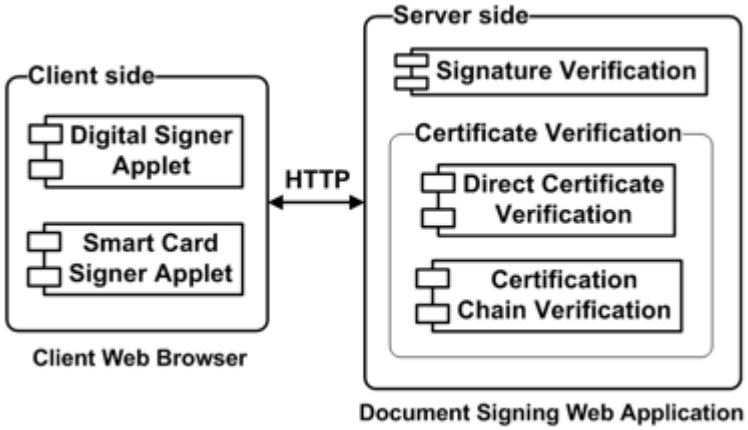
Slika 9.3 *Potpisivanje pomoću pametne kartice u web okruženju*

Ukoliko je proces potpisivanja uspešno izvršen rezultat bi trebao da bude kao na slici 9.4 :



Slika 9.4 *Rezultat uspešnog procesa potpisivanja dokumenta*

Predstavljeni sistem koristi tradicionalnu klijent-server arhitekturu sačinjenu od standardnog Web browsera i Java Web aplikacije , kako je prikazano na slici 9.5.



Slika 9.5 *Sistem za digitalno potpisivanje dokumenata u Web okruženju*

Na strani klijenta web browser obavlja odn. izvršava Java applet za digitalno potpisivanje dokumenta.

Server pokreće Java baziranu Web aplikaciju. Ona prihvata potpisani document i proverava njegov digitalni potpis kao i sertifikat korišten za njegovo potpisivanje.

ZAKLJUČAK

Ogroman napredak informacionih tehnologija je omogućio da gotovo svaka oblast iz naše svakodnevnice egzistira i u virtuelnom svetu. Internet omogućava daleko brže i efikasnije delovanje u raznim sferama života, uključujući trgovinu, bankarstvo, zdravstvene usluge, glasanje itd. čime se otvara pitanje sigurnosti obavljanja on line transakcija preko nesigurne mreže kakva je Internet, tako da je primena savremenih kriptografskih rešenja ne samo neophodna nego i jedan od osnovnih preduslova za njihovo obavljanje.

Pametne kartice danas polako postaju uređaj kojeg ćemo u bliskoj budućnosti svi imati u novčaniku. Do sve veće popularnosti pametnih kartica dolazi zbog njihove glavne prednosti naspram magnetskih i memorijskih kartica, a to je sigurnost koju pametna kartica pruža za podatke koji se nalaze na njoj. Java Card tehnologija jedna je od vodećih tehnologija za izradu programa namijenjenih izvršavanju na pametnoj kartici. Pametne kartice imaju i svojih nedostataka međutim bez obzira na njih Java Card tehnologiju čeka svijetla budućnost.

Kroz praktičan rad prikazan je primer upotrebe pametnih kartica u web okruženju. Predstavljen je samo osnovni deo koji predstavlja sam čin komunikacije aplikacije sa pametnom karticom , korićenje dobijenih resursa za digitalno potpisivanje dokumenta. Nadogradnjom i slobodnom izmenom koda prikazani primer se može prilagoditi i za druge načine upotrebe kartice. Kao npr. za identifikaciju , potvrdu identiteta bez potpisivanja dokumenta sa lokalnog diska itd.

Rad se sastoji iz devet poglavlja gde svako poglavlje opisuje :

- **Prvi deo :** u ovom delu se govori uopšteno o kriptografiji. Objašnjeni su simetrični i asimetrični algoritmi kaoi si jednosmerne hash funkcije.
- **Drugi deo :** objašnjava tehniku digitalnog potpisa koji sam kasnije koristi u praktičnom delu rada.
- **Treći deo :** algoritmi za razmenu ključeva . “Algoritmi za razmenu ključa služe za sigurnu i poverljivu komunikaciju između dva ili više subjekta. Poverljivost informacije se temelji na paru ključeva kojim se poruka kriptuje, odnosno dekriptuje.”
- **Četvrti deo. :** u ovom delu se govori o PKI infrastrukturi. “IETF definicija PKI sistema glasi: *PKI je skup hardwer-a, programske opreme, ljudi, pravila i*

*funkcija potrebnih za stvaranje, upravljanje, pohranjivanje, distribuiranje i opozivanje sertifikata baziranih na kriptografiji javnim ključe*m.“

- **Peti deo :** opisuje X.509 sertifikat javnog ključa. “Sertifikat je najvažniji deo celokupnog PKI sistema. Na njemu počiva uzajamno poverenje vlasnika sertifikata i onoga ko sertifikat proverava.“
- **Šesti deo u** ovom delu je opisana glavna tema rada , pametne kartice (smart card). Njihova upotreba , funkcionalnost , objašnjenje načina rada itd.
- **Sedmi deo :** govori o Java Card tehnologiji. „Java Card tehnologija predstavlja najmanju Java platformu koja je namenjena memorijski ograničenim uređajima poput pametnih kartica, te omogućava da se programi pisani u Java programskom jeziku mogu izvršavati na takvim uređajima. “.
- **Osmi deo :** opisuje OpenCard tehnologiju. OpenCard tehnologija je otvorena arhitektura napravljena u programskom jeziku Java koja daje programski interfejs koji omogućava komunikaciju s bilo kojom karticom i bilo kojim čitačem neovisno o proizvođaču.
- **Deveti deo :** predstavlja praktičan rad u kome je implementirana gore navedena teorija.

LITERATURA

[1] Bruce Schneier , Primenjena kriptografija – drugo izdanje

[2] Man Young Rhee , Internet Securiy – Cryptographic principles, algorithms and protocols .

[3] RSA Laboratories, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) profile*, dostupno na Internet adresi *http://tools.ietf.org/html/rfc3280#section-5.3.1*

[4] The Apache software foundation, *Tomcat 5.0.28. manual* dostupno na Internet adresi *http://tomcat.apache.org*

[5] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CLR). Dostupno na interent adresi http://www.ietf.org/rfc/rfc3280.txt (Visited Dec. 2009)

[6] OpenCA projekt, dostupno na Internet adresi www.openca.org/openca (Visited Nov. 2009)

[7] Apache web server, dostupno na Internet adresi www.apache.org

[8] *W. Rankl, W. Effing*: **Smart Card - Hand Book**, Third Edition, John Wiley & Sons, New York, 1999.

[9] *Anthony Cola:* **Security Is Not Child"s Play**, Card Technology, August 2002. http://www.basiccard.com

[10] OpenCard consortium, *http://www.opencard.org* (Visited Nov. 2009)

[11] Sun Microsytems, *Java Card™ Development Kit User's Guide*

[12] *Anthony Cola:* **Security Is Not Child"s Play**, Card Technology, August 2002. http://www.basiccard.com

[13] ActivCard, dostupno na Internet adresi www.actividentitz.com.

[14] The ISO 7816 Smart Card Standard http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816.aspx

DODATAK

Listing 1: SmartCardSignerApplet.java

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ByteArrayInputStream;
import java.util.Arrays;
import java.util.Enumeration;
import java.util.List;
import java.security.*;
import java.security.cert.CertPath;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.lang.reflect.Constructor;

import netscape.javascript.JSException;
import netscape.javascript.JSObject;

* This file is part of NakovDocumentSigner digital document
 * signing framework for Java-based Web applications:
 * http://www.nakov.com/documents-signing/
 *
 * Copyright (c) 2005 by Svetlin Nakov - http://www.nakov.com
 * All rights reserved. This code is freeware. It can be used
 * for any purpose as long as this copyright statement is not
 * removed or modified.
 */
public class SmartCardSignerApplet extends Applet {

   private static final String FILE_NAME_FIELD_PARAM =
      "fileNameField";
   private static final String CERT_CHAIN_FIELD_PARAM =
      "certificationChainField";
   private static final String SIGNATURE_FIELD_PARAM =
      "signatureField";
   private static final String SIGN_BUTTON_CAPTION_PARAM =
      "signButtonCaption";

   private static final String PKCS11_KEYSTORE_TYPE =
      "PKCS11";
   private static final String X509_CERTIFICATE_TYPE =
      "X.509";
   private static final String CERTIFICATION_CHAIN_ENCODING =
      "PkiPath";
   private static final String DIGITAL_SIGNATURE_ALGORITHM_NAME =
```

```
   "SHA1withRSA";
private static final String SUN_PKCS11_PROVIDER_CLASS =
   "sun.security.pkcs11.SunPKCS11";

private Button mSignButton;

/**
 * Initializes the applet - creates and initializes its graphical
 * user interface. Actually the applet consists of a single
 * button, that fills its all surface. The button's caption is
 * taken from the applet parameter SIGN_BUTTON_CAPTION_PARAM.
 */

public void init() {
   String signButtonCaption =
      this.getParameter(SIGN_BUTTON_CAPTION_PARAM);
   mSignButton = new Button(signButtonCaption);
   mSignButton.setLocation(0, 0);
   Dimension appletSize = this.getSize();
   mSignButton.setSize(appletSize);
   mSignButton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
         signSelectedFile();
      }
   });
   this.setLayout(null);
   this.add(mSignButton);
}

/**
 * Signs the selected file. The file name comes from a field
 * in the HTML document. The result consists of the calculated
 * digital signature and certification chain, both placed in
 * fields in the HTML document, encoded in Base64 format.
 * The HTML document should contain only one HTML form.
 * The name of the field, that contains the name of the file
 * to be signed is obtained from FILE_NAME_FIELD_PARAM applet
 * parameter. The names of the output fields for the signature
 * and the certification chain are obtained from the parameters
 * CERT_CHAIN_FIELD_PARAM and SIGNATURE_FIELD_PARAM. The user
 * is asked to choose a PKCS#11 implementation library and a
 * PIN code for accessing the smart card.
 */

private void signSelectedFile() {
   try {
      // Get the file name to be signed from the form in the
      // HTML document
      JSObject browserWindow = JSObject.getWindow(this);
      JSObject mainForm =
         (JSObject) browserWindow.eval("document.forms[0]");
      String fileNameFieldName =
         this.getParameter(FILE_NAME_FIELD_PARAM);
      JSObject fileNameField =
        (JSObject) mainForm.getMember(fileNameFieldName);
      String fileName = (String) fileNameField.getMember("value");
```

```
      // Perform the actual file signing
      CertificationChainAndSignatureBase64 signingResult =
         signFile(fileName);
      if null) {
         // Document signed. Fill the certificate and
         // signature fields
         String certChainFieldName =
            this.getParameter(CERT_CHAIN_FIELD_PARAM);
         JSObject certChainField =
            (JSObject) mainForm.getMember(certChainFieldName);
         certChainField.setMember("value",
            signingResult.mCertificationChain);
         String signatureFieldName =
            this.getParameter(SIGNATURE_FIELD_PARAM);
         JSObject signatureField =
            (JSObject) mainForm.getMember(signatureFieldName);
         signatureField.setMember("value",
             signingResult.mSignature);
      else {
         // User canceled signing
      }
   }
   catch (DocumentSignException dse) {
      // Document signing failed. Display error message
      String errorMessage = dse.getMessage();
      JOptionPane.showMessageDialog(this, errorMessage);
   }
   catch (SecurityException se) {
      se.printStackTrace();
      JOptionPane.showMessageDialog(this,
         "Unable to access the local file system.\n" +
         "This applet should be started with full security
          permissions.\n" +
         "Please accept to trust this applet when the Java
          Plug-In asks you.");
   }
   catch (JSException jse) {
      jse.printStackTrace();
      JOptionPane.showMessageDialog(this,
         "Unable to access some of the fields of the\n" +
         "HTML form. Please check the applet parameters.");
   }
   catch (Exception e) {
      e.printStackTrace();
      JOptionPane.showMessageDialog(this,
         "Unexpected error: " + e.getMessage());
   }
}

/**
 * Signs given local file. The certificate and private key to
 * be used for signing come from the locally attached smart
 * card. The user is requested to provide a PKCS#11
 * implementation library and the PIN code for accessing the
 * smart card. @param aFileName the name of the file to be
 * signed.
 * @return the digital signature of the given file and the
```

```
     * certification chain of the certificatie used for signing the
     * file, both Base64-encoded or null if the signing process is
     * canceled by the user.
     * @throws DocumentSignException when a problem arised during
     * the singing process (e.g. smart card access problem,
     * invalid certificate, invalid PIN code, etc.)
     */

    private CertificationChainAndSignatureBase64 signFile(String
       aFileName)
    throws DocumentSignException {

       // Load the file for signing
       byte[] documentToSign = null;
       try {
          documentToSign = readFileInByteArray(aFileName);
       } catch (IOException ioex) {
          String errorMsg = "Cannot read the file for signing " +
             aFileName + ".";
          throw new DocumentSignException(errorMsg, ioex);
       }

       // Show a dialog for choosing PKCS#11 implementation library
       // and smart card PIN
       PKCS11LibraryFileAndPINCodeDialog pkcs11Dialog =
          new PKCS11LibraryFileAndPINCodeDialog();
       boolean dialogConfirmed;
       try {
          dialogConfirmed = pkcs11Dialog.run();
       } finally {
          pkcs11Dialog.dispose();
       }

       if (dialogConfirmed) {
          String oldButtonLabel = mSignButton.getLabel();
          mSignButton.setLabel("Working...");
          mSignButton.setEnabled(false);
          try {
             String pkcs11LibraryFileName =
                pkcs11Dialog.getLibraryFileName();
             String pinCode = pkcs11Dialog.getSmartCardPINCode();

             // Do the actual signing of the document with the
             // smart card
             CertificationChainAndSignatureBase64 signingResult =
                signDocument(documentToSign, pkcs11LibraryFileName,
                   pinCode);
             return signingResult;
          finally {
             mSignButton.setLabel(oldButtonLabel);
             mSignButton.setEnabled(true);
          }
       }
       else {
          return null;
       }
    }
```

```
private CertificationChainAndSignatureBase64 signDocument(
   byte[] aDocumentToSign, String aPkcs11LibraryFileName,
      String aPinCode)
throws DocumentSignException {
   if (aPkcs11LibraryFileName.length() == 0) {
      String errorMessage = "It is mandatory to choose " +
         "a PCKS#11 native implementation library for " +
         "smart card (.dll or .so file)!";
      throw new DocumentSignException(errorMessage);
   }

   // Load the keystore from the smart card using the specified
   // PIN code
   KeyStore userKeyStore = null;
   try{
      userKeyStore = loadKeyStoreFromSmartCard(
          aPkcs11LibraryFileName, aPinCode);
   catch (Exception ex) {
      String errorMessage = "Cannot read the keystore from "
         "the smart card.\n" +
         "Possible reasons:\n" +
         " - The smart card reader in not connected.\n" +
         " - The smart card is not inserted.\n" +
         " - The PKCS#11 implementation library is invalid.\n" +
         " - The PIN for the smart card is incorrect.\n" +
         "Problem details: " + ex.getMessage();
      throw new DocumentSignException(errorMessage, ex);
   }

   // Get the private key and its certification chain from the
   // keystore
   PrivateKeyAndCertChain privateKeyAndCertChain = null;
   try {
      privateKeyAndCertChain =
         getPrivateKeyAndCertChain(userKeyStore);
   } catch (GeneralSecurityException gsex) {
      String errorMessage = "Cannot extract the private key " +
         "and certificate from the smart card. Reason: " +
         gsex.getMessage();
      throw new DocumentSignException(errorMessage, gsex);
   }

   // Check if the private key is available
   PrivateKey privateKey = privateKeyAndCertChain.mPrivateKey;
   if (privateKey == null) {
      String errorMessage = "Cannot find the private key on " +
         "the smart card.";
      throw new DocumentSignException(errorMessage);
   }

   // Check if X.509 certification chain is available
   Certificate[] certChain =
      privateKeyAndCertChain.mCertificationChain;
   if (certChain == null) {
      String errorMessage = "Cannot find the certificate on " +
         "the smart card.";
```

```
            throw new DocumentSignException(errorMessage);
        }

        // Create the result object
        CertificationChainAndSignatureBase64 signingResult =
            new CertificationChainAndSignatureBase64();

        // Save X.509 certification chain in the result encoded in
        // Base64
        try {
            signingResult.mCertificationChain =
                encodeX509CertChainToBase64(certChain);
        }
        catch (CertificateException cee) {
            String errorMessage = "Invalid certificate on the "
                "smart card.";
            throw new DocumentSignException(errorMessage);
        }

        // Calculate the digital signature of the file,
        // encode it in Base64 and save it in the result
        try {
            byte[] digitalSignature =
                signDocument(aDocumentToSign, privateKey);
            signingResult.mSignature =
                Base64Utils.base64Encode(digitalSignature);
        catch (GeneralSecurityException gsex) {
            String errorMessage = "File signing failed.\n" +
                "Problem details: " + gsex.getMessage();
            throw new DocumentSignException(errorMessage, gsex);
        }

        return signingResult;
    }

    /**
     * Loads the keystore from the smart card using its PKCS#11
     * implementation library and the Sun PKCS#11 security provider.
     * The PIN code for accessing the smart card is required.
     */

    private KeyStore loadKeyStoreFromSmartCard(String
        aPKCS11LibraryFileName, String aSmartCardPIN)
    throws GeneralSecurityException, IOException {
        // First configure the Sun PKCS#11 provider. It requires a
        // stream (or file) containing the configuration parameters -
        // "name" and "library".
        String pkcs11ConfigSettings =
            "name = SmartCard\n" + "library = " + aPKCS11LibraryFileName;
        byte[] pkcs11ConfigBytes = pkcs11ConfigSettings.getBytes();
        ByteArrayInputStream confStream =
            new ByteArrayInputStream(pkcs11ConfigBytes);

        // Instantiate the provider dynamically with Java reflection
        try {
            Class sunPkcs11Class =
                Class.forName(SUN_PKCS11_PROVIDER_CLASS);
```

```
      Constructor pkcs11Constr = sunPkcs11Class.getConstructor(
         java.io.InputStream.class);
      Provider pkcs11Provider =
        (Provider) pkcs11Constr.newInstance(confStream);
      Security.addProvider(pkcs11Provider);
   catch (Exception e) {
      throw new KeyStoreException("Can initialize " +
         "Sun PKCS#11 security provider. Reason: " +
         e.getCause().getMessage());
   }

   // Read the keystore form the smart card
   char[] pin = aSmartCardPIN.toCharArray();
   KeyStore keyStore = KeyStore.getInstance(PKCS11_KEYSTORE_TYPE);
   keyStore.load(null, pin);
   return keyStore;
}

/**
 * @return private key and certification chain corresponding
 * to it, extracted from the given keystore. The keystore is
 * considered to have only one entry that contains both
 * certification chain and its corresponding private key.
 * If the keystore has no entries, an exception is thrown.
 */

private PrivateKeyAndCertChain getPrivateKeyAndCertChain(
   KeyStore aKeyStore)
throws GeneralSecurityException {
   Enumeration aliasesEnum = aKeyStore.aliases();
   if (aliasesEnum.hasMoreElements()) {
      String alias = (String)aliasesEnum.nextElement();
      Certificate[] certificationChain =
         aKeyStore.getCertificateChain(alias);
      PrivateKey privateKey =
         (PrivateKey) aKeyStore.getKey(alias, null);
      PrivateKeyAndCertChain result = new PrivateKeyAndCertChain();
      result.mPrivateKey = privateKey;
      result.mCertificationChain = certificationChain;
      return result;
   else {
      throw new KeyStoreException("The keystore is empty!");
   }
}

/**
 * @return Base64-encoded ASN.1 DER representation of given
 * X.509 certification chain.
 */

private String encodeX509CertChainToBase64
   (Certificate[] aCertificationChain)
throws CertificateException {
   List certList = Arrays.asList(aCertificationChain);
   CertificateFactory certFactory =
      CertificateFactory.getInstance(X509_CERTIFICATE_TYPE);
   CertPath certPath = certFactory.generateCertPath(certList);
```

```
   byte[] certPathEncoded =
      certPath.getEncoded(CERTIFICATION_CHAIN_ENCODING);
   String base64encodedCertChain =
      Base64Utils.base64Encode(certPathEncoded);
   return base64encodedCertChain;
}

/**
 * Reads the specified file into a byte array.
 */

private byte[] readFileInByteArray(String aFileName)
throws IOException {
   File file = new File(aFileName);
   FileInputStream fileStream = new FileInputStream(file);
   try {
      int fileSize = (int) file.length();
      byte[] data = new byte[fileSize];
      int bytesRead = 0;
      while (bytesRead < fileSize) {
         bytesRead += fileStream.read(
            data, bytesRead, fileSize-bytesRead);
      }
      return data;
   }
   finally {
      fileStream.close();
   }
}

/**
 * Signs given document with a given private key.
 */

private byte[] signDocument(byte[] aDocument,
                            PrivateKey aPrivateKey)
throws GeneralSecurityException {
   Signature signatureAlgorithm =
      Signature.getInstance(DIGITAL_SIGNATURE_ALGORITHM_NAME);
   signatureAlgorithm.initSign(aPrivateKey);
   signatureAlgorithm.update(aDocument);
   byte[] digitalSignature = signatureAlgorithm.sign();
   return digitalSignature;
}

/**
 * Data structure that holds a pair of private key and
 * certification chain corresponding to this private key.
 */

static class PrivateKeyAndCertChain {
   public PrivateKey mPrivateKey;
   public Certificate[] mCertificationChain;
}

/**
 * Data structure that holds a pair of Base64-encoded
```

```
     * certification chain and digital signature.
     */

   static class CertificationChainAndSignatureBase64 {
      public String mCertificationChain = null;
      public String mSignature = null;
   }

   /**
    * Exception class used for document signing errors.
    */

   static class DocumentSignException extends Exception {
      public DocumentSignException(String aMessage) {
         super(aMessage);
      }

      public DocumentSignException(String aMessage,
            Throwable aCause) {
         super(aMessage, aCause);
      }
   }

}
```

Listing 2: PKCS11LibraryFileAndPINCodeDialog.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import java.io.*;
import java.util.Properties;

public class PKCS11LibraryFileAndPINCodeDialog extends JDialog {

   private static final String CONFIG_FILE_NAME =
      ".smart_card_signer_applet.config";
   private static final String PKCS11_LIBRARY_FILE_NAME_KEY =
      "last-PKCS11-file-name";

   private JButton mBrowseForLibraryFileButton = new JButton();
   private JTextField mLibraryFileNameTextField = new JTextField();
   private JLabel mChooseLibraryFileLabel = new JLabel();
   private JTextField mPINCodeTextField = new JPasswordField();
   private JLabel mEnterPINCodeLabel = new JLabel();
   private JButton mSignButton = new JButton();
   private JButton mCancelButton = new JButton();
```

```
private boolean mResult = false;

/**
 * Initializes the dialog - creates and initializes its GUI
 * controls.    */

public PKCS11LibraryFileAndPINCodeDialog() {
   // Initialize the dialog
   this.getContentPane().setLayout(null);
   this.setSize(new Dimension(426, 165));
   this.setBackground(SystemColor.control);
   this.setTitle("Select PKCS#11 library file and smart " +
      "card PIN code");
   this.setResizable(false);

   // Center the dialog in the screen
   Dimension screenSize =
      Toolkit.getDefaultToolkit().getScreenSize();
   Dimension dialogSize = this.getSize();
   int centerPosX =
      (screenSize.width - dialogSize.width) / 2;
   int centerPosY =
      (screenSize.height - dialogSize.height) / 2;
   setLocation(centerPosX, centerPosY);

   // Initialize certificate keystore file label
   mChooseLibraryFileLabel.setText(
      "Please select your PKCS#11 implementation library file
       (.dll / .so) :");
   mChooseLibraryFileLabel.setBounds(new Rectangle(10, 5, 400, 15));
   mChooseLibraryFileLabel.setFont(new Font("Dialog", 0, 12));

   // Initialize certificate keystore file name text field
   mLibraryFileNameTextField.setBounds(
      new Rectangle(10, 25, 315, 20));
   mLibraryFileNameTextField.setFont(
      new Font("DialogInput", 0, 12));
   mLibraryFileNameTextField.setEditable(false);
   mLibraryFileNameTextField.setBackground(SystemColor.control);

   // Initialize browse button
   mBrowseForLibraryFileButton.setText("Browse");
   mBrowseForLibraryFileButton.setBounds(
      new Rectangle(330, 25, 80, 20));
   mBrowseForLibraryFileButton.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            browseForLibraryButton_actionPerformed();
         }
      });

   // Initialize PIN code label
   mEnterPINCodeLabel.setText("Enter the PIN code to access " +
      "your smart card:");
   mEnterPINCodeLabel.setBounds(new Rectangle(10, 55, 350, 15));
   mEnterPINCodeLabel.setFont(new Font("Dialog", 0, 12));
```

```
      // Initialize PIN code text field
      mPINCodeTextField.setBounds(new Rectangle(10, 75, 400, 20));
      mPINCodeTextField.setFont(new Font("DialogInput", 0, 12));

      // Initialize sign button
      mSignButton.setText("Sign");
      mSignButton.setBounds(new Rectangle(110, 105, 75, 25))
      mSignButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            signButton_actionPerformed();
         }
      });

      // Initialize cancel button
      mCancelButton.setText("Cancel");
      mCancelButton.setBounds(new Rectangle(220, 105, 75, 25));
      mCancelButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            cancelButton_actionPerformed();
         }
      });

      // Add the initialized components into the dialog's content pane
      this.getContentPane().add(mChooseLibraryFileLabel, null);
      this.getContentPane().add(mLibraryFileNameTextField, null);
      this.getContentPane().add(mBrowseForLibraryFileButton, null);
      this.getContentPane().add(mEnterPINCodeLabel, null);
      this.getContentPane().add(mPINCodeTextField, null);
      this.getContentPane().add(mSignButton, null);
      this.getContentPane().add(mCancelButton, null);
      this.getRootPane().setDefaultButton(mSignButton);

      // Add some functionality for focusing the most appropriate
      // control when the dialog is shown
      this.addWindowListener(new WindowAdapter() {
         public void windowOpened(WindowEvent windowEvent) {
            String libraryFileName =
               mLibraryFileNameTextField.getText();
            if (libraryFileName != null && libraryFileName.length() !=
0)
               mPINCodeTextField.requestFocus();
            else
               mBrowseForLibraryFileButton.requestFocus();
         }
      });
   }

   /**
    * Called when the "Browse" button is pressed.
    * Shows file choose dialog and allows the user to locate a
    * library file.
    */

   private void browseForLibraryButton_actionPerformed() {
      JFileChooser fileChooser = new JFileChooser();
      LibraryFileFilter libraryFileFilter = new LibraryFileFilter();
      fileChooser.addChoosableFileFilter(libraryFileFilter);
```

```
   String libraryFileName = mLibraryFileNameTextField.getText();
   File directory = new File(libraryFileName).getParentFile();
   fileChooser.setCurrentDirectory(directory);
   if (fileChooser.showOpenDialog(this)
      == JFileChooser.APPROVE_OPTION) {
   String selectedLibFile =
      fileChooser.getSelectedFile().getAbsolutePath();
   mLibraryFileNameTextField.setText(selectedLibFile);
   }
}

/**
 * Called when the sign button is pressed. Closses the dialog
 * and sets the result flag to true to indicate that the user
 * is confirmed the information entered in the dialog.
 */

private void signButton_actionPerformed() {
   mResult = true;
   this.setVisible(false);
}

/**
 * Called when the cancel button is pressed. Closes the dialog
 * and sets the result flag to false that indicates that the
 * dialog is canceled.
 */

private void cancelButton_actionPerformed() {
   mResult = false;
   this.setVisible(false);
}

/**
 * @return the file name with full path to it where the dialog
 * settings are stored.
 */

private String getConfigFileName() {
   String configFileName = System.getProperty("user.home") +
      System.getProperty("file.separator") + CONFIG_FILE_NAME;
   return configFileName;
}

/**
 * Loads the dialog settings from the dialog configuration file.
 * These settings consist of a single value - the last used
 * library file name with its full path.
 */

private void loadSettings()
throws IOException {
   String configFileName = getConfigFileName();
   FileInputStream configFileStream =
      new FileInputStream(configFileName);
   try {
      Properties configProps = new Properties();
```

```
      configProps.load(configFileStream);

      // Apply setings from the config file
      String lastLibraryFileName =
         configProps.getProperty(PKCS11_LIBRARY_FILE_NAME_KEY);
      if (lastLibraryFileName != null)
         mLibraryFileNameTextField.setText(lastLibraryFileName);
      else
         mLibraryFileNameTextField.setText("");
   } finally {
      configFileStream.close();
   }
}

/**
 * Saves the dialog settings to the dialog configuration file.
 * These settings  consist of a single value - the last used
 * library file name with its full path.
 */

private void saveSettings()
throws IOException {
   // Create a list of settings to store in the config file
   Properties configProps = new Properties();
   String currentLibraryFileName =
      mLibraryFileNameTextField.getText();
   configProps.setProperty(PKCS11_LIBRARY_FILE_NAME_KEY,
      currentLibraryFileName);

   // Save the settings in the config file
   String configFileName = getConfigFileName();
   FileOutputStream configFileStream =
      new FileOutputStream(configFileName);
   try {
      configProps.store(configFileStream, "");
   } finally {
      configFileStream.close();
   }
}

/**
 * @return the library file selected by the user.
 */

public String getLibraryFileName() {
   String libraryFileName = mLibraryFileNameTextField.getText();
   return libraryFileName;
}

/**
 * @return the PIN code entered by the user.
 */

public String getSmartCardPINCode() {
   String pinCode = mPINCodeTextField.getText();
   return pinCode;
}
```

```
    /**
     * Shows the dialog and allow the user to choose library file
     * and enter a PIN code.
     * @return true if the user click sign button or false if the
     * user cancels the dialog.
     */

    public boolean run() {
        try {
            loadSettings();
        } catch (IOException ioex) {
            // Loading settings failed. Default settings will be used.
        }

        setModal(true);
        this.setVisible(true);

        try {
            if (mResult) {
                saveSettings();
            }
        } catch (IOException ioex) {
            // Saving settings failed. Cannot handle this problem.
        }

        return mResult;
    }

    /**
     * File filter class, intended to accept only .dll and .so files.
     */

    private static class LibraryFileFilter extends FileFilter {
        public boolean accept(File aFile) {
            if (aFile.isDirectory()) {
                return true;
            }

            String fileName = aFile.getName().toLowerCase();
            boolean accepted =
                (fileName.endsWith(".dll") || fileName.endsWith(".so"));
            return accepted;
        }

        public String getDescription() {
            return "PKCS#11 v2.0 ot later implementation library " +
                "(.dll, .so)";
        }
    }

}
```

Listing 3: Base64Utils.java

```
/**
 * Provides utilities for Base64 encode/decode of binary data.
 */

public class Base64Utils {

   private static byte[] mBase64EncMap, mBase64DecMap;

   /**
    * Class initializer. Initializes the Base64 alphabet
    * (specified in RFC-2045).
    */

   static {
   byte[] base64Map = {
      (byte)'A', (byte)'B', (byte)'C', (byte)'D', (byte)'E', (byte)'F',
      (byte)'G', (byte)'H', (byte)'I', (byte)'J', (byte)'K', (byte)'L',
      (byte)'M', (byte)'N', (byte)'O', (byte)'P', (byte)'Q', (byte)'R',
      (byte)'S', (byte)'T', (byte)'U', (byte)'V', (byte)'W', (byte)'X',
      (byte)'Y', (byte)'Z',
      (byte)'a', (byte)'b', (byte)'c', (byte)'d', (byte)'e', (byte)'f',
      (byte)'g', (byte)'h', (byte)'i', (byte)'j', (byte)'k', (byte)'l',
      (byte)'m', (byte)'n', (byte)'o', (byte)'p', (byte)'q', (byte)'r',
      (byte)'s', (byte)'t', (byte)'u', (byte)'v', (byte)'w', (byte)'x',
      (byte)'y', (byte)'z',
      (byte)'0', (byte)'1', (byte)'2', (byte)'3', (byte)'4', (byte)'5',
      (byte)'6', (byte)'7', (byte)'8', (byte)'9', (byte)'+', (byte)'/'
};
   mBase64EncMap = base64Map;
   mBase64DecMap = new byte[128];
   for (int i=0; i<mBase64EncMap.length; i++)
      mBase64DecMap[mBase64EncMap[i]] = (byte) i;
   }

   /**
    * This class isn't meant to be instantiated.
    */

   private Base64Utils() {
   }

   /**
    * Encodes the given byte[] using the Base64-encoding,
    * as specified in RFC-2045 (Section 6.8).
    *
    * @param aData the data to be encoded
    * @return the Base64-encoded <var>aData</var>
    * @exception IllegalArgumentException if NULL or empty
    * array is passed
    */

   public static String base64Encode(byte[] aData) {
   if ((aData == null) || (aData.length == 0))
      throw new IllegalArgumentException(
```

```
      "Cannot encode NULL or empty byte array.");

byte encodedBuf[] = new byte[((aData.length+2)/3)*4];

// 3-byte to 4-byte conversion
int srcIndex, destIndex;
for (srcIndex=0, destIndex=0; srcIndex < aData.length-2;
     srcIndex += 3) {
   encodedBuf[destIndex++] =
      mBase64EncMap[(aData[srcIndex] >>> 2) & 077];
   encodedBuf[destIndex++] = mBase64EncMap[
      (aData[srcIndex+1] >>> 4) & 017 |
      (aData[srcIndex] << 4) & 077];
   encodedBuf[destIndex++] = mBase64EncMap[
      (aData[srcIndex+2] >>> 6) & 003 |
      (aData[srcIndex+1] << 2) & 077];
   encodedBuf[destIndex++] = mBase64EncMap[
      aData[srcIndex+2] & 077];
}

// Convert the last 1 or 2 bytes
if (srcIndex < aData.length) {
   encodedBuf[destIndex++] =
      mBase64EncMap[(aData[srcIndex] >>> 2) & 077];
   if (srcIndex < aData.length-1) {
      encodedBuf[destIndex++] =
         mBase64EncMap[(aData[srcIndex+1] >>> 4) & 017 |
            (aData[srcIndex] << 4) & 077];
   encodedBuf[destIndex++] =
      mBase64EncMap[(aData[srcIndex+1] << 2) & 077];
}
else {
   encodedBuf[destIndex++] =
      mBase64EncMap[(aData[srcIndex] << 4) & 077];
   }
}

// Add padding to the end of encoded data
while (destIndex < encodedBuf.length) {
   encodedBuf[destIndex] = (byte) '=';
   destIndex++;
}

String result = new String(encodedBuf);
return result;
}


/**
 * Decodes the given Base64-encoded data,
 * as specified in RFC-2045 (Section 6.8).
 *
 * @param aData the Base64-encoded aData.
 * @return the decoded <var>aData</var>.
 * @exception IllegalArgumentException if NULL or empty data
 * is passed
 */
```

```
   public static byte[] base64Decode(String aData) {
   if ((aData == null) || (aData.length() == 0))
      throw new IllegalArgumentException(
         "Cannot decode NULL or empty string.");

   byte[] data = aData.getBytes();

   // Skip padding from the end of encoded data
   int tail = data.length;
   while (data[tail-1] == '=')
      tail--;

   byte decodedBuf[] = new byte[tail - data.length/4];

   // ASCII-printable to 0-63 conversion
   for (int i = 0; i < data.length; i++)
      data[i] = mBase64DecMap[data[i]];

   // 4-byte to 3-byte conversion
   int srcIndex, destIndex;
   for (srcIndex = 0, destIndex=0; destIndex < decodedBuf.length-2;
         srcIndex += 4, destIndex += 3) {
      decodedBuf[destIndex] = (byte)
         ( ((data[srcIndex] << 2) & 255) |
         ((data[srcIndex+1] >>> 4) & 003) );
      decodedBuf[destIndex+1] = (byte)
         ( ((data[srcIndex+1] << 4) & 255) |
         ((data[srcIndex+2] >>> 2) & 017) );
      decodedBuf[destIndex+2] = (byte)
         ( ((data[srcIndex+2] << 6) & 255) |
         (data[srcIndex+3] & 077) );
   }

   // Handle last 1 or 2 bytes
   if (destIndex < decodedBuf.length)
      decodedBuf[destIndex] = (byte) (
         ((data[srcIndex] << 2) & 255) |
         ((data[srcIndex+1] >>> 4) & 003) );
   if (++destIndex < decodedBuf.length)
      decodedBuf[destIndex] = (byte)
         ( ((data[srcIndex+1] << 4) & 255) |
         ((data[srcIndex+2] >>> 2) & 017) );

   return decodedBuf;
   }

}
```

Listing 4: TestSmartCardSignerApplet.html

```
<html>
```

```
<head>
    <title>Test Smart Card Signer Applet</title>
</head>

<body>
    <form name="mainForm" method="post" action="FileUploadServlet">
        Choose file to upload and sign:
        <input type="file" name="fileToBeSigned">
        <br>
        Certification chain:
        <input type="text" name="certificationChain">
        <br>
        Signature:
        <input type="text" name="signature">
    </form>

    <object
        classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        codebase="http://java.sun.com/update/1.5.0/jinstall-1_5-windows-
i586.cab#Version=5,0,0,5"
        width="130" height="25" name="SmartCardSignerApplet">
        <param name="code" value="SmartCardSignerApplet">
        <param name="archive" value="SmartCardSignerApplet.jar">
        <param name="mayscript" value="true">
        <param name="type" value="application/x-java-applet;version=1.5">
        <param name="scriptable" value="false">
        <param name="fileNameField" value="fileToBeSigned">
        <param name="certificationChainField" value="certificationChain">
        <param name="signatureField" value="signature">
        <param name="signButtonCaption" value="Sign selected file">

        <comment>
            <embed
                type="application/x-java-applet;version=1.5"
                code="SmartCardSignerApplet"
archive="SmartCardSignerApplet.jar"
                width="130" height="25" scriptable="true"

pluginspage="http://java.sun.com/products/plugin/index.html#download"
                fileNameField="fileToBeSigned"
                certificationChainField="certificationChain"
                signatureField="signature"
                signButtonCaption="Sign selected file">
            </embed>
            <noembed>
                Smart card signing applet can not be started because
                Java Plugin 1.5 or newer is not installed.
            </noembed>
        </comment>
    </object>
</body>

</html>
```