UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Avtor seminarja

Aleš Černivec

Naslov seminarske naloge

Infrastruktura javnih ključev

Seminarska naloga pri predmetu
KRIPTOGRAFIJA IN RAČUNALNIŠKA VARNOST

Mentor

izr. prof. dr. Aleksandar Jurišić

Ljubljana: junij, 2010

Kazalo

1 Slovarček izrazov IJK

Podajamo definicijo nekaterih najpogostejših izrazov, ki jih srečamo v literaturi o infrastrukturi javnih ključev (IJK).

Atributni certifikat (AC) podatkovna struktura, digitalno podpisana s strani avtoritete za ravnanje z atributi (Attribute Authority), ki povezuje vrednosti atributov neke entitete z njeno identiteto, torej jamči pristnosti opisanih zmožnostih neke entitete.

Avtoriteta za ravnanje z atributi (Attribute Authority) avtoriteta, ki je zadolžena za izdajanje atributnih certifikatov.

Seznam preklicanih atributnih certifikatov (Attribute CRL, ACRL) seznam preklicanih certifikatov, hrani povezave na atributne certifikate, ki niso več veljavni.

Avtoriteta, agencija entiteta, ki je zadolžena za izdajo certifikatov. V IJK ponavadi nastopa več tipov avtoritet, v tem delu bomo omenjali avtoriteto za ravnanje s certifikati (Certification Authority - CA), ki je zadolžena za izdajanje certifikatov z javnimi ključi in avtoriteto za registracijo (Registration Authority - RA), ki je zadolžena za ugotavljanje kredibilnosti prosilcev certifikatov z javnim ključem.

Certifikat agencije certifikat, ki je bil izdan agenciji (podagenciji) v hierarhiji agencij.

Osnovni CRL seznam preklicanih certifikatov, ki je bil izdan kot osnova za generacijo dCRL.

Delta-CRL (dCRL) delna lista preklicanih certifikatov, vsebuje vnose preklicanih certifikatov, katerih status se je spremenil od referenčnega osnovnega CRL.

Posredni CRL (Indirect CRL - iCRL) seznam preklicanih certifikatov, ki so bili izdani s strani druge certifikatne agencije, ki je izdala omenjeni CRL.

Certifikat CA certifikat agencije, ki je bil izdan s strani druge CA.

Izjava o prakticiranju certificiranja (Certification Practice Statement) izjava o certificiranju, katero spoštuje določena CA pri izdajanju certifikatov z javnimi ključi.

Seznam preklicanih certifikatov (Certifikate Revocation List - CRL) podpisana lista certifikatov, ki niso več veljavni in so bili izdani s strani določene certifikatne agencije. Obstajajo bolj specifični tipi CRL za razliko od splošnega izraza.

Popoln seznam preklicanih certifikatov (full CRL) seznam vseh preklicanih certifikatov neke domene oz. certifikatne agencije.

Validacija certifikata proces, ki zagotavlja veljavnost certifikata ob določenem trenutku. Proces lahko vsebuje potrjevanje in procesiranje celotne poti certificiranja, ki preveri veljavnost vseh certifikatov na poti (certifikati niso pretečeni ali preklicani).

Certifikatna agencija (CA) agencija, ki je zadolžena za kreiranje certifikatov z javnim ključem za uporabnike, ki ji popolnoma zaupajo. CA lahko v izjemnih primerih lahko nudi tudi uporabnikove ključe.

Distribucijska točka CRL direktorij oz. izvorna točka CRLjev. Taka točka lahko nudi CRLje več certifikatnih agencij oz. več vrst CRL v različnih domenah. Ravno v tem smislu prihaja do težav pri distribuciji list CRL.

Navzkrižni certifikati (cross-certificates) certifikati javnih ključev (ali atributnih certifikatov), kjer sta prejemnik in izdajatelj različni certifikatni agenciji (ali atributni agenciji). Navzkrižni certifikati sestavljajo porazdeljen model zaupanja.

Delegacija prenos privilegijev z ene entitete na drugo. Pot delegacije - urejeno zaporedje certifikatov, ki skupaj s certifikatom entitete tvorijo dokaz, ali je ta entiteta vredna zaupanja.

Javni ključ ključ uporabnika, lahko je javno znan.

Certifikat z javnim ključem (Public-Key Certificate - PKC) javni ključ uporabnika skupaj s še nekaj informacijami o izdajatelju in prejemniku certifikata, slednji certifikat je nemogoče ponarediti.

Infrastruktura javnih ključev infrastruktura, ki omogoča upravljanje javnih ključev, podpira avtentikacijo, enkripcijo, integriteto sporočil in storitve, ki onemogočajo ponarejevanje sporočil.

Samo-izdani certifikat certifikat, kjer sta izdajatelj in subjekt enaka (CA). CA ustvari take certifikate v primeru, ko starejšim certifikatom poteče rok veljavnosti in jih je potrebno zamenjati z novejšimi.

Zaupanje zaupanje med dvema entitetama nastane, ko prva entiteta predpostavlja in predvideva obnašanje druge entitete. Zaupanje se lahko nanaša povsem na določeno funkcijo. Ključna vloga zaupanja v IJK je opisovanje relacije med entiteto, ki omogoča avtentikacijo in entiteto, ki je avtenticirana. Slednja popolno zaupa entiteti, ki omogoča izdajanje uporabnih in zanesljivih certifikatov.

2 Okrajšave (akronimi)

AA (Attribute Authority) avtoriteta za izdajanje atributnih certifikatov

AC (Attribute Certificate) atributni certifikat

CA (Certification Authority) avtoriteta za avtenticiranje

CARL (Certification Authority Revocation List) lista preklicanih certifikatov izdana s strani avtoritete za avtenticiranje

CRL (Certificate Revocation List) lista preklicanih certifikatov

dCRL (Delta Certificate Revocation List) delta lista preklicanih certifikatov

OCSP (Online Certificate Status Protocol) protokol za obveščanje o certifikatih

PKC (Public-Key Certificate) certifikat z javnim ključem

PKCS (Public-Key Cryptosystem) kriptosistem s certifikati javnih ključev

PKI (Public-Key Infrastructure) infrastruktura certifikatov javnih ključev (IJK)

3 Uvod

Ker se danes gradijo vedno večji in kompleksnejši računalniški sistemi [1, 2, 3] (porazdeljeni, razpršeni) z vedno večjim številom storitev in še večjim številom uporabnikov različnih organizacij, postaja varnost komunikacije in zaupanje v odločitve posameznih storitev vedno bolj pereč problem.

V tem delu bomo predstavili zgodovino javne kriptografije, sam koncept infrastrukture javne kriptografije (IJK), glavne komponente, ki tak sistem sestavljajo in nenazadnje predstavili tudi načrt za izgradnjo lastne infrastrukture IJK v programskem jeziku Java z uporabo odprto-kodne knjižnice, ki bi se ga dalo vključiti v poljuben sistem. Poleg tega bomo predstavili tudi uporabo sistema OpenSSL in ga z uporabniškega vidika primerjali z Bouncy Castle [18] knjižnico. V delu se bomo dotaknili tudi težjih problemov, kot so možnost porazdelitve posameznih storitev ter zagotavljanje rešive CRL (angl. *Certificate Revocation List*). Delo lahko uporabi razvijalec pri razvoju lastnih varnostnih storitev.

Osredotočili smo se na dva podproblema realizacije infrastrukture: preklic certifikatov (angl. *revocation*) in veriženja certifikatov (angl. *chaining, path validation*). Delo je namenjeno izključno kot referenčna točka, ki uporabniku omogoča vstopno točko pri realizaciji samostojne storitve in jo umestiti v obstoječ sistem.

Predstavili smo že slovarček izrazov IJK in okrajšav. Sledita poglavja, ki sta posvečena sorodni literaturi, opisu IJK in njeni arhitekturi.

V naslednjem poglavju smo opisali problem preklica certifikatov in uporabo CRL. Kot alternativo implementaciji smo predstavili protokol OCSP (angl. Online Certificate Status Protocol).

Sledi poglavje o veriženju certifikatov in predzadnje poglavje o realizaciji minimalne certifikatne agencije z uporabo SSL orodja in namig, kako bi potekala realizacija s knjižnico *Bouncy Castle*.

V zaključku smo zapisali nekaj misli in razmislili o možnih razširitvi dela.

4 Sorodna literatura

Javna kriptografija ima močno matematično podlago. Osnove javne kriptografije bralec lahko najde v delu [4, 5], kjer je podano matematično ozadje in matematična orodja za realizacije osnovnih elementov javne kriptografije. Kljub uporabi matematično dokazanega varnega sistem, se še vedno pojavljajo vprašanja, do kake mere je tak sistem varen [6]. Dejstvo je, da je sistem šibkejši, kakor je močan (bolje šibek) najšibkejši člen. V zadnjem času se pojavljajo nove spletne tehnologije, kar zahteva načrtovanje varnosti storitev z upoštevanjem skalabilnosti, dinamičnosti in odpornosti na napake. Tako so vedno bolj uporabljene storitve ponudnikov t.i. "elastičnih" storitev, ki dopuščajo uporabniku izbor konfiguracije najetih sistemov (angl. Cloud Services) in porazdeljenih storitev [1, 7, 8] (omrežja enakih subjektov).

Predlagana je bila tudi shema porazdeljene sheme strežnikov za streženje informacije o preklicanih certifikatov [8]. V tem delu so predvideni tudi premiki klientov s ključi in prilagajanje porazdeljene sheme trenutni situaciji. Rešitev podpira tudi določeno stopnjo odpornosti na izpade strežnikov in prilagajanje povezanosti klientov na strežnike na podlagi pretoka podatkov s strežnika na klienete in s tem odpravlja težavo preobremenjenosti strežnikov v nekem IJK sistemu.

Zelo splošen opis IJK in vzrokov, zakaj uporaba IJK še vedno ni na pravi stopnji kljub dovolj zgodnjem odkritju IJK, se nahaja v [9]. Avtorji nanizajo pet glavnih faktorjev s katerimi opravičijo izostanek uporabe IJK: nove tehnologije

na področju IJK, politika in vodenje vpeljav IJK s strani velikih institucij, cena vpeljave takšnega sistema in socialni faktorji, ki temeljijo zgolj na napačnih domnevah ljudi. Faktorji so izpostavljeni kot glavni krivci za izostanek vpeljav poleg njih pa so predstavljeni tudi glavni pozitivni elementi in doprinosi, ki jih prinese vpeljava IJK. Članek [9] se zaključi s pogledom v prihodnost, kjer avtorji vidijo razširitev IJK z vpeljavo močnih načinov preverjanja avtentikacije in avtorizacije s pomočjo RBAC sistemov (ang. Roll-based access control), t.j. sistemov za preverjanje vlog in določitev avtorizacije na podlagi vlog preverjenega subjekta. Poleg RBAC bodo vpeljane nove tehnologije, npr. deklarativni jezik za kontrolo dostopa subjektov (ang. eXtensible Access Control Markup Language - XACML), ki bodo pospešili in olajšali uporabo kompleksnejših sistemov za odločanje. XACML [1] je namreč specializacija jezika XML z vpeljavo določenih predefiniranih oznak, s katerimi lahko izrazimo relacije med objekti, ki so uporabljeni v sistemu. Z izrazi v XACML lahko ustvarimo logično plast, ki poveže podatke iz uporabniških (ali drugačnih) certifikatov z drugimi načini opisa pravic.

Poznamo različne načine avtentikacije in kontrole dostopa med posameznimi vozlišči v omrežjih enakovrednih subjektov. Problemi, ki se pojavijo pri IJK v omrežjih enakih subjektov (p2p omrežjih) sta identiteta subjektov in vsebino sporočil med subjekti. Tako poznamo poleg navadnega pristopa vzpostavitve IJK s pomočjo izdaje certifikatov tudi tehnologijo CL-PKC, s katero je moč vzpostaviti javno kriptografijo brez certifikatov [10]. Dejstvo je, da se mora ob vpeljavi novih tehnologij prilagajati tudi infrastruktura, ki omogoča varno, zaupanja vredno komunikacijo.

Problem avtentikacije pri veliko razpršenih certifikatnih agencijah predstavlja resno oviro pri načrtovanju sistema IJK [11]. Preklic certifikatov je predstavljen kot eden izmed glavnih problemov pri izdelavi takega sistema. Težava je pri zbiranju in propagaciji celotne informacije o trenutno aktivnih uporabniških računih vsem agencijam in nuditi na vseh področjih porazdeljenega sistema enako stopnjo varnosti v smislu avtentikacije. Problem preklica uporabniških certifikatov se lahko reši z vpeljavo t.i. **svetilnikov**, ki so zadolženi za zbiranje informacije o pretečenih certifikatov in obveščanje entitetam, ki so prijavljene za prejem teh informacij. Tak sistem je poimenovan **BCPR** – angl. *Beakon Certificate Push Revocation*. Celoten sistem je sestavljen iz naslednjih komponent: Goyeva zgoščevalna funkcija za zagotavljanje granularnosti CA, spremenjen mehanizem CPR (Certificate Push Revocation), svetilnik za zbiranje informacij o pretečenih certifikatov. S pomočjo Goyeve zgoščevalne funkcije se lahko hitro ugotovi, da v nekem obdobju ni bilo preklicanega nobenega certifikata. Ko se ugotovi preklic certifikata, se s pomočjo mehanizma CPR spremembe propagirajo **svetilnikom**. Druge entitete spoznajo spremembe preko svetilnikov. Tako se zagotavlja konsistentnost celotne porazdeljene baze preklicanih certifikatov.

V poglavju 6 bomo opisali knjižnico **Bouncy Castle Crypto**[2], ki združuje orodja za različne programske jezike in z uporabo katere je možno hitreje in z manj napakami realizirati varen način komunikacije in druge načine in mehanizme javne kriptografije (podpisovanje dokumentov, preverjanje veljavnosti certifikatov, realizacija različnih tipov certifikatov, itd.). Na tem mestu je primerno vključiti tudi dela [12, 13], ki uporablja omenjeno knjižnico.

Realizacija standarda PKCS#12 in pripadajoče infrastrukture za izdajanje in preverjanja certifikatov lahko storimo s pomočjo knjižnice Bouncy Castle [12]. Realiziramo lahko tudi novejši koncept uporabe certifikatov, t.i. koncept enkratne prijave – SSO (angl. **Single Sign On**), ki zagotavlja delegacijo storitev v imenu uporabnika, ki se le enkrat prijavi v sistem (z vpisom gesla). Tudi v

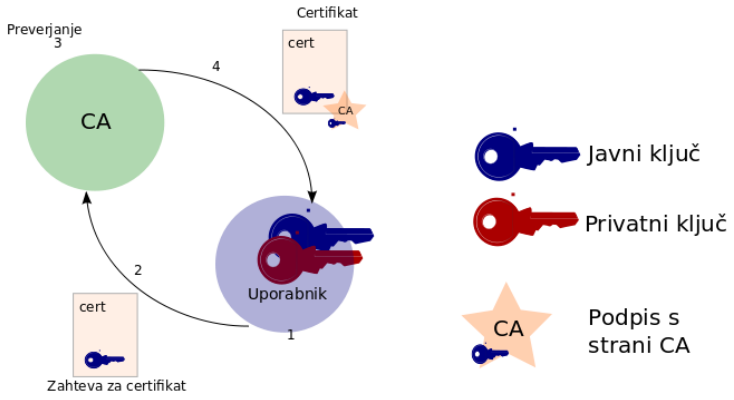
[1] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[2] http://www.bouncycastle.org/

mobilnih komunikacijah je pomembna infrastruktura za zagotavljanje varnosti [13]. V slednjem delu je opisana realizacija sistema za avtentikacijo žetonov na mobilnih telefonih z uporabo "lažje" različice knjižnice Bouncy Castle, prirejene za mobilne naprave.

5 Infrastruktura javne kriptografije (IJK)

Glavni namen javne kriptografije in pripadajoče infrastrukture je upravljanje s ključi in pripadajočimi certifikati. Vsak subjekt v omrežju mora imeti ključ, s katerim se lahko identificira preostalim subjektom v omrežju. Tako bi vsak moral poznati preostale ključe v omrežju, kar skupaj znaša $\frac{n \cdot (n-1)}{2}$ ključev za vseh n subjektov [5]. Za upravljanje ključev, ki predstavljajo identitete subjektov v omrežju, potrebujemo infrastrukturo – infrastrukturo javne kriptografije (IJK). Slednja bi zagotavljala identifikacijo vsakega subjekta v omrežju in omogočala določitev povzročitelja vsake akcije s pomočjo certifikatov: e-pošta (S/MIME), komercialne transakcije (HTTPS), pridobitev in namestitev aplikacij (digitalni podpis izvorne kode oz. paketa). Na žalost danes certifikati (še) niso uporabljeni tako globalno, ker taka infrastruktura, ki bi povezala vse že razvite protokole za varno komuniciranje, ni zastonj.

Certifikat "veže" identiteto subjekta na njegov javni ključ. Identiteta je shranjena v namenskem polju certifikata. Certifikat vsebuje tudi podpis certifikatne agencije (CA)



Slika 1: Generiranje uporabnikovega certifikata.

Na sliki 1 so predstavljeni osnovni koraki pridobitve uporabniškega certifikata certifikatne agencije s strani uporabnika. V koraku 1 uporabnik generira privatni in javni ključ. Z javnim ključem zaprosti certifikatno agencijo za pridobitev uporabniškega certifikata. V koraku 3 CA generira zahtevani certifikat, nakar ga v naslednjem koraku posreduje uporabniku. Uporabnik se sedaj lahko identificira s certifikatom. Ponavadi je v proces pridobitve vključena tudi registracijska avtoriteta (angl. *Registration Authority*), ki je zadolžena za ugotavljanje kredibilnosti subjektov, ki zaprosijo za certifikat.

5.1 Kaj je IJK

Razvijalci internetnih storitev in aplikacij morajo poskrbeti, da bodo njihovi izdelki varni. Ker kriptografija sama ne poskrbi, da je končni izdelek varen, je

nujno, da vsak razvijalec obvladuje tehnologije in teorijo, ki se skriva za izrazom IJK.

Razložimo, kaj pomeni infrastruktura IJK za uporabnika, njene značilnosti (kakor tudi značilnosti kateregakoli varnega sistema). Obsežen sistem, ki uporabnikom nudi (idealno) transparenten način javne kriptografije in omogoča uporabo digitalnih podpisov kot sredstvo za zagotavljanje integritete sporočil, imenujemo infrastruktura javne kriptografije (IJK, infrastruktura PKI, angl. Public-Key Infrastructure). Namen uporabe IJK je ravnanje s certifikati in pripadajočimi ključi, ki zagotavljajo avtentikacijo (proces identifikacije ene entitete drugi z uporabo nekega dokumenta pristnosti - v našem primeru je to kar digitalni certifikat). Avtentikacija je le eden izmed štirih elementov, ki morajo biti del vsakega večjega varnega računalniškega sistema: avtentikacija, avtorizacija, integriteta in - slednji izraz je nekoliko neroden - knjiženja (angl. *accounting*).

Avtentikacija, kot smo že omenili, pomeni proces medsebojne identifikacije entitet, ki med seboj komunicirajo. V literaturi pogosto srečamo tudi izraz medsebojna avtentikacija (angl. *mutual authentication, two-way authentication*).

Pri slednji gre za medsebojno izmenjavo informacije o identiteti med strežnikom in klientom. Pri internetnih storitvah pogostokrat medsebojna avtentikacija pomeni medsebojno izmenjava javnih certifikatov obeh strani.

Avtorizacija ima nekoliko višji pomen in že sega v samo vsebino poizvedbe (komunikacija) med entitetami. Gre za dovoljevanje pravic subjektom na podlagi omejitev, ki so nekako zapisane v sistemu. V različnih sistemih poznamo različne pristope zapisa teh omejitev. Lahko so zapisani v samih certifikatih identitet, atributnih certifikatih [15] ali pa v ta namen uporabljamo posebne storitve. Slednje imajo (lokalno ali porazdeljeno) bazo, kjer so shranjene informacije o posameznih omejitvah uporabnikov glede na pripadnost skupinam, vlogam ali kateri drugi lastnosti uporabnika. Seveda so te lastnosti odvisne tudi od samega sistema (vsak sistem ima svoje pojmovanje o skupinah, vlogah ali katerikoli drugi lastnosti). Preslikovanje identitet uporabnikov in njihovih pripadajočih pravic je poseben primer problema in se v tem delu ne bomo spuščali v podrobnosti.

Integriteta v splošnem pomeni konsistenco med zahtevami in dejanskimi dejanji neke entitete. *Podatkovna integriteta* označuje dejstvo, da so podatki skozi proces prenosa sporočila nespremenjeni. To pomeni, da je sporočilo s, ki ga je oddala entiteta A entiteti B, nespremenjeno prispelo do vstopne točke entitete B. Med postopkom prenosa (na komunikacijskem kanalu) v tem primeru ni prišlo do posredovanja neke tretje entitete C, ki bi spremenila vsebino s. Infrastruktura IJK tako z uporabljeno kriptografsko knjižnico poskrbi, da je sporočilo ustrezno kodirano in podpisano.

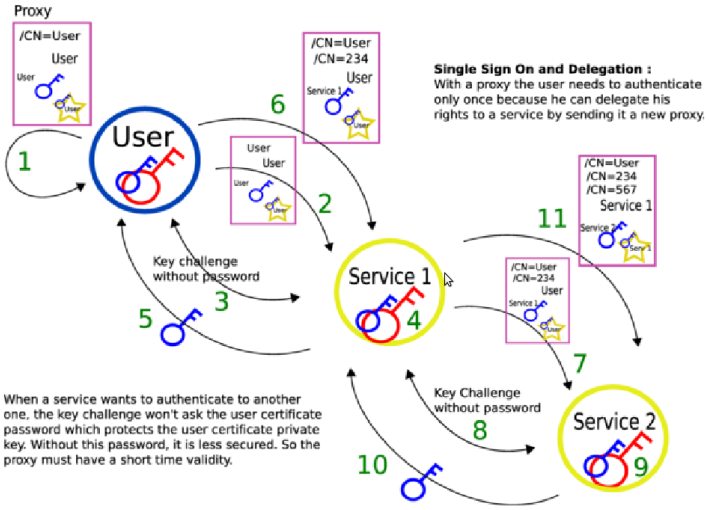
Četrti element je nerodno poimenovano *knjiženje* (angl. accounting), ki pa je zopet poglavje zase in se v tem delu ne bomo spuščali v podrobnosti. Gre namreč za sledenje uporabe virov, do katerih imajo dostop in jih koristijo uporabniki. Gre za sprotno beleženje dogodkov, ki si sledijo časovno in so ponavadi identificirani s peterko:

$$\langle \check{c}as\ za\check{c}etka, \check{c}as\ konca, id\ subjekta, id\ vira, akcija \rangle.$$

S slednjimi informacijami lahko uporabnikom zaračunavamo določeno storitev in so osnova za tako početje. Elemente brez integritete včasih poimenujemo kar trojček AAA (akronim njihovih angleških imen: *Authentication, Authorisation, Accounting*). Ker nas pri realizaciji IJK zanima predvsem identifikacija uporabnikov in zagotavljanje integritete sporočil, nas zanima le "prvi" A (z dodatnimi možnostmi kriptiranja).

Pomembna lastnost IJK je tudi *transparentnost*. Dejstvo je, da uporabnik ne želi vedno več dodatne interakcije s sistemom, če ga posodobi z novimi raz-

širitvami. Če bi se tak princip uporabljal, potem bi bili uporabniki prej ali slej nasičeni s podatki in ne bi želeli uporabljati tako kompleksnega sistema. Tako mora biti IJK transparentna za uporabnika. Na tem mestu je primerno omeniti sistem, ki omogoča enkratno prijavo uporabnika v sistem in delegiranje storitev v njegovem imenu [12]. Omenjeni koncept se imenuje koncept SSO (angl. *Single Sign On*. Ideja koncepta SSO je prikazana na sliki 2.



Slika 2: Koncept SSO in delegacija.

Uporabnik na zahtevo generira t.i. delegacijski certifikat (angl. *proxy certificate*), ki se uporabi v imenu uporabnika in mu tako ni potrebno vedno znova vnašati uporabniškega gesla. Delegacijski certifikati imajo kratko življenjsko dobo (do nekaj ur). Ko storitev 1 (slika 2) zaprosi za delegacijski certifikat, ga uporabnik generira in podpiše javni ključ storitve 1 s svojim privatnim ključem in ga pošlje storitvi 1. Storitev 1 se tako lahko izkaže storitvi 2 z novim certifikatom v imenu uporabnika.

5.2 Osnovni pojmi IJK

V tem poglavju bomo predstavili nekaj osnovnih pojmov infrastrukture IJK. Namen slednje je identifikacija uporabnika še predenj pride do prenosa pomembnejših podatkov med dvema udeleženima subjektoma v komunikaciji. Naj na grobo opišemo, kaj potrebujemo za izgradnjo infrastrukture IJK:

- programsko opremo (implementacijo IJK oz. uporaba javne kriptografije - PKCS),

  odjemalčevo programsko opremo,

  strežniško programsko opremo,

- strojno opremo (strežnike z pripadajočo programsko opremo, npr. CA, AA, RA),

- določene protokole (pravna zagotovila, pogodbe), procedure.

Če se osredotočimo na programsko opremo, saj slednjo nameravamo tudi opisati, lahko ugotovimo, da mora vsaka IJK zagotavljati slednje zahteve (bolj pravilno storitve, ti bodo temelj našega dela):

- izdajo javnih certifikatov, ki vsebujejo javne ključe, identiteto uporabnika, razširjeno tudi druge lastnosti uporabnikov, identiteto CA, ki je izdala certifikat [14]. Med drugimi je potrebno določiti

  format procesiranja

  proces procesiranja

  razdeljevanje certifikatov

- skladišče certifikatov (lahko posreduje tudi kopije certifikatov, ki so jih uporabniki "izgubili", varnostna kopija),
- storitve, ki omogočajo preklic certifikatov,
- samodejno posodabljanje parov ključa in certifikatov,
- vse naštete storitve morajo komunicirati s stranjo klienta na varen način (npr. z uporabo SSL knjižnic),
- politika certificiranja (namen in obseg izdanega certifikata),
- izjava o prakticiranju certificiranja (CPS)

Pomembna je tudi opredelitev uporabnosti certifikatov. Infrastruktura mora imeti komponento (storitev), ki ponuja informacijo o dovoljenih aplikacijah, kjer se certifikati uporabljajo in o aplikacijah, kjer se certifikati ne smejo uporabljati.

5.3 Arhitektura IJK

Tu predstavimo osnovne storitve, ki sestavljajo IJK [16]. Celotno arhitekturo IJK si lahko ogledate na spodnji sliki 3. K sliki spadajo naslednji opisi posameznih storitev.

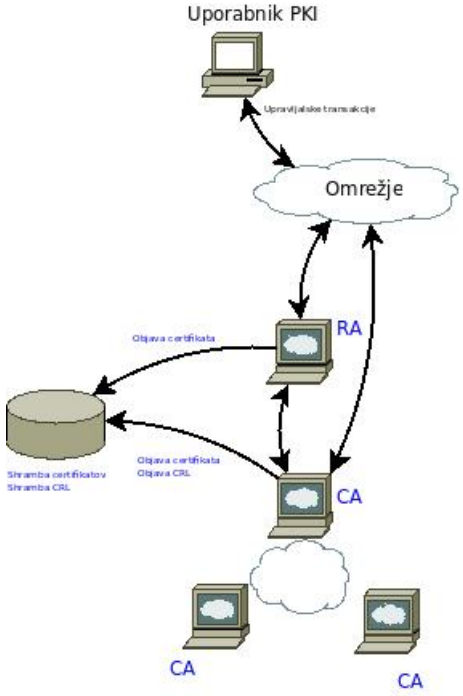
RA (Registration Authority) CA delagira nekatere upravljalske funkcije tej storitvi. Ta ugotavlja kredibilnost subjektov, ki zaprosijo za certifikat. Ugotovi, ali je subjekt upravičen do certifikata (lahko po različnih komunikacijskih poteh). Zadolžena je tudi za rokovanje z zahtevki za preklic in posodobitev/spremenitev certifikatov. RA ne izdaja ali podpisuje certifikatov, slednje je naloga storitve CA.

CA (Certificate Authority) je najpomembnejša komponenta IJK, ki je zadolžena za izdajanje, podpisovanje, vodenje certifikatov, hkrati pa lahko nudi storitve, ki vodijo informacije preklicanih certifikatih (CRL). Lahko je organizirana hierarhično, tako da npr. znotraj velikega podjetja *notranja* CA skrbi za izdajanje certifikatov CA, ki so bolj izpostavljeni, torej so nižji v hierarhiji storitev CA. Lahko so razdeljene tudi po oddelkih ali vejah organizacije, tako da ima vsak oddelek svojo CA za uporabnike znotraj tega oddelka.

Shramba certifikatov in shramba CRL je lahko porazdeljena storitev (baza), ki uporabnikom nudi seznam certifikatov in seznam preklicanih certifikatov (CRL)

CA hrani naslednje podatke v svojih bazah:

- evidenčne podatke uporabnikov certifikatov,
- rok veljavnosti certifikatov,
- seznam preklicanih certifikatov,



Slika 3: Shematski prikaz IJK.

- notranja pravila delovanja (CPS - Certificate Practice Statement),
- direktoriji (dostopna točka za storitev).

Lahko hranijo javne ključe, digitalno podpisane certifikate ali listo preklicanih certifikatov. Direktorji so dostopne točke uporabnikov IJK, kjer lahko na podlagi pravilne zahteve npr. za listo preklicanih certifikatov dobijo povratno informacijo o zahtevku.

Glavne operacije in procesi IJK:

Registracija je proces, ki vključuje tudi RA. V tem procesu se uporabnik avtenticira in poda svoje podatke.

Generiranje ključev na strani uporabnika in CA. Generiranje uporabniškega certifikata je prikazano na sliki 1.

Certificiranje pomeni izdajanje digitalnega certifikata uporabniku s strani CA.

Določitev periode veljavnost vsak certifikat vsebuje informacijo o času veljavnosti izdanega certifikata.

Preklic certifikatov še pred iztekom veljavnosti certifikata CA lahko prekliče veljavnost certifikata z javnim ključem.

Vzdrževanje list preklicanih certifikatov.

6 Realizacija IJK

V tem poglavju bomo raziskali možnosti realizacije osnovnih elementov infrastrukture IJK. Realizirali jih bomo na dva načina: s pomočjo knjižnice OpenSSL

[3] in Bouncy Castle [4]. Prvi način je primeren za uporabo lastne CA in izdajo certifikatov za malo število uporabnikov, medtem ko z drugim pristopom proces lahko avtomatiziramo s pomočjo programskega jezika Java.

6.1 Programski paketi - knjižnice

Predstavimo dva pristopa k realizaciji IJK. Najprej si bomo podrobneje ogledali programsko knjižnico Bouncy Castle in nato odprtokodni program OpenSSL.

Bouncy Castle - implementacija Java kriptografske knjižnice (API) Knjižnica Bouncy Castle je odprtokodna javanska knjižnica za podpisovanje, enkripcijo, razgradnjo sporočil (angl. *message digest*) in podobne funkcionalnosti, ki so nepogrešljive pri implementaciji in uporabi infrastrukture IJK. Sama knjižnica je napisana v večih verzijah: lahka verzija [5] je namenjena manjšim napravam, ki ne premorejo večje procesorske moči, npr. uporaba v J2ME aplikacijah, in je uporaba celotne knjižnice (neokrnjene) predraga; celotna verzija pa podpira vse potrebne funkcionalnosti, ki jih potrebujemo pri implementaciji IJK storitev.

Knjižnica je nastala z iniciativo ljudi, ki so želeli lažji in hitrejši način dostopa vseh funkcionalnosti za delo s certifikati in algoritmi za zagotavljanje varnosti v javanskih aplikacijah. Razredi v knjižnici Bouncy Castle omogočajo uporabo standardnih ali popravljenih razredov za rokovanje z npr. digitalnimi podpisi, preverjanjem vrednosti atributov različnih vrst certifikatov in implementacijo JCE (Java Cryptography Extension) funkcionalnosti. Poleg osnovnih razredov, ki jih ponuja Java za delo z razredi za zagotavljanje varne komunikacije, je API knjižnice Bouncy Castle Crypto za Javo sestavljen iz: "lahkega" API-ja za uporabo na prenosnih napravah (J2ME), ponudnika za Javanske kriptografske razširitve (Java Cryptography Extensions), knjižnico za branje enkodiranih ASN.1 objektov, zelo lahko različico TLS API za mobilne naprave, generatorje za verzije 1 in 3 X509 certifikatov, verzijo 2 list CRL in za rokovanje PKCS#12 datotek in še veliko drugih razširitev.

V naslednjih poglavjih bomo podrobneje opisali dva elementa IJK in sicer sezname preklicanih certifikatov in alternativno storitev, ki omogoča dostop do informacije o preklicanih certifikatih, protokol OCSP (angl. Online Certificate Status Protocol). Poleg tega bomo poskušali opis komponent podpreti s primeri uporabe javanske knjižnice Bouncy Castle.

OpenSSL Cilj odprtokodnega projekta OpenSSL je razviti robustno orodje, ki poleg osnovnih orodij kriptografske knjižnice ponuja uporabo varnostnih protokolov SSL (*Secure Socket Layer*, TLS (*Transport Layer Security*).

6.2 Seznami preklicanih certifikatov, CRL

Liste preklicanih certifikatov so del nekaterih certifikatnih standardov, vendar se jih v nekaterih realizacijah kljub temu razvijalci skušajo izogniti (vsaj tako je bilo še ne dolgo tega), saj po svoji naravi spominjajo na arhaični način ravnanja s številkami čekovnih računov oz. računov kreditnih kartic, ki so jih sem ter tja vpisovali v knjižice ob blagajni trgovin. Tudi CRL so videti kot nekakšni dolgi seznami, ki znajo biti preveč zastareli za uporabo. Torej kako se rokuje s CRL? Na CRL lahko gledamo kot na anti-certifikat, ki izniči veljavnost certifikata. Klient, ki želi preveriti, ali je certifikat veljaven, mora najprej preveriti zaporedno številko certifikata v CRL (torej je bil slednji certifikat preklican).

[3] http://www.openssl.org
[4] http://www.bouncycastle.org

Dejstvo je, da CRL kršijo pravilo kardinalnosti podatkovno-vodenega programiranja: ko v nekem trenutku podamo datum, ga ne moremo več preklicati. Če na izdajo in preklic certifikata gledamo kot na transakcijo, pomeni izdaja certifikata PRIPRAVI (PREPARE) in preklic NAREDI (COMMIT). To pomeni, da med prvim in drugim dejanjem ne moremo delati s certifikati ničesar, saj bi s tem uničili atomičnost in konsistenco transakcije. Če namreč dovolimo dodatne operacije s certifikatom med tema dvema dogodkoma, potem vpeljemo nedeterminizem. Če upoštevamo slednja dejstva, potem lahko preprosto zaključimo, da so "certifikati veljavni do datuma zapadlosti, le če ne izvemo o preklicu kako in kdaj drugače". Več o težavah s CRL je zaslediti v delu [16].

6.2.1 Uporaba OpenSSL

Ko se spremeni kaka lastnost entitete v IJK, ki nastopa z lastnim javnim ključem v certifikatu in ta zaprosi za novejši certifikat, je potrebno starega nekako preklicati. CA v ta namen uporablja listo preklicanih certifikatov (CRL), ki je javno dosegljiva preko strežnika certifikatne agencije. Uporabnik mora periodično preverjati listo preklicanih certifikatov, da lahko pravočasno zavrne certifikate, katerih je potekla veljavnost oz. kateri so se spremenili zaradi kakršnega koli razloga.

Najprej generiramo RSA javni in privatni ključ velikosti 1024 bitov naše storitve z uporabo OpenSSL. z drugim ukazom lahko pogledamo vsebino RSA para ključev.

```
$ openssl genrsa -out CA_key.pem 1024
$ # Preverimo vsebino kljuca
$ openssl rsa -in CA_key.pem -noout -text
```

Ustvarimo lastno podpisani certifikat in ga pregledamo.

```
$ openssl req -new -nodes -x509 -out cert.pem -key CA_key.pem -days 365
$ openssl x509 -text -in CA_cert.pem -noout
```

Nato ustvarimo prazno indeksno datoteko, kamor se bodo osveževali podatki o preklicanih certifikatih.

```
$ touch demoCA/index.txt
$ openssl ca -gencrl -crldays 90 -keyfile demoCA/private/CA_key.pem
-cert demoCA/CA_cert.cer -out demoCA/my_crl.pem
```

Z zgornjim ukazom ustvarimo 90 dni veljaven CRL. Ko veljavnost CRL-ja poteče, ga preprosto ponovno ustvarimo z enakim ukazom. Ko imamo ustvarjen CRL, lahko z njegovo uporabo preprosto prekličemo določen certifikat.

```
$openssl ca -revoke tomcat.cer -keyfile demoCA/private/CA_key.pem -cert
demoCA/CA_cert.cer
```

Ko izvršimo slednji ukaz, se posodobi podatkovna baza preklicanih certifikatov, ki je v zapisana v treh tekstovnih datotekah:

```
./demoCA/index.txt
./demoCA/index.txt.attr
./demoCA/index.txt.old
```

Z ukazom

```
$ openssl crl -text -in demoCA/my_crl.pem
```

pregledamo trenutno listo preklicanih certifikatov in opazimo, da zaenkrat sprememba še ni odražena v CRL datoteki. Potrebno je namreč izvršiti naslednji ukaz:

```
$ openssl ca -gencrl -crldays 90 -keyfile demoCA/private/CA_key.pem -cert
demoCA/CA_cert.cer -out demoCA/my_crl.pem
```

in takoj za tem lahko opazimo razliko v CRL datoteki:

```
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=SI/ST=Slovenija/L=Ljubljana/O=XLAB/OU=Research/CN=Ales
Cernivec/emailAddress=ales.cernivec@xlab.si
        Last Update: Aug 31 19:49:40 2008 GMT
        Next Update: Nov 29 19:49:40 2008 GMT
Revoked Certificates:
    Serial Number: A60FBC09E74F3508
        Revocation Date: Aug 31 19:40:45 2008 GMT
```

Opazimo, da se je spremenil čas zadnje spremembe CRL datoteke in dodal vnos preklicanega certifikata s serijsko številko le-tega.

6.2.2 Uporaba knjižnice Bouncy Castle

Osnovni CRL vsebuje informacijo o CA (izdajatelju CRL), o samem CRL in o certifikatih, ki so že bili preklicani do sedaj. CRL mora vsebovati razširjene atribute, katerih vrednosti govorijo o identiteti certifikata, ki je bil uporabljen za podpis CRL in številka CRL, da bodo klienti lažje prepoznali za katero verzijo CRL gre [9]. Slednja izvorna koda realizirana z uporabo Bouncy Castle knjižnico ustvari CRL s preklicano številko certifikata 1. CRL označimo, da je bil izdan v tem trenutku in hkrati povemo, kdaj mora klient ponovno zaprositi za nov CRL (podamo datum naslednjega osveževanja CRL - **nextUpdate**). Vnos preklicanega certifikata se naredi z uporabo ukaza **addCRLEntry()** z razlogom **privilegeWithdraw**. Kakor že povedano, dodamo tudi dva razširitvena atributa: podatek o ključu avtoritete, ki izdaja CRL in zaporedno številko CRL. Nazadnje CRL tudi izdelamo.

```
import java.security.cert.X509CRL;
import java.security.cert.X509Certificate;

import org.bouncycastle.asn1.x509.CRLReason;
import org.bouncycastle.asn1.x509.CRLNumber;
import org.bouncycastle.asn1.x509.X509Extensions;
import org.bouncycastle.x509.X509V2CRLGenerator;
...
X509V2CRLGenerator   crlGen = new X509V2CRLGenerator();
Date                 now = new Date();
Date                 nextUpdate = ...;
X509Certificate      caCrlCert = ...;
PrivateKey           caCrlPrivateKey = ...;
crlGen.setIssuerDN(new X500Principal("CN=Test CA"));
crlGen.setThisUpdate(now);
crlGen.setNextUpdate(nextUpdate);
crlGen.setSignatureAlgorithm(signatureAlgorithm);
crlGen.addCRLEntry(BigInteger.ONE, now, CRLReason.privilegeWithdrawn);
crlGen.addExtension(X509Extensions.AuthorityKeyIdentifier,
                false, new AuthorityKeyIdentifierStructure(caCrlCert));
crlGen.addExtension(X509Extensions.CRLNumber,
                false, new CRLNumber(crlNumber));
X509CRL    crl = crlGen.generateX509CRL(caCrlPrivateKey, "BC");
```

Ko je potrebno obstoječo CRL posodobiti z novim podatkom o preklicanem certifikatu, uporabimo metodo *addCRL()*:

```
crlGen.setThisUpdate(now);
crlGen.setNextUpdate(nextUpdate);
crlGen.setSignatureAlgorithm(signatureAlgorithm);
crlGen.addCRL(existingCRL);
crlGen.addCRLEntry(BigInteger.valueOf(2), now, CRLReason.privilegeWithdrawn);
crlGen.addExtension(X509Extensions.AuthorityKeyIdentifier,
                  false, new AuthorityKeyIdentifierStructure(caCrlCert));
crlGen.addExtension(X509Extensions.CRLNumber,
                  false, new CRLNumber(crlNumber));
X509CRL    crl = crlGen.generateX509CRL(pair.getPrivate(), "BC");
```

6.3 Protokol za obveščanje o preklicu certifikatov - OCSP

Protokol OCSP omogoča aplikacijam določanje informacije stanja o pretečenosti certifikatom. Za razliko od navadnih list CRL, do katerih dostop omogoča certifikatna agencija, je mogoče z uporabo OCSP časovno bolj pogosto preverjati veljavnost certifikatov in s tem posledično ohranjati informacijo o pretečenosti certifikatov karseda aktualno. Odjemalec OCSP izda zahtevek OCSP strežniku in s tem prestavi sprejetje novega certifikata na trenutek, ko odjemalec dobi odgovor. Za bralca je mogoče zanimiv RFC 2560 [17], kjer je mogoče podrobneje pregledati vsebino zahtevkov in seznam podatkov, ki jih je potrebno izmenjati pri poizvedbi med odjemalcem in strežnikom z uporabljo opisanega protokola.

6.3.1 Poizvedba

Poizvedba OCSP strežniku mora vsebovati naslednje podatke:

- verzijo protokola,
- poizvedbo storitvi,
- identifikacija ciljnega certifikata,
- razširitve kot atributi, ki jih potrebuje strežnik OCSP.

Strežnik najprej preveri pravilnost poizvedbe (ali je sporočilo pravilno formulirano), nato upravičenost poizvedovalca certifikata in nazadnje tudi, ali poizvedba vsebuje vse potrebne podatke. V primeru, če kaka izmed naštetih komponent manjka oz. ni izpolnjena, strežnik OCSP poizvedbo zavrne s sporočilom o napaki.

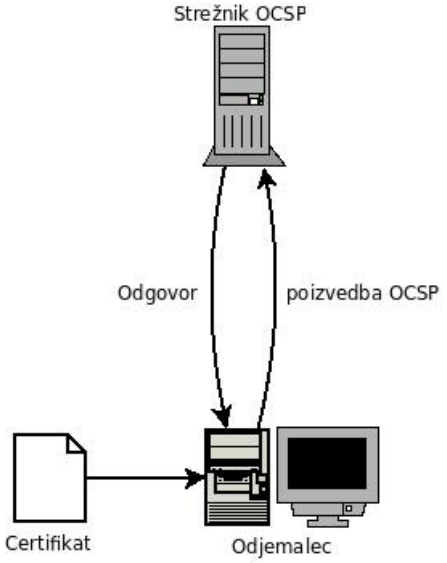
6.3.2 Odgovor

Osnovni odgovor OCSP sestavlja

- verzijo sintakse odgovora,
- ime (DN) strežnika,
- sestavljene odgovore za vsak certifikat v poizvedbi *,
- možne razširitvene atribute,
- OID (angl. Object Identifier) podpisnega algoritma,
- podpis vrednosti zgoščevalne funkcije preko sporočila.

* Odgovor na vsak certifikat, ki je v povpraševanju, mora vsebovati naslednje podatke:

- identifikacijo ciljnega subjekta,



Slika 4: Shematski prikaz protokola OCSP.

- status certifikata,
- časovno periodo veljavnosti certifikata,
- možne razširitvene atribute

Možni odgovori za posamezni certifikat v povpraševanju so torej: **v redu**, **preklican**, **neznano**. Stanje **v redu** pomeni, da je certifikat veljaven. Vendar slednji odgovor še ne povemeni, da je bil certifikat zagotovo izdan ali pa je bil morda odgovor generiran v času veljavnosti, medtem ko je certifikat že potekel oz. bil preklican.

Stanje **preklican** pomeni, da je bil certifikat preklican oz. le začasno preklican.

Stanje **neznano** pomeni, da strežnik ne more odločiti veljavnosti o certifikatu, ki je v poizvedbi.

6.4 Veriženja certifikatov

Da bi povečali varnost, nekatere organizacije uporabljajo veriženje certifikatov. V verigi povezanih certifikatov predstavlja zadnji certifikat korenski certifikat, ki mora biti prisoten, če želimo preveriti avtentičnost certifikata, ki je bil sprejet npr. preko omrežja.

Celotna veriga certifikatov mora biti shranjena v datoteko (npr. **pem** datoteka) ali v posebne datoteke, kjer vsaka zase predstavlja certifikat CA. Če želimo uporabljati veriženje certifikatov, moramo paziti, da je vsak certifikat iz verige shranjen v **skladiču certifikatov**.

6.4.1 Skladišče certifikatov

V dodatku 9.2 prilagamo listing orodij, s kraterimi lahko naložimo certifikate v formatu *PEM* iz nekega direktorija. Vstopna metoda je metoda s podpisom *initiate(String)*, ki prične z nalaganjem certifikatov v pomnilnik. Novo ustvarjeni statični objekt lahko preprosto uporabimo za preveritev, ali je bil certifikat

izdan s strani katerega koli certifikata, ki se nahaja v ustvarjenem objektu *ServiceTrustStore*.

Preveritev novega certifikata lahko naredimo z naslednjimi vrsticami:

```
private ServiceTrustStore sTrustStore;
...
if (sTrustStore.checkValidity(userCtx)){
// Ce je certifikat veljaven
}
else{
// Dodamo izjemo, certifikat ni veljaven.
}
...
```

6.4.2 Uporaba veriženja certifikatov

Veriženje certifikatov se uporablja pri avtentikaciji uporabnikov. Za avtentikacijo uporabnika sta potrebna dva koraka: validacija poti certifikata (veriženje) in validacija ključa.

Vsak certifikat vsebuje podpis certifikatne agencije, s strani katere je bil izdan. S pomočjo javnega ključa, ki se nahaja v certifikatu CA_1, se lahko ta podpis preveri. Uporabnikov certifikat je veljaven, če je veljaven njegov podpis, veljaven mora biti tudi certifikat CA_1. Naprej se preveri, ali je certifikat CA_1 veljaven. Podobno kot smo preverili veljavnost uporabnika, sedaj uporabimo enako pravilo, in preverimo certifikat CA_1 s certifikatom CA_2. Postopek ponavljamo, dokler ne pridemo do korenskega certifikata, ki je samo-podpisan in je neodvisen od preostalih certifikatov CA. Privzeto se slednjemu (korenskemu certifikatu) zaupa.

Preverjanje verižnih certifikatov je potrebno izvršiti preko vseh certifikatov, ki so vključeni na celotni, verižni poti. Če želimo preveriti certifikatno pot lokalno (npr. v neki aplikaciji), je potrebno imeti v *skladišču certifikatov* celotno pot preverjenih certifikatov. Tako lahko celotni paket certifikatov, ki se nahajajo na poti, imenujemo kar *varnostni paket*. Ta paket je potrebno uporabiti pri avtentikaciji, saj mora uporabnik podati entiteti na drugi strani še celotno pot certifikatov. Le tako je možno preveriti avtentičnost uporabnika.

7 Realizacija CA

V naslednjem poglavju bomo opisali nekaj osnovnih orodij, ki jih potrebujemo za realizacijo osnovnih elementov certifikatne agencije: izdajanje certifikata, podpisovanje in preverjanje digitalnega podpisa s pomočjo izdanega certifikata.

7.1 Uporaba OpenSSL

Storitev CA deluje kot certifikatna agencija, ki poljubni storitvi ponuja uporabnikov certifikat ali certifikat poljubne storitve, ki bo služil kot identifikacija subjekta. Predenj se pravilno formirani zahtevi ugodi (**PKCS10CertificationRequest**), se za ravnokar avtenticiranega subjekta preko storitve RA preveri, če obstaja račun za tega subjekta. Tako se na stran klienta prenese veljaven certifikat. Najprej potrebujemo korenski certifikat, ki bo temelj zaupanja celotne hierarhije certifikatnih agencij. Ustvarimo privatni ključ za naš CA, nato pa še certifikat, ki je javni del našega CA. Nastavimo geslo: `krvdemoca`.

```
openssl genrsa -des3 -out demoCA/private/CA_key.pem 2048
openssl req -new -key demoCA/private/CA_key.pem -x509 -days 365 -out
demoCA/CA_cert.cer
```

Naloga CA v IJK je omogočanje porazdeljenega zaupanja. To storimo z izdajanjem in preklicem certifikatov z našim strežnikom. Če želimo implementirati strežnik z uporabo vsebnika v Tomcat-u, potrebujemo tako privatni kot javni ključ (certifikat), ki bosta osnova zaupanja CA. Generiranje privatnega RSA ključa za naš strežnik. Geslo: `krvdemotomcat`.

```
openssl genrsa -des3 -out tomcat.pem 2048
```

S slednjim ukazom ustvarimo novi RSA ključ, ki je zaščiten z DES3 geslom. Ker želimo, da lahko z uporabo CA izdajamo certifikate z javnim ključem, je potrebno ustvariti zahtevek za podpis certifikata (CSR - angl. Certificate Signing Request) iz našega privatnega ključa. Z uporabo OpenSSL knjižnice je to preprosto:

```
$ openssl req -new -key tomcat.pem -out tomcat_request.csr
Enter pass phrase for tomcat.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:SI
State or Province Name (full name) [Some-State]:Slovenija
Locality Name (eg, city) []:Ljubljana
Organization Name (eg, company) [Internet Widgits Pty Ltd]:XLAB
Organizational Unit Name (eg, section) []:Research
Common Name (eg, YOUR name) []:ales.cernivec@xlab.si
Email Address []:ales.cernivec@xlab.si

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:krvdemo
An optional company name []:XLAB
ales@ales-laptop-lenovo:~/Documents/faks/1letnik/KRV/seminar/source/RootCA$
ls
demoCA  tomcat.pem  tomcat_request.csr
```

CSR datoteka predstavlja zahtevek CA za generiranje javnega ključa (certifikata). Seveda je ta zahtevek vezan na privatni ključ (v našem primeru je to **tomcat.pem**). Potrebno je vnesti dodatne informacije o uporabniku, ki izdaja CSR, saj je certifikat izdan v obliki CSRja. Certifikat CA kasneje le podpiše z svojim privatnim ključem in vrne certifikat izdajatelju CSRja.

Ko CA sprejme CSR datoteko, je potrebno preveriti uporabnika, ki je izdal CSR datoteko. Proces je lahko dolgotrajen in formalen (npr. CA povratno obvesti izdajatelja za dodatne informacije, tu ima ključno vlogo RA, ki je opisana kasneje). Po tem, ko smo prepričani o avtentičnosti izdajatelja, potrdimo zahtevek z izdajo certifikata. Z uporabo javnega dela in privatnega dela CA-ja (javni in privatni ključ) podpišemo in izdamo javni certifikat, ki je vsebovan v CSR-ju:

```
$ openssl x509 -req -days 30 -in tomcat_request.csr -CA demoCA/CA_cert.cer
-CAkey demoCA/private/CA_key.pem -CAcreateserial -out tomcat.cer
```

S slednjim ukazom izdamo nov certifikat, ki je podpisan s strani CA z veljavnostjo 30 dni. Pozorni moramo biti na parameter **CAcreateserial**, ki omogoča enolično kreiranje zaporedne številke certifikata. Tako se ustvari nova datoteka `demoCA/CA_cert.srl`, ki vsebuje številko 02, kar predstavlja naslednjo številko, ki bo prirejena naslednjemu certifikatu, ki bo izdan s pomočjo tega CA. Za

nadaljnje izdajanje certifikatov moramo uporabljati na strani CA nekoliko spremenjen ukaz (eksplicitno podamo datoteko, ki narekuje uporabo zaporednih številk certifikatov):

```
$ openssl x509 -req -days 365 -in new_request.csr -CA demoCA/CA_cert.cer
-CAkey
demoCA/private/CA_key.pem -CAserial demoDA/CA_cert.srl -out
new_certificate.cer
Ko ustvarimo certifikat, ga lahko po"sljemo nazaj uporabniku, ki je izdal
zahtevo za certifikat. Z uporabo ukaza
$ openssl x509 -text -in tomcat.cer -noout
Lahko pregledamo vsebino novonastalega certifikata. Za nas zanimiv del
certifikata je slednji:
Version: 1 (0x0)
        Serial Number:
            a6:0f:bc:09:e7:4f:35:08
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=SI, ST=Slovenija, L=Ljubljana, O=XLAB, OU=Research, CN=Ales
Cernivec/emailAddress=ales.cernivec@xlab.si
        Validity
            Not Before: Aug 31 18:08:15 2008 GMT
            Not After : Sep 30 18:08:15 2008 GMT
        Subject: C=SI, ST=Slovenija, L=Ljubljana, O=XLAB, OU=Research,
CN=ales.cernivec@xlab.si/emailAddress=ales.cernivec@xlab.si
        Subject Public Key Info:
```

Opazimo, da je vsebina polja CN izdajatelja enaka CN certifikatne agencije, vsebina polja CN subjekta pa so podatki, ki smo jih vnesli ob kreiranju CSR datoteke.

Podpisovanje podatkov in preverjanje podpisov Zelo pomembno orodje pri uporabi kriptografskih knjižnic je orodje za podpisovanje podatkov. Z njim lahko kreiramo digitalni podpis, ki ga prejemniku npr. pošljemo skupaj s podatki. Prejemnik tako lahko z uporabo certifikata pošiljatelja preveri podpis, če se ujema s podatki, ki so bili poslani. Tako zagotovimo integriteto sporočila. Kot primer si oglejmo vhodno datoteko testni.data z naslednjo vsebino:

```
$ more testni.dat
Kdor ne najde poti, najde izgovor.
```

Iz ukazne vrstice lahko podpišemo vhodno datoteko z ukazom `openssl dgst`:

```
$ openssl dgst -sha1 -sign tomcat.pem -out testni.sign testni.data
```

in tako kreiramo digitalni podpis `testni.sign`. Vsebina te datoteke nam brez ponovne uporabe kriptografske knjižnice ne pove prav veliko, vsebuje pa podpisani generirani podpis. Podpis je narejen s pomočjo zgoščevalne funkcije **sha1**. Preverjanje podpisanega dokumenta je enostavno in podobno podpisovanju. V roki moramo imeti le javni ključ pošiljatelja in izvršiti naslednji ukaz (vrstico nižje je odgovor, da je podpis pristen):

```
$ openssl dgst -verify tomcat.pub -signature podpis.sign testni.data
Verified OK
```

Če imamo v roki le certifikat tipa `x509`, je potrebno pred tem izluščiti javni ključ. To lahko storimo z naslednjim `x509` ukazom:

```
$ openssl x509 -pubkey -in tomcat.cer -noout > tomcat.pub
```

7.2 Kriptografska knjižnica Bouncy Castle

Knjižnica Bouncy Castle je podrobneje opisana v poglavju 6.1. Za podpisovanje, branje in preverjanje podpisa, so vsi potrebni razredi podprti že s strani knjižnice JCE (angl. *Java Cryptography Extension*).

Kreiranje zahtevka klienta za izdajo certifikata Razred **PKCS10CertificationRequest** je osnova za kreiranje zahtevka za pridobitev javnega certifikata s pomočjo knjižnice Bouncy Castle.

Listing 1: Izdaja zahtevka

```
PKCS10CertificationRequest certRequest = new PKCS10CertificationRequest(
        "SHA256withRSA",
        new X500Principal(""),
        keyPair.getPublic(),
        null,
        keyPair.getPrivate());
```

S prvim parametrom izberemo algoritem, s katerim bo certifikat podpisan, drugi parameter določa izdajatelja certifikata (lahko je enak CN naše CA, lahko pa kar prazen kot v našem primeru, to polje se lahko upošteva na strani CA-ja). Spremenljivka `keyPair` je tipa `java.security.KeyPair` in hrani javni in privatni ključ našega zahtevka. Javni in privatni ključ se ustvarita na strani klienta, medtem ko se na strani CA, torej na strani strežnika le doda podpis CA in tako potrdi avtentičnost certifikata, ki je poslan nazaj uporabniku.

Podpisovanje in preverjanje podpisa Tu predstavimo način podpisovanja in preverjanja podpisa nekega Javanskega objekta. Najprej nastavimo spemenljivko `data`, ki drži celotno vsebino podatkov v obliki polja tipa `byte`. Instanca objekta `signer` je zadolžena za hrambo digitalnega podpisa podatkov, ki se nastavi ob klicu medote `signer.update(data)`. Kako dobimo vrednost digitalnega podpisa, ki ga na koncu hrani spremenljivka `digitalSignature`,

Listing 2: Primer digitalnega podpisa.

```
final Signature signer = Signature.getInstance("SHA256WithRSAEncryption");
signer.initSign(key);
// Convert data to byte array
final ByteArrayOutputStream baos = new ByteArrayOutputStream();
final ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(suitable_resources);
oos.flush();
final byte[] data = baos.toByteArray();
// Sign the data
signer.update(data);
digitalSignature = (signer.sign());
```

V listingu 3 je prikazana uporaba javanskega razreda za preverjanje digitalnega podpisa. Spremenljivka `encrHashStr` vsebuje podpis, ki ga bomo preverjali z javnim ključem pošiljatelja. S klicem `signature.update(origStringBytes)` izračunamo vrednost podpisa, ki ga nato preverimo z želenim. Preverimo ga v zadnji vrstici listinga 3.

Listing 3: Preverjanje digitalnega podpisa.

```
byte[] encrHash = hexToBytes(encrHashStr);
byte[] origStringBytes = stringToBytes(origString);
Signature signature = Signature.getInstance(alg);
signature.initVerify(pubkey);
signature.update(origStringBytes);
boolean retval = signature.verify(encrHash);
```

V prilogi kot listing 4 prilagamo tudi realizacijo knjižnice, ki uporabniku olajša delo s certifikati. Razred `Utils` ponuja metode za generiranje javnega in privatnega ključa, branje uporabnikovega in atributnega certifikata, branje

privatnega ključa iz datoteke `PEM` z branjem uporabnikovega gesla. Omogoča tudi izpis para ključev v datoteko in zpis in branje certifikatov v/iz datoteke.

Priloga v poglavju 9.2 (listing 5), prikazuje realizacijo branja certifikatov iz določenega direktorija in preverjanja, ali uporabnikov certifikat pripada kateremu iz tega skladišča (v smislu veriženja certifikatov).

8 Zaključek

V tem delu smo podali bralcu nekaj napotkov in definicije osnovnih pojmov IJK. Bralca smo opremili tudi z orodji in osnovnim znanjem, kako se lotiti implementacije IJK s pomočjo knjižnice Bouncy Castle oz. knjižnice OpenSSL. Predstavili smo tudi nekaj problemov, ki nastopajo pri konceptu in realizaciji certifikatnih agencij v IJK - realizacija CRL. Slednji je pereč problem (v svojem bistvu) že kar dve desetletji in očitno ostaja tudi danes, samo dokler se ne osredotočimo na ožji kontekst (manjšo domeno), npr. organizacijo saj je vzdrževanje list preklicanih certifikatov težak problem. Predstavili smo tudi alternativo CRL, to je mrežni protokol OCSP. Slednji zagotavlja boljši pristop k preverjanju preklicanih certifikatov, vendar tudi ne povsem transparentno.

Kot nadaljnje delo bi lahko uporabili zbrano dosedanje znanje in se lotili implementacije dejanskega strežnika CA, ki uporablja protokol OCSP in ga ponudili kot odprto orodje.

9 Dodatek

9.1 Orodje za delo s certifikati

V tem listingu je priložena koda, ki smo jo napisali za delo s certifikati. Slednja knjižnica, ki uporablja knjižnico Bouncy Castle, omogoča generiranje para ključev (privatni, javni), branje datotek PEM, branje certifikatov X509 iz datoteke, pisanje certifikatov v datoteko, pisanje javnih ali privatnih ključev na izhodni tok podatkov.

Listing 4: Realizacija osnovnih metod za delo s certifikati.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.cert.X509Certificate;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import org.bouncycastle.openssl.PEMReader;
import org.bouncycastle.openssl.PEMWriter;
import org.bouncycastle.openssl.PasswordFinder;
import org.bouncycastle.x509.X509AttributeCertificate;
import org.bouncycastle.x509.X509V2AttributeCertificate;

public class Utils {

        final protected static String COMMA = ",";
        protected static String CREDENTIAL_SEPERATOR = ",";

 public static KeyPair generateKeyPair(String algorithm, int keylen)
 throws NoSuchAlgorithmException {
```

```
 KeyPairGenerator kpGen = KeyPairGenerator.getInstance(algorithm);
 SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
 byte[] bytes = new byte[20];
 random.nextBytes(bytes);
 kpGen.initialize(keylen, random);
 return kpGen.generateKeyPair();
}

public static class MyPassword implements PasswordFinder {
 private char[] password;
 public MyPassword(char[] password) {
  this.password = password;
 }
 @Override
 public char[] getPassword() {
  return password;
 }

}

public static String findSubjectAltName(Collection subjectAltNames, int nameType) {
 String subjectAltName = null;
 for (Iterator i = subjectAltNames.iterator(); i.hasNext(); ) {
  List item = (List)subjectAltNames.iterator().next();
  Integer type = (Integer)item.get(0);
  if (type == nameType)
   subjectAltName = (String)item.get(1);
  break;

 }
 return subjectAltName;
}

public static X509Certificate readX509Certificate(String keyFilename, char[] password)
throws FileNotFoundException, IOException {
 Object readResult = readPEM(keyFilename, password, "X509Certificate");
 return (X509Certificate) readResult;

}

public static X509V2AttributeCertificate readX509Certificate(
   String keyFilename)
throws FileNotFoundException, IOException {
 Object readResult = readPEM(keyFilename, new char[0],
 "X509V2AttributeCertificate");
 return (X509V2AttributeCertificate) readResult;

}

public static KeyPair readKeyPair(String keyFilename, char[] password)
throws FileNotFoundException, IOException {
 return (KeyPair) readPEM(keyFilename, password, "KeyPair");

}

public static Object readPEM(String keyFilename, char[] password, String type)
throws FileNotFoundException, IOException {
 PEMReader reader = null;
 Object o = null;
 PasswordFinder myPass = new MyPassword(password);
 Arrays.fill(password, ' '); // Should only do in caller?
 reader = new PEMReader(new InputStreamReader(new FileInputStream(keyFilename)),
   myPass);
 o = reader.readObject();
 reader.close();
 return o;
}

public static X509Certificate readCertificate(InputStreamReader isr)
throws IOException {
 return (X509Certificate) readPEM(isr);
}

public static Object readPEM(InputStreamReader isr)
throws IOException {
 PEMReader preader = null;
 Object o = null;
 preader = new PEMReader(isr);
 o = preader.readObject();
 if (o == null) {
  throw new IOException("Read a NULL PEM Object");
```

```
  }
  return o;
 }

 public static void writeKey(OutputStream os, PrivateKey key, String alg, char[] password) {
  PEMWriter keyWriter = null;
  try {
   keyWriter = new PEMWriter(new OutputStreamWriter(os));
   keyWriter.writeObject(key, alg, password, new SecureRandom());
   keyWriter.flush();
  } catch (IOException e) {
   System.err.println("writeKey" + e);
  }
 }

 public static void writeKey(OutputStream os, PrivateKey key) {
  PEMWriter keyWriter = null;
  try {
   keyWriter = new PEMWriter(new OutputStreamWriter(os));
   keyWriter.writeObject(key);
   keyWriter.flush();
  } catch (IOException e) {
   System.err.println("writeKey" + e);
  }
 }

 public static void writeCertificate(OutputStreamWriter osw, X509Certificate cert)
 throws IOException {
  PEMWriter pemWriter = null;
  try {
   pemWriter = new PEMWriter(osw);
   pemWriter.writeObject(cert);
   pemWriter.flush();
  } catch (IOException e) {
   System.err.println("writeCertificate" + e);
  }
 }

 public static void writeCertificate(OutputStreamWriter osw,
    X509AttributeCertificate cert)
 throws IOException {
  PEMWriter pemWriter = null;
  try {
   pemWriter = new PEMWriter(osw);
   pemWriter.writeObject(cert);
   pemWriter.flush();
  } catch (IOException e) {
   System.err.println("writeCertificate" + e);
  }
 }
}


public static Object readPEM(String keyFilename, char[] password, String type)
 throws FileNotFoundException, IOException {
 PEMReader reader = null;
 Object o = null;
 PasswordFinder myPass = new MyPassword(password);
 Arrays.fill(password, ' '); // Should only do in caller?
 reader = new PEMReader(new InputStreamReader(new
 FileInputStream(keyFilename)),myPass);
 o = reader.readObject();
 reader.close();
 return o;
}
```

9.2 Skladiščenje certifikatov

V tem listingu je priložena koda, ki smo jo napisali za shranjevanje certifikatov.

Listing 5: Realizacija skladiščenja certifikatov.

```
/**
 * Necessary imports
 */
import java.io.File;
...
```

```
/**
 * Provides tools for checking the validity of the certificates inside
 * service calls.
 *
 * @author ales.cernivec@xlab.si
 */
public class ServiceTrustStore {
 static Logger logger = Logger.getLogger(ServiceTrustStore.class);
 private TrustManager[] trustManagers = null;
 private static KeyStore trustStore = null;

 /**
  * Reads a directory for PEM certificates and ads them to hashmap (with
  * aliases). If there exist private keys or files other than valid public
  * certificates (X509Certificate)
  *
  * @return hash map of certificates.
  */
 private static HashMap<String, X509Certificate> getCertMap(File policyStorageDirectory)
 throws Exception{
  logger.debug("Entering - ServiceTrustStore.getCertMap");
  HashMap<String, X509Certificate> certMap = new HashMap<String, X509Certificate>();
  File[] allFiles = policyStorageDirectory.listFiles();
  if ((allFiles == null) || (allFiles.length == 0)) {
   logger.error("Truststore seems to be empty or non-existing: " +
     (policyStorageDirectory.isAbsolute() ? "" :
      System.getProperty("user.dir") + File.separator) +
      policyStorageDirectory.toString());
  } else {
   for(File file:allFiles)
   {
    if(file.getName().toLowerCase().endsWith(".pem")){
     X509Certificate cert = null;
     boolean err = false;
     try{
      cert= Utils.readX509Certificate(file.getAbsolutePath(),
        "".toCharArray());
     }
     catch(ClassCastException cce){
      // There may be KeyPair also inside trustore.
      err = true;
      logger.debug("[ServiceTrustStore.certMap] There " +
        "is instance of KeyPair in trustore." + cce);
     }
     catch(IOException ioe){
      //
      err = true;
      logger.debug(ioe);
     }
     if (!err)
      certMap.put(file.getName(), cert);
    }
   }
  }
  logger.debug("Exiting - ServiceTrustStore.getCertMap");
  return certMap;
 }

 /**
  * Initiates KeyStore with certificates, provided with certMap parameter.
  *
  * @param HashMap containing aliases and public certificates.
  */
 private static KeyStore initiateTrustStore(HashMap<String, X509Certificate> certMap)
 throws Exception{
  logger.debug("Entering - ServiceTrustStore.initiateTrustStore");
  KeyStore trustStore = KeyStore.getInstance("JKS");
  trustStore.load(null);
  for(String alias :certMap.keySet()){
   X509Certificate cert = certMap.get(alias);
   trustStore.setCertificateEntry(alias, cert);
  }
  logger.debug("Exiting - ServiceTrustStore.initiateTrustStore");
  return trustStore;
 }

 /**
  * Calls {@link #getCertMap(File)} and {@link #initiateTrustStore(HashMap)}
  * respectively.
  *
  * @param pathToTrustStore path to keystore.
```

```
 */
 public static void initiate(String pathToTrustStore) throws Exception{
  logger.debug("Entering - ServiceTrustStore.initiate");
  Security.addProvider(new BouncyCastleProvider());
  HashMap<String, X509Certificate> certMap = getCertMap(new File(pathToTrustStore));
  trustStore = initiateTrustStore(certMap);
  logger.debug("Exiting - ServiceTrustStore.initiate");
 }

 /**
  * Method checks validity of the clientCertificate cert with the KeyStore
  * initiated using {@link #initiateTrustStore(HashMap)} method.
  *
  * @param trustedServerCertificate
  * @param clientCertificate
  * @return
  * @throws Exception
  */
 public static boolean checkValidity(X509Certificate clientCertificate)
 throws Exception{
  //Iterator<String> itAliases=trustStore.aliases();
  clientCertificate.checkValidity();
  Enumeration<String> enumAliases = trustStore.aliases();
  boolean isValid = false;
  while(enumAliases.hasMoreElements()){
   X509Certificate cert = (X509Certificate)trustStore.getCertificate(enumAliases.nextElement());
   try{
    clientCertificate.verify(cert.getPublicKey());
    isValid = true;
    break;
   }
   catch(Exception err){
    logger.error(err);
   }
  }
  return isValid;
 }

 /**
  * @param args
  */
 public static void main(String[] args) throws Exception {
  Security.addProvider(new BouncyCastleProvider());
  HashMap<String, X509Certificate> certMap = getCertMap(new File(localTrustStore));
  initiateTrustStore(certMap);
  X509Certificate clientCert = Utils.readX509Certificate(clientCertificateFile,"".toCharArray());
  try{
   if(checkValidity( clientCert))
    System.out.println("Certificate is valid.");
   else
    System.out.println("Certificate is NOT valid!");
  }
  catch(Exception err){
   System.err.print("Certificate is not valid:" + err);
  }
 }
}
```

Literatura

[1] XtreemOS Consortium, First Specifications of Securiy Services, D3.5.3, Maj 2007.

[2] M. Coppola, Y. Ygou, B. Matthews, C. Morin, L. P. Prieto, O. D. Sanches, E. Y. Yang, H. Yu, Virtual Organization Support within a Grid-Wide Operating System, *IEEE Internet Computing* **16** (2008), 20–28.

[3] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS **3779** (2006), 2–13.

[4] W. Diffie, M. Hellman, New Directions in Cryptography,*IEEE Transactions on Information Theory* (1976), 644–654.

[5] R. L. Rivest, A. Shamir, L. M. Adelman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Commun. ACM 21* **2** (1978), 388–392.

[6] C. Ellison, B. Schneier, Ten Risks of PKI: What You are not Being Told about Public Key Infrastructure, *Computer Security Journal* **16** (2000), 1–7.

[7] H. Mikkonen, K. Happonen, J. Hahkala, Online Host Certificate Registration and Renewal System, *Proceedings of the 2009 International Conference on Grid Computing*, julij 2009

[8] S. Mista et al, Dividing PKI in Strongest Availability Zones, *Proceedings of the 7th ACS/IEEE International Conference on Computer Systems and Applications* (2009), 10–13.

[9] H.G. Miller, J.L. Fisher: Requiring Strong Credentials: What's Taking So Long?, *IEEE Educational Activities Department*, IT Professional **1** (2001), 57–60.

[10] Z. Li, X. Xu, at al: Authentication in Peer-to-Peer Network: Survey and Research Directions, IEEE, *Third International Conference on Network and System Security* (2009), 115–122.

[11] Solworth, J.A.: Beacon certificate push revocation, ACM, *Proceedings of the 2nd ACM workshop on Computer security architectures* (2008), 59–66.

[12] Y. Peng: The Application of PKCS12 Digital Certificate in User Identity Authentication System, *World Congress on Software Engineering*, IEEE Computer Society, 2009.

[13] P. Kunyu, Z. Jiande, Y. Jing, An identity authentication system based on mobile phone token, *Network Infrastructure and Digital Content* (2009), 570–575.

[14] R. Housley et al, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RSA Laboratories, April 2002.

[15] S. Farrell, R. Housley, RFC3281 – An Internet Attribute Certificate Profile for Authorization, RSA Laboratories, April 2002.

[16] The Office of the Federal Privacy Commissioner, Privacy and Public Key Infrastructure: Guidelines for Agencies using PKI to communicate or transact with individuals, whitepaper, december 2001.

[17] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, http://www.ietf.org/rfc/rfc2560.txt, junij 1999.

[18] D. Hook, Beginning Cryptography with Java, Wiley (Indianapolis) 2005.