# Digitalni certifikati in agencija za izdajo digitalnih certifikatov (CA)

Miha Vuk

12. maj 2001

**Povzetek**

Pri vseh asimetričnih kriptografskih sistemih se pojavi problem, kako preveriti identiteto lastnika javnega ključa. Certifikati rešujejo ta problem na način, da dodamo identiteto zraven k javnemu ključu in vse skupaj podpiše nekdo, ki mu zaupaš (agencija CA). Po pregledu splošnih načinov uporabe si bomo podrobneje ogledali razne vrste certifikatov in njihovo uporabo. Nato se bomo posvetili zgradbi. Opisali bomo abstraktno strukturo certifikata in nato način kako to pretvoriti v niz bytov. Na koncu bom še opisal program, za generiranje certifikatov, ki sem ga napisal in podrobno razložil njegovo uporabo, za morebitne kasnejše uporabnike.

# 1 Uvod

## 1.1 Simetrični in asimetrični sistemi in prenos ključev

Že dolgo ljudje uporabljajo razne načine šifriranja, ki temeljijo na simetrični kritografiji. To pomeni, da pošiljtelj in sprejemnik poznati isti ključ (ali šifrirni postopek), ki mora biti zato tajen. Pred komunikacijo si morata ta ključ na nek varen način izmenjati, kar pa ni vedno možno ali enostavno. Pri izmenjavi lahko tudi preverita identiteto sogovornika.
Leta 1976 sta se Whit Diffe in Martin Hellman domislila, da bi za šifriranje in dešifriranje uporabila dva različna ključa. Eden od obeh bi bil javen, drugi pa skriven (privaten) in bi ga imel le uprabnik (lastnik). Temu pravimo asimetričen sistem. Očitno je sicer, da je možno sporočila dešifrirati tudi brez skrivnega ključa (npr. s poskušanjem), kot tudi odkriti skrivni ključ, vendar je dober asimetričen sistem zasnovan tako, da je to računsko prezahtevno (skoraj nemogoče).
S takim sistemom je mogoče sporočila tudi podpisovati (tako, da vsebina ostane javna). Če sporočilo zašifriramo z našim privatnim ključem, se ga lahko preveri (oz. dešifriramo) le z javnim. To pa lahko stori vsak prejemnik. Običajno sporočilo zgostimo z zgoščevalno (hash) funkcijo in nato podpišemo le ta izvleček. Podpis dodamo originalnemu sporočilu, kar omogoča prejemniku, da spet izračuna izvleček in preveri podpis.

## 1.2 Problem distribucije ključev in (pod)problem identitete

Privatni ključ ima le lastnik, javni pa mora biti dostopen vsem ali vsaj tistim, s katerimi lastnik dopušča komunikacijo. Če si sogovornika izmenjata ključe po varnem kanalu, običajno sproti preverita še sogovornikovo identiteto, sicer pa je o njen vedno kanček dvoma. Med osebi A in B, ki se želita pogovarjati, bi se namreč lahko vrinil sovražnik C. A-ju bi se predstavil kot B, B-ju pa kot A. A in B bi mislila, da se pogovarjta en z drugim, v resnici pa bi A-jeva sporočila C dešifriral, jih nato spet zašifriral z drugim ključem ter poslal B-ju. Vsem sporočilom bi prisluškoval, lahko pa bi jih tudi spreminjal. Temu pravimo aktivni sovržnik.
Problem se torej glasi: Osebi A in B, ki se ne poznata in še nista komunicirali, bi želeli varno komunicirati. Kako si izmenjati izmenjati ključe?

## 1.3 Znane rešitve

Poznamo rešitve, ki temeljijo na simetričnih in na asimetričnih sistemih. Vsem pa je skupnih nekaj značilnosti.
Ker se osebi A in B še ne poznata, morata poznati neko skupno osebo in ji zaupati, da bo jamčila za identito obeh (angleško Trusted Authority (TA)). Poleg tega mora imeti vsaka oseba, določene atribute, po katerih se sogovornik odloči ali se želi z njo pogovarjati (ali je to sploh prava oseba). Taki atributi so: ime, primek, naslov, funkcija ali poklic,... Lahko gre tudi za navidezne osebe kot so: računalniki, programi, web strani,... Navidezne osebe imajo seveda precej drugačne atribute. TA mora zagotoviti, da se določen ključ res nanaša na neko osebo (opisano z atributi).

### 1.3.1 Kerberos

Ta temelji na simetričnih sistemih. Vsaka oseba A si z TA deli ključ $K_A$, tako da se lahko z njo varno pogovarja. Ko se A želi pogovarjati z B, se pri TA najprej pozanima, če je to res željena oseba. Nato zaprosi za sejni ključ. TA izda ključ $K_{AB}$, ki bo veljal le za ta pogovor. Pošlje ga (varno s pomočjo $K_A$ oz. $K_B$) A-ju in B-ju. Zraven doda še čas veljavnosti in lahko še kake druge podatke (oz. omejitve). Sedaj si A in B delita $K_{AB}$ in se lahko varno pogovarjata.
Slabosti tega sistema so, da zahteva dostop do TA ob začetku vsakega pogovora. Zato ni primeren za velika omrežja in za naprave brez priključka na internet. Zahteva tudi uskladitev ur, če se želimo držati omejtev veljavnosti.

### 1.3.2 Certifikati

Glej poglavje 2.

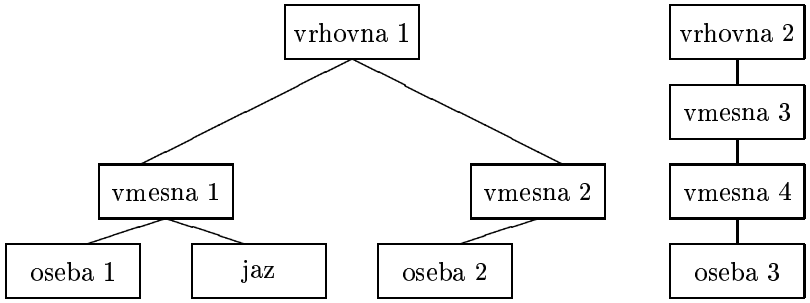
# 2 Certifikati in PKI

## 2.1 Splošno

Ta drugi pristop temelji na asimetričnih sistemih. Ti omogočajo podpisovanje sporočil. Certifikat je sporočilo, ki vsebuje atribute osebe, njen javni ključ, omejitve uporabe ter druge podatke in je podpisano z strani TA. Vsaka oseba v sistemu mora poznati javni ključ TA, zato lahko preveri podpis certifikata. S certifikatom TA garantira, da je dani javni ključ res last osebe opisane z atributi.
Certifikat dobi vsaka oseba, ko se priključi sistemu. Pred tem mora seveda dokazati resničnost svojih atributov in dodati še svoj javni ključ. Nato ji TA izda certifikat. TA, ki izdaja certifikate, pravimo CA (Certification Agency). Oseba nato ob začetku pogovora pošlje sogovorniku svoj certifikat, s čimer dokaže svojo identiteto. Če bi mu poslala nek tuj certifikat, pogovora seveda ne bi razumela.
Tak sistem potrebuje dostop do TA le ob priključitvi sitemu. Delo si lahko deli tudi več agencij (CA-jev), tako da obremenitve strežnikov niso velike in funkcije niso centralizirane.
Uporabnik lahko vsem tem agencijam vnaprej verjame ali pa tudi ne. Tistim, ki jim verjame pravimo vrhovne agencije (Root Agency). Taka agencija RA lahko tudi pooblasti kako drugo agencijo A za izdajanje certifikatov (taki pravimo vmesna agncija). Potem bomo verjeli tudi certifikatom, ki jih izdaja A.
Če si narišemo graf agencij in oseb dobimo drevesno strukturo. Na vrhu so vrhovne agencije, listi so osebe, vmes pa so vmesne agencije. Vsemu skupaj pravimo Infrastruktura Javnega Ključa (IJK).

## 2.2 Uporaba certifikatov

Možnosti uporabe so na vseh nivojih programske opreme in telekomunikacij. Navedel bom le nekaj področij, kjer se je uporaba certifikatov že prijela in jih lahko uporablja vsak uporabnik osebnega računalnika:




Slika 1: Infrastruktura Javnega Ključa (IJK)

- **elektronska pošta:** Pošto je možno podpisati ali zašifrirati. Najlažje to naredimo z Outlook Express-om.
- **varne domače strani:** Strani, ki se začenjajo z https:, uporabljajo tehnologijo varnih strani. Brkljalnik se s strežnikom pogovarja prek protokola SSL (Secure Socket Layer). Strežnik se na zahtevo klienta predstavi z svojim certifikatom. Klient ga preveri, nakar zgenerira simetrični ključ in ga pošlje strežniku zašifriranega z javnim ključem iz certifikata. Nato se pogovarjata prek varnega kanala (delita si simetrični ključ). Autorizacije klienta ta protokol ne predvideva.
- **bančne strani:** Tu se uporabnik autetificira z pomočjo gesla in svojega certifikata. Same strani seveda uporabljajo zgornji protokol.
- **digitalni denar:** Pravi digitlani denar še ni v rabi, bo pa prav gotovo. Tudi tu bodo certifikati odigrali svojo vlogo.
- **podpisovanje software-a:** Programi pobrani z interneta, bi lahko bili podtaknjeni, okuženi ali kako drugače potencilano škodljivi. Če so programi podpisani, vsaj vemo s kom imamo opravka in kdo bo odgovarjal za morebitno škodo.

## 2.3 Vsebina certifikat

Certifikati morajo biti zgrajeni na način, da jih razumejo vsi uporabniki oz. njihovi programi. Zato je bil sprejet stanadard X.509, ki detajlno opisuje zgradbo certifikata. Certifikat deli na obvezno vsebino, dodatke oz. razširitve in podpis celotne vsebine.
Pod obvezno vsebino spadajo:

- verzija (1,2,ali 3). Danes se uporablja verzija 3, ki ponuja največ možnosti za opis certifikata.
- serijska številka certifikata (določi jo CA)
- izdajtelj
- veljavnost
- oseba oz. lastnik certfikata
- javni ključ osebe in pripadajoči algoritem (običajno RSA)

Razširitve omogočajo prilagodljivost certifikatov, saj lahko z ravojem programske opreme, definiramo nove. Nekatere podpirajo vsi programi, ki uporabljajo certifikate, ostale pa le nekateri. Zato so nekatere razširitve označene kot kritične, kar pomeni, da jih program mora nujno podpirati, če želi uporabljati ta certifikat.

Med tipične razširitve spadajo:

- dovoljeni načini (nameni) uporabe ključa (za podpise, za enkripcijo, ...)
- alternativno ime osebe (lahko tudi npr. email)
- CRL distribucijska točka in mnogi drugi parametri glede CRL-ja
- ID ključa izdajatelja (pomaga pri iskanju izdajateljovega certifikata)
- ID ključa osebe
- osnovne omejitev (npr. končna oseba ali CA)

To še zdaleč niso vse razširitve. Obstaja namreč še kar nekaj različnih atributov, ki opisujejo uporabo in politiko (policy) dela z certifikati. Tudi večina obveznih vsebin ima lahko svoje dvojnike med razširitvami. Različni proizvajalci so si namreč želeli te podatke malce prilagoditi.
Naj omenim, da CRL (Certificate Revocation List) pomeni listo preklicanih ključev. Eden večjih problemov dela s certifikati je ravno, nadzor nad preklicanimi ključi. Zato se v razširitvi CRL, navede mesto (na internetu), kjer naj aplikacija preveri ali je bil ta certifikat morda preklican. Žal tega večina aplikacij ne preverja.

# 3 Tipi certifikatov in njihova uporaba

V IJK (Infrastrukturi Javnih Ključev) poznamo štiri tipe oseb-agencij in vsaka ima malce drugačen certifikat. Razlikujejo se tudi glede načina uporabe ter glede hranjenja in uporavljanja z njimi. Primere bom navajal iz oklolja Windows, vendar so splošne značilnosti skupne vsem operacijskim sistemom.
Windowsi (vsaj 98) hranijo certifikate v skupnem skladišču. Do njega pridete z Settings/Internet Options/Content/Certificates. Certifikati so razdeljeni v štiri skupine, ki jih bom v nadaljevanju opisal. Na vsak certifikat lahko kliknemo, ga preverimo in pogledamo. Vidimo vse osnovne in razširjene vsebine ter tudi verigo certifikatov (vse, ki so potrebni za zaupanje v ta certifikat). Certifikatu lahko, tudi omejimo namene uprabe (ni isto kot razširitev nameni uporabe), ali celo dodamo nove. Začetno stanje namenov uporabe se samo nastavi glede na vsebino certifikata.

## 3.1 Tvoj certifikat

Tak certifikat navaden certifikat osebe, ki ima v skladišču dodan tudi privatni ključ. Za uporabo le tega je včasih potrebno še dodatno geslo. Vedno pa je potrebna dodatna zaščita z geslom, če certifikat (z privatnim klučem vred) izvoziš. To pomeni, da ga spraviš v datoteko, za varnostno kopijo ali za prenos na drug računalnik.
Lasten certifikat je potreben, če želimo npr. podpisati ali zašifrirati elektronsko pošto (email). Še posebej enostavno je to z Outlook Express-om.

## 3.2 Certifikat osebe

Te certifikati naj bi se najpogosteje uporabljali. Dobimo jih, ko prejmemo podpisano ali zašifrirano elektronsko pošto, ko se povežemo na strežnik prek varne povezave (SSL oz. za brkljalnike https:), ali pa, ko iz interneta poberemo nov program. Takrat proizvajalec pogosto dokaže svojo avtentičnost s certifikatom in podpisom programa. V kolikor te certifikate že imamo, se ne nalagajo ponovno, če pa manjka kak certifikat iz certfikatne verige, se običajno priskrbi avtomatično.

## 3.3 Certifikat vmesne agencije

Vmesna agencija je agencija, ki jo same ljudje ne poznajo dovolj, da bi ji zaupali, zato njen certifikat podpiše neka vrhovna agencija.

### 3.4 Certifikat vrhovne agencije

Vrhovnim agencijam moramo vnaprej zaupati. Zato nas sistem, vedno ko dobimo kak nov tak certifikat, izredstno vpraša, ali smo mu res pripravljeni zaupati in to za vse namene in pod vsemi pogoji navedenimi v certifikatu (kasneje lahko to spremenimo). Ob namestivi Windows-ov je privzeto, da zaupamo kar lepemo številu vrhovnih (VeriSign,...)in tudi vmesnih agencij. Lahko jim sicer tudi nehamo zaupati, ampak običajno to ni smiselno.
Če pogledamo vsebino takega certifikata, je bistvena razlika ta, da sta izdajatelj in oseba enaka (tudi njuna ID-ja sta enaka). Različnost se sicer kaže tudi pri ostalih atributih. Namen uporabe je včasih širok, včasih pa omejen le na izdajo drugih certifikatov.

## 4 Abstraktna struktura certifikata

Vsebino, ki smo jo opisali v poglavju 2.3 želimo strogo in enolično definirati. Temu pravimo, da definiramo abstraktno podatkovno strukturo. Le tako bo stvar postala izmenljiva in na njej bomo lahko izvajali nadaljne operacije.
Da bi definicijo razumelo čimveč ljudi, so standardizirali jezik za opis abstraktnih podatkovnih struktur. Imenuje se ASN.1 (Abstrakt Syntax Notation). Naj na hitro povzamem sintakso jezika. Ločimo osnovne (podatkove) tipe:

- BOOLEAN (logična vrednost)
- INTEGER (celo število)
- BIT STRING (zaporedje bitov)
- OCTET STRING (zaporedje bytov)
- NULL (prazno (včasih pride prav)
- OBJECT IDENTIFIER (OID) (ID objekta)
- PrintableString
- IA5String
- UTCTime (mednarodni čas)

Poleg teg poznamo še dva sestavljena tip SEQUENCE (zaporedje) in SET (množica). Ta dva vsebujeta več osnovnih ali sestavljenih tipov (kot struktura v C-ju). Pri zaporedju je vrstni red pomemben, pri množici pa ne. Zgoraj opisani tipi sicer niso vsi, vendar za pregled zadostujejo.
Naj za primer navedem kar strukturo certifikata (določena s standardom X.509) v jeziku ASN.1 .

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate      TBSCertificate,
    signatureAlgorithm  AlgorithmIdentifier,
    signatureValue      BIT STRING  }
TBSCertificate  ::=  SEQUENCE  {
    version          [0]  EXPLICIT Version DEFAULT v1,
    serialNumber          CertificateSerialNumber,
    signature             AlgorithmIdentifier,
    issuer                Name,
    validity              Validity,
    subject               Name,
    subjectPublicKeyInfo  SubjectPublicKeyInfo,
    issuerUniqueID   [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- podprto le v verziji v2 in v3
```

```
    subjectUniqueID  [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- podprto le v verziji v2 in v3
    extensions       [3]  EXPLICIT UniqueIdentifier OPTIONAL,
                          -- podprto le v verziji v3
    }
Version  ::=  INTEGER   {  v1(0), v2(1), v3(2)  }
CertificateSerialNumber  ::=  INTEGER
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm               OBJECT IDENTIFIER,
    parameters              ANY DEFINED BY algorithm OPTIONAL  }
Name  ::=  CHOICE  {
    RDNSequence  }
RDNSequnce  ::= SEQUENCE OF RelativeDistinguishedName
RelativeDistinguishedName  ::=  SET OF AttributeTypeAndValue
AttributeTypeAndValue  ::=  SEQUENCE  {
    type         AttributeType,
    value        AttributeValue  }
AttributeType  ::=  OBJECT IDENTIFIER
AttributeValue  ::=  ANY DEFINED BY AttributeType
Validity  ::=  SEQUENCE  {
    notBefore         Time,
    notAfter          Time  }
Time  ::=  CHOICE  {
    utcTime           UTCTime,
    generalTime       GeneralizedTime  }
UniqueIndentifier  ::=  BIT STRING
SubjectPublicKeyInfo  ::=  SEQUENCE  {
    algorithm         AlgorithmIdentifier,
    subjectPublicKey  BIT STRING  }
Extensions  ::=  SEQUENCE SIZE (1.MAX) OF Extension
Extension  ::=  SEQUENCE  {
    extnID       OBJECT IDENTIFIER,
    critical     BOOLEAN DEFAULT FALSE,
    extnValue    OCTET STRING  }
```

Razširjeno vsebino predstavljajo Extensions, ostalo pa je osnovna (oz. obvezna) vsebina.
Pomudimo se še malo pri različnih tipih.
OBJECT IDENTIFIER (ali kratko OID) predstavlja zaporedje številk, ki enolično opisujejo določen objekt. Primer 1.2.840.113549.1.1.4 pomeni ( 1 - ISO, 2 - ISO član, 840 - US, 113549 - RSADSI, 1 - PKCS, 1 - PKCS-1, 4 - MD5 z RSA enkripcijo). Kot vidimo obstaja hiarhija izdajateljev in vsak lahko na svojem področju sam izdaja OID-je. Zato se lahko podobne ali skoraj enake stvari nahajajo na povsem različnih mestih. Iskanje po OID drevesu in ostale operacije nam olajša stran http://www.alvestrand.no/~hta/objectid/
Name je predstavljen z množico atributov (s tipom in vrednostjo). Različni atributi so: navadno (osebno) ime, deželno ime, lokalno ime, država, organizacija in oddelek v orgnizaciji, itd. Primer bi bil lahko: Miha Vuk iz Ljubljane v Sloveniji, študent na FMF.
Razširitev je sestavljena kot je zgoraj opisano. Vsak tip razširitve ima svoj OID. Vsebina je zapisana v OCTET STRING-u, ki pa je v resnici neka ASN.1 struktura odvisna od tipa razširitve. Do sedaj uporabljane razšitive so opisane v standardu X.509.

# 5 Od abstraktne strukture do zaporedja bytov

Rekli smo, da smo strukturo certifikata dobro definirali, zato da bomo lahko na njej izvajali nadaljne operacije. Na koncu želimo dobiti zaporedje byte-ov, ki ga lahko posnamemo v datoteko.

Le tako so podatki primerni za prenos med računalniki. Pretvorbi iz abstraktne podatkovne strukture v zaporedje byte-ov pravimo kodiranje.
Skupaj z jezikom ASN je bilo razvito tudi kodiranje zanj. Uporablja DER (Distinguish Encoding Rules) kodiranje, ki je nadomestilo BER(Basic Encoding Rules). To žal ni imelo željene lastnosti, da je kodiranje enolična preslikava, je pa bilo zato učinkoviteješe.
Poglejmo si nekaj osnovnih primerov kodiranja DER.

### 5.1 Kodiranje DER

DER kodiranje temlji na trojkah tip(tag)/dolžina/vrednost. Vsi trije elementi zavzemajo celo število oktetov (bytov). Tip in dolžina sta predstavljena z števili. Glede na potrebo lahko vsak zasede enega ali več oktetov. Tip (običajno zahteva le en oktet) vsebuje tudi zastavice ali je tip sestavljen ali osnovni in ali je odvisen od konteksta, aplikacije, okolja ali pa je splošen.
Kodiranje vsebine je močno odvisno od tipa. OCTET STRING se na primer kar direktno prenese v vsebino, za ostale tipe pa ne gre tako preprosto (za podrobneješi opis glej [**?**])
Če gre za zaporedje (SEQUENCE), vsebino predstavljajo kar zaporedno zloženi zakodirani elementi zaporedja.

## 6 Izdelava preprostega certifikata

Lotimo se sedaj izdelave kar se da preprostega certifikata. Omenjal bom le splošne prijem in vrste polj, detajle pa si oglejte v priloženi programski kodi.
Vsak certifikat, tudi najpreprostejši, mora imeti izpoljnena obvezna polja. To tudi zadostuje za osnovno uporabo certifikata. Vendar imamo tudi znotraj obveznih polj več možnosti in bi rad razložil najpreprostejše.

**Version** naj bo v3, **serialNumber** pa poljuben.
**signature** sicer ne vem čemu služi, predlagam pa naj bo enak kot signatureAlgorithm (npr. 1.2.840.113549.1.1.4=md5RSA) (pri vseh OID-jih si pomagajte z zgoraj omenjeno internetno stranjo [**?**]).
**issuer** in **subject** naj bosta poimenovana kar z navadnim imenom (CN=2.5.4.3) ali pa imenom organizacije(2.5.4.10).
Pri veljavnosti (**validity**) lahko uporabljamo UTCTime za datume do 2050, sicer pa moramo uporabiti GeneralizedTime. Oba seveda vsebujeta tudi čas dneva.

Kmalu bi želeli za resno uporabo certifikat malo nadgraditi z razširitvami.
Najprej mu dodajmo **basicConstraints** (osnovne omejitve). Z njim določimo ali gre za agencijo (CA) ali za končno osebo, ter tudi omejitve poti certificiranja (verige zaupanja).
Lahko bi dodali tudi **keyUsage** (omejitve uporabe ključa). Z njimi detajlno določimo, za kaj se kluč sme in za kaj se ne sme uporabljati. Če tega ni si lahko vsak uporabo razlaga po svoje (večinoma se privzame, da ni omejitev).
Obe zgornji razširitvi sta kritični.

Dodajmo še dve nekritični razširitvi, ki pa sta verjetno še bolj pomembni.
Gre za **subjectKeyIdentifier** in **authorityKeyIdentifier**. To sta 20 bytni poljubni zaporednji, nekakšna označba certifikata. Le z njuno pomočjo, je sistem sposoben poiskati pri sebi izdajateljev certifikat in z njegovo pomočjo preveriti našega.

Z vsemi temi dodatki je naš certifikat že povsem uporaben. Da ne pozabimo bi še enkrat omenil razlike pri izdelavi treh različnih vrst certifikatov.

- **certifikat osebe:** Le sledimo gornjim navodilom.
- **certifikat vmesne agnecije:** Pravilno nastavimo basicConstraints in keyUsage.

- **certifikat vrhovne agnecije:** Velja isto kot za vmesno agencijo, poleg tega pa sta izdajatelj (issuer) in oseba (subject) enaka. Enaka morata biti zato tudi subjectKeyIdentifier in authorityKeyIdentifier.
- **privatni certifikat:** Za njihov prenos se uporabljajo drugačni formati (npr. datoteke .pfx), ki vsebujejo tudi privatni ključ. Z njimi se nisem ukvarjal.

# 7 Programje za izdelavo certifikata

## 7.1 Uvod

Naj razložim sedaj moj pristop. Ker se nisem želel ukvarjati z implementacijo potrebnih algoritmov, sem iz interneta pobral paket RSAEuro. Od njega potrebujemo le knjižnico rsaeurolib.lib in program za generiranje ključev REDEMO.EXE. Za vsak slučaj kljub temu prilagam tudi celoten paket Rsaeuro.zip.
Kar se tiče moje kode, je za vas najpomembnejši MAKECERT.CPP, kjer se generira certifikat. Glavnina implementacije je sicer skrita v CERT.H, za dostop do knjižnice rsaeurolib.lib pa je potreben še CertREDEMO.h.
Za začetno preiskušanje imate program MAKECERT.EXE že preveden, po spreminjanju določenih parametrov, pa ga boste morali ponovno prevesti (po manjših spremembah MAKECERT.CPP). V pomoč je moj projekt MAKECERT.DSP (za Visal C++), vendar bo verjetno treba popraviti nekatere poti. Z Visal C++ -om se vse prevede brez težav, z ostalimi prevajalniki pa se lahko zatakne že pri prevajanju knjižnice.

## 7.2 Izdelava ključa

Kluč poljubne dolžine zgeneriramo z programom REDEMO.EXE in ga posnamemo v .key datoteko. Program ima splošen uporabniški vmesnik in je splošno uporaben (za RSA), vendar ga mi za druge stvari ne potrebujemo.

## 7.3 Vnos parametrov

Ni nam sicer treba, vendar običajno želimo spremeniti vsaj ime subjekta, agencijo, ... Ker gre le za prototip, je treba večino teh vsebin popravljati kar v programski kodi (MAKECERT.CPP). Izjeme so le izhodna datoteka (certifikat), ključ subjekta in agencije ter serijska številka. Ti se podajajo kot agumenti v programski vrstici.
V kodi torej nastavimo:

- ime izdajatelja: AtributeValue1
- ime subjekta (osebe): AtributeValue2
- obdobje veljavnsti: notBefore in notAfter ( (01,9,18,21,0,5) pomeni 18.9.2001 21:00:05).
- id izdajateljovega ključa: authkeyId (20 bytov v hex-u)

Id izdajateljovega ključa preberemo iz njegovega certifikata (Authority Key Identifier). Preko njega sistem tudi poišče pot certificiranja. V kolikor pa delamo certifikat vrhovne agencije, navedemo kar `char *authkeyId=keyId`, saj morata biti oba id-ja enaka. Id ključa subjekta se naključno zgenerira sam.
Lahko se tudi poigramo z basic constraints in key usage, vendar si prej poglejmo opis teh dveh razširitev v standardu [**?**].

### 7.4 Prevajanje in zaganjanje programa

Program prevedemo (za pomoč je priložena projektna datoteka) ali pa uporabimo že prevedeno različico. Zaženemo ga brez parametrov in nam izpiše pomoč.

```
F:\Kripto\Projekt\Certifikati\Cert>makecert
Premalo prametrov!
 makecert cert_file subject_key_file [authority_key_file serial_number]
```

Za primer pravilnih parametrov nam lahko služi test.bat.

### 7.5 Preverjanje oz. uporabnost narejenih certifikatov

Če smo vse uspešno opravili smo dobili certifikat (argument `cert_file`),ki naj ima končnico `.der`.
Če nanj dvakrat kliknemo si ga lahko ogledamo in preverimo, če je njegova vsebina res taka kot smo jo želeli. Če smo Authority Key Identifier pobrali iz nekega certifikata, ki mu zaupamo, potem bo naš certifikat veljaven in sistem bo sam našel pot certificiranja. V nasprotnem primeru, ga bo sistem označil kot neveljavnega, vendar bomo njegovo vsebino še vedno lahko brali. Če bomo naredili certifikat vrhovne agencije, nas bo sistem vprašal ali ji verjamemo, kar se bomo morali odločiti sami.

## 8 Zaključek

Kot vidimo so certfikati dovolj fleksibilni in še dovolj preprosti, da odpirajo kritografiji z javnimi ključi široke možnosti. Vključili naj bi se v vse naše komunikacije in nam približali novo stopnjo varnosti, zaupnosti in zaupanja. Kot primer naj navedem, da naj bi že v kratkem vsi državljani dobili pametne kartice s svojim privatnim ključem ter zraven seveda še certifikat. Z njim se bomo lahko autentificirali pri delu z državno upravo in tudi ostalimi osebami. Dokumente bomo lahko podpisovali, jih kriptirali in počeli še marsikaj drugega. Pametna kartica bo služila kot varna shramba za privatni kluč. Vse to lahko počnemo že sedaj, le da se uporaba še ni tako razmahila.

Ne smemo pa seveda pozabiti tudi na probleme certifikatov. Tudi če je privatni ključ še tako varno shranjen na pametni kartici ali čem podobnem, še vedno obstaja možnost kraje ali rabitja ključa. V tem primeru je seveda edina rešitev čimprejšnji preklic ključa in izsleditev tatu. Zato obstajajo (podobno kot pri kreditnih karticah) liste preklicanih ključev (certifikatov) (CRL liste). Običajno certifikat vsebuje podatek, katero listo naj uporabnik preveri pred uporabo certifikata. Vendar, če bi vsi preverjali CRL-je, bi bili CRL strežniki močno obremenjeni, poleg tega pa to zahteva skoraj stalno povezavo na internet. Ta pogoj za mnoge "off-line"naprave ni sprejemljiv, pa tudi sicer izniči mnoge prednosti certifikatov pred Kerberosom (glej 1.3.1). Zato se zaenkrat tega praviloma ne preverja, kako bo v prihodnosti pa bomo videli.
Drugi problem predstavlja filozofija zaupanja. Zaenkrat velja, da so vse agencije, ki jih certificira, neka vrhovna agencija, ki ji zaupamo, tudi vredne zaupanja. Zato bomo tudi njim zaupali, da certificirajo druge osebe in agencije. Ta sklep se seveda rekurzivno ponavlja, čeprav nikakor ni očiten. Bili so že razni predlogi izboljšanja tega upravljanja z zaupanjem (trust management), vendar se v praksi še niso prijeli.
Lahko bi na primer posameznim agencijam določili koliko jim zaupamo, kot maksimalno število vmesnih agencij med njo in končno osebo. Taka omejena tranzitivnost zaupanja bi bolje ustrezala realnosti, vendar bi novopečeni uporabnik imel težave z začetnimi nastavitvami.

Skratka na področju certifikatov ne manjka ne perspektiv, ne nerešenih problemov. Zato bomo o njih gotovo še dosti slišali (tudi v vsakdanjem življenju).