SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD

Simulacija osnovnih sigurnosnih protokola u programskom jeziku Obol

Anamarija Friganović

Voditelj: Doc. dr. sc. Marin Golub

Zagreb, listopad, 2008.

Sažetak

Cilj ovog diplomskog rada jest upoznavanje s osnovnim elementima Obola te njegovom primjenom, kao i na konkretnim primjerima pokazati mogućnosti Obola pri ostvarenju sigurnosnih protokola. U praktičnom dijelu ovog rada ostvareno je nekoliko sigurnosnih protokola u programskom jeziku Obol. Pokazalo se da je Obol zaista intuitivan i jednostavan jezik, koncipiran tako da se standardni zapisi sigurnosnih protokola s malo truda mogu prebaciti u oblik Obol skripte.

Abstract

The purpose of this diploma thesis is to get known to basic elements of Obol programming language, and to show the posibilities of Obol at implementation of security protocols by examples. In the practical part of this diploma thesis some security protocols were implemented in Obol. Obol is really intuitive and simple language, designed for easy transformation of security protocols in traditional notation to Obol scripts.

Sadržaj

1. UVOD

U okviru ovog rada ostvareno je nekoliko sigurnosnih protokola u programskom jeziku Obol. Obol je relativno nov programski jezik poznat maloj skupini ljudi. Obol je programski jezik visoke razine (po svojim svojstvima bi mogao pripadati četvrtoj generaciji programskih jezika) razvijen u svrhu pojednostavljivanja implementacije sigurnosnih protokola u računalne sustave. Tako sigurnosni protokoli napisani u Obolu za izradu zahtijevaju mnogo manje vremena i složenosti. Dobar primjer za tu tvrdnju možemo vidjeti usporedimo li samo veličinu skripte nastale ostvarenje mehanizma digitalnog pečata (jedan od primjera u šestom poglavlju) u Obolu s ostvarenjem digitalnog pečata na vježbama iz predmeta Operacijski sustavi 2 u programskim jezicima C# ili C.

Cilj ovog diplomskog rada jest upoznavanje s osnovnim elementima Obola te njegovom primjenom, kao i na konkretnim primjerima pokazati mogućnosti Obola pri ostvarenju sigurnosnih protokola. U drugom poglavlju je prikazan kratki uvod u računalnu sigurnost, opisani su najčešći sigurnosni mehanizmi te sigurnosni protokoli od kojih su neki ostvareni u praktičnom dijelu rada. U trećem poglavlju opisana je osnovna ideja i namjena Obola, temelji na kojima je nastao te gdje i kako se može primjenjivati. Također je dan pregled tipova podataka, vrsta varijabli i njima pripadajućih svojstava. Četvrto poglavlje daje pregled operatora i pravila za njihovo korištenje, dok se peto poglavlje bavi programskim okruženjem i njemu pripadajućim metanaredbama. U šestom poglavlju su prikazani jednostavni primjeri programa u Obolu i način pokretanja, dok se sedmo poglavlje bavi praktičnim radom. Za praktični dio diplomskog rada u programskom jeziku Obol ostvareni su slijedeći protokoli: Kerberos, protokol za raspodjelu ključeva u zatvorenom asimetričnom kriptosustavu, protokol za obostranu autentifikaciju u zatvorenom asimetričnom kriptosustavu, Otway-Rees protokol za autentifikaciju i dodjelu tajnog ključa, Wide Mouth Frog protokol za autentifikaciju i dodjelu tajnog ključa.

2. UKRATKO O RAČUNALNOJ SIGURNOSTI

2.1. Prijetnje i napadi

U distribuiranim računalnim sustavima informacije se prenose raznovrsnim otvorenim i nesigurnim komunikacijskim kanalima. Prijetnja je svaki događaj koji može poništiti ili smanjiti učinkovitost sustava, odnosno ograničiti ili onemogućiti ispunjenje cilja sustava ili procesa. Pojedine vrste napada (prijetnji) uzrokuju različite oblike narušavanja sigurnosti [5]:

Prisluškivanje ili presretanje – uljez (Eva) prisluškuje komunikacijski kanal između Ane i Branka i neovlašteno dolazi do povjerljive informacije.

Prekidanje – uljez djeluje tako da prekine komunikacijski kanal između Ane i Branka (onemogućeno je razmjenjivanje poruka).

Lažno predstavljanje – uljez koristi identitet druge osobe i neovlašteno pristupa podacima koji su namijenjeni samo prvoj osobi. Npr. Eva šalje poruke Branku predstavljajući se kao Ana.

Ponovno slanje snimljenih starih poruka – u kombinaciji s prisluškivanjem. Paketi odaslani preko komunikacijskog kanala koji se prisluškuje snimaju se i ponovo šalju kasnije. Napadač može odašiljanjem starih, ali ispravnih paketa dobiti pristup sustavu za koji nije ovlašten.

Modifikacija poruka – uljez mijenja pakete namijenjene nekom drugom, a ponekad ih i uništava.

Poricanje – poricanje da je neka poruka poslana, na primjer, kod plaćanja preko Interneta.

2.2. Sigurnosni zahtjevi

Sigurnosni zahtjevi su [1]:

Tajnost (povjerljivost) – skrivanje povjerljivih informacija ili sredstava. Podaci koji se šalju komunikacijskim kanalom se kriptiraju kako bi se otklonila prijetnja prisluškivanja.

Integritet (nepromjenjivost) – primalac mora biti siguran da je dobio poruku u nepromijenjenom obliku, to jest, da poruka u prijenosu nije mijenjana (napad modifikacijom poruke). Ovaj sigurnosni zahtjev ostvaruje se sigurnosnim mehanizmom sažetka poruke.

Raspoloživost – sva sredstva moraju uvijek biti raspoloživa i komunikacijski kanali neprekinuti. Ostvarenje ovog sigurnosnog zahtjeva otklanja prijetnju prekidanja.

Autentičnost (provjera identiteta) – koristi se na razini korisnika i na razini informacije. Dva korisnika koja počinju komunikaciju trebaju se predstaviti jedan drugome. Predstavljanje na razini informacije znači da za informaciju koja prolazi komunikacijskim kanalom treba provjeriti odakle dolazi, tko je vlasnik informacije, koliko je stara i kojeg je tipa[5]. Ostvarenje sigurnosnog zahtjeva autentičnosti otklanja sigurnosnu prijetnju lažnog predstavljanja.

Neporecivost – zaštita od poricanja, to jest, onemogućuje negiranje prethodno počinjenog djela od strane korisnika.

Autorizacija – obuhvaća provjeru autentičnosti i provjeru prava pristupa korisnika određenom

sredstvu.

Sigurnost poruka sastoji se od slijedećih sigurnosnih zahtjeva: tajnost, integritet, autentičnost i neporecivost.

2.3. Sigurnosni mehanizmi

Kako bi sigurna raspodijeljena aplikacija ili računalni sustav bili dobro strukturirani, moguće je uporabiti više sigurnosnih mehanizama. Radi boljeg snalaženja u istim, uobičajena je praksa sigurnosne tehnologije izražavati preko komunikacije tri lika kao sudionika, po imenima Ana, Branko i Eva. Ana (A) i Branko (B) strane su koje razmjenjuju tajnu informaciju, a Eva (E) je prisluškivač (*eavesdropper*).

2.3.1. Kriptografija

Kriptografija je ključni element implementacije sigurnosnih sustava. Koristi se za očuvanje tajnosti i integriteta informacije, kao i za podršku pri autentifikaciji pri komunikaciji između dvije strane te za digitalni potpis. Enkripcija je proces transformacije pomoću algoritma i pripadajućeg ključa koji služi za sakrivanje sadržaja poruke [1]. Kriptografski algoritam transformira čisti tekst u kriptirani skup podataka koji se ne može pretvoriti nazad u čisti tekst bez odgovarajućeg ključa.

Kriptosustavi mogu biti simetrični i asimetrični.

Kod **simetričnih** kriptosustava, koji se zasnivaju na uporabi logičke operacije XOR (specifična po tome što nakon dvostruke uporabe nad nekim podatkom dobijemo taj isti original), koristi se isti unaprijed dogovoreni ključ za kriptiranje i dekriptiranje – primjeri su DES, IDEA, 3-DES i drugi[1].

Kod **asimetričnih** kriptosustava ne koristi se isti ključ, nego je ključ K_E kojim se kriptiraju podaci različit od ključa K_D kojim se dekriptiraju podaci. Štoviše, na osnovi poznatog ključa K_E nije moguće (ili bar ne u razumnoj količini vremena) doći do vrijednosti ključa K_D. Asimetrični algoritmi se još zovu i *algoritmi s javnim ključem (engl. public-key algorythms)*, zbog njihovog svojstva da je ključ K_E kojim se kriptiraju podaci slobodno može otkriti. Ključ za kriptiranje K_E se naziva javnim ključem *(engl. public key)*, dok se ključ za dekriptiranje K_D naziva privatnim ključem *(engl. private key)*. Najkorišteniji asimetrični algoritmi se baziraju na problemu faktoriziranja velikih brojeva koji su umnožak dvaju prim-broja. U tako opisanom kriptosustavu komunikacija između Ane i Branka odvija se na sljedeći način:

- Branko i Ana objavljuju svoje javne ključeve K_{EA} i K_{EB}, a čuvaju tajne K_{DA} i K_{DB}.
- Kad Ana šalje Branku poruku, kriptira je njegovim javnim ključem K_{EB}.
- Branko sad može tu poruku dekriptirati svojim tajnim ključem K_{DB}.

Kako su simetrični kriptosustavi brži, učinkovitiji, ali i nesigurniji od asimetričnih zbog problema prenošenja zajedničkog ključa, često se ta dva kriptosustava koriste u kombinaciji. Tako se ključ prenosi pomoću asimetričnog, a poruka pomoću simetričnog kriptosustava (nešto se slično primjenjuje kod digitalne omotnice) [1].

2.3.2. Digitalna omotnica

Digitalna omotnica praktičan je način prenošenja kriptiranih podataka koji kombinira prednosti simetričnih i asimetričnih kriptosustava. Kako su asimetrični kriptosustavi presloženi i prespori za učinkovito kriptiranje većih količina podataka, podaci se kriptiraju simetričnim ključem te se taj simetrični ključ kriptira asimetričnim javnim ključem primatelja i dodaje kriptiranoj poruci. Time je riješen i problem prenošenja simetričnog tajnog ključa. Digitalna omotnica osigurava tajnost, ali ne i integritet poruke.

Digitalna omotnica = {E(poruka, K); E(K, K_{EB})}

Gdje je K tajni ključ simetričnog algoritma, a K_{EB} javni ključ primatelja.

2.3.3. Digitalni potpis

Pošiljatelj iz poruke, koristeći jednu od funkcija za izračunavanje sažetka poruke (*hash*), izračunava sažetak te taj sažetak potom kriptira svojim privatnim ključem i dodaje izvornoj poruci. Digitalni potpis je sigurnosni mehanizam koji ne osigurava tajnost poruke, ali koristi se za ostvarenje autentičnosti, integriteta i neporecivosti [1].

Digitalni potpis = {poruka; E(Qp, K_{DA})}

Gdje je Qp sažetak poruke a K_{DA} tajni ključ pošiljatelja

2.3.4. Digitalni pečat

Digitalni pečat digitalno je potpisana digitalna omotnica. Zadovoljava četiri sigurnosna zahtjeva – tajnost, autentičnost, integritet i neporecivost [1].

Digitalni pečat = {digitalna omotnica; E(Qp, K_{DA})}

2.4. Sigurnosni protokoli

2.4.1. Diffie-Hellmanov postupak za razmjenu ključeva

Ana i Branko unaprijed se slože o dva vrlo velika broja n i g, takva da je NZD(g,n) = 1, te ih mogu javno objaviti. Najbolje je za n odabrati prosti broj p. Ana odabire veliki nasumični prirodni broj x i šalje Branku: $X = g^x \bmod p$. Branko također odabire veliki nasumični broj y i šalje Ani poruku: $Y = g^y \bmod p$.

Sada oboje mogu izračunati ključ K:

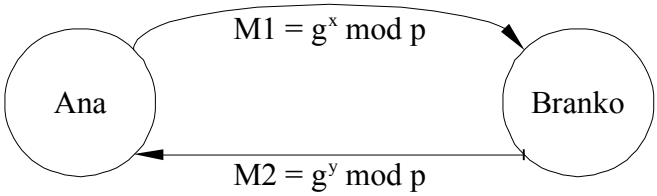
(Ana) $K = Y^x \bmod p = (g^y)^x \bmod p = g^{yx} \bmod p$

(Branko) $K = X^y \bmod p = (g^x)^y \bmod p = g^{yx} \bmod p$ [1]

K može poslužiti kao simetrični ključ za kriptiranje sljedećih poruka koje će razmijeniti Ana i Branko.

Diffie-Hellmanov postupak za razmjenu ključeva je osjetljiv na napad čovjeka u sredini. Eva ima mogućnost presretanja i zadržavanja poruka. Kako Ana i Branko računaju svoje brojeve X i Y, tako i Eva može izračunati svoj broj $Z = g^z \bmod p$. Eva zatim presreće komunikaciju

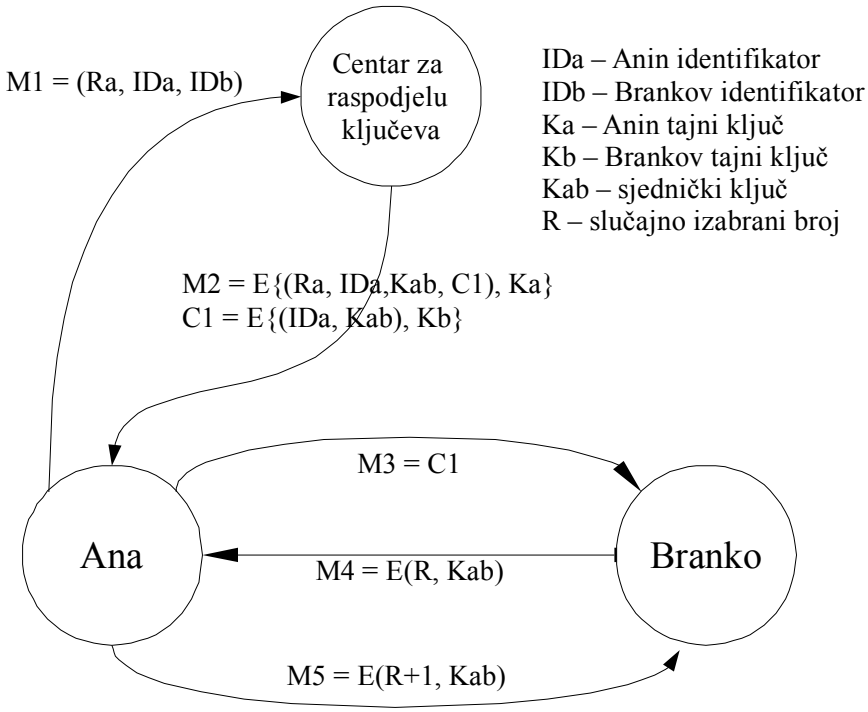
između Ane i Branka i spremi brojeve X i Y, a njima oboma pošalje broj Z. Ana i Branko, ne znajući da je Eva presrela poruke, normalno izračunavaju tajni ključ i šalju ga natrag. Međutim, sad je taj tajni ključ, umjesto iz brojeva X i Y, izračunat s brojem Z (kod Ane X i Z, a kod Branka Y i Z). Eva s tim ključevima može presretati i mijenjati poruke koje su Ana i Branko namijenili jedan drugome.



Slika 2.1 Prikaz Diffie-Hellmanovog postupka za razmjenu ključeva

2.4.2. Raspodjela ključeva prema Needhamu i Schroederu

Raspodjela ključeva po Needhamu i Schroederu osnovni je protokol za raspodjelu ključeva (temelj za ostale protokole pa i Kerberos) u sustavima s više korisnika. Kad bi svaka dva korisnika za međusobnu komunikaciju imala zajednički tajni ključ za sustav od N korisnika bilo bi potrebno imati (i osigurati čuvanje) N*(N-1)/2 ključeva –> svaki par korisnika ima svoj posebni ključ. Kako je to jako teško i složeno za izvođenje, ponajprije zbog očuvanja tajnosti ključeva, javlja se rješenje u obliku centra za raspodjelu ključeva. Tu ulogu preuzima pouzdani poslužitelj u kojeg svi imaju povjerenje.



Slika 2.2Prikaz poruka u protokolu Needham-Schroeder

Needham-Schroeder protokol funkcionira tako da se svi potencijalni sudionici moraju unaprijed prijaviti, te se uvode i vremenske oznake kako bi se spriječilo ponavljanje starih zahtjeva. Poslužitelj na zahtjev korisnika A nasumično generira tajni ključ za komunikaciju

među korisnikom A i korisnikom B, koji onda kriptira tajnim ključevima korisnika A i B te pošalje korisniku A. Korisnik A dekriptira svoj dio poruke, a ostatak proslijedi korisniku B. Nakon što B dekriptira svoj dio poruke, A i B imaju zajednički ključ i sve preduvjete za sigurnu međusobnu komunikaciju.

2.4.3. Kerberos

Kerberos je nazvan prema Cerberusu, čudovišnom troglavom psu iz grčke mitologije koji je čuvao ulaz u Had. Kerberos je mrežni autentifikacijski protokol koji omogućuje, korištenjem enkripcije pomoću tajnog ključa, sigurnu obostranu autentifikaciju preko nesigurne mreže. Sigurna autentifikacija se ostvaruje bez oslonca na postavke operativnog sustava, bez zahtjeva za fizičkom sigurnošću svih dijelova sustava te pod pretpostavkom da paketi koji putuju mrežom mogu biti pročitani i promijenjeni. Kerberos kao provjerena treća strana osigurava autentifikaciju pod navedenim uvjetima koristeći uobičajenu kriptografiju tajnog ključa.

Kerberos je prvotno bio napravljen kao autentifikacijski protokol namijenjen osiguravanju projekta Atena na MIT-u (*Massachusetts Institute of Technology*). Protokol zasad postoji u pet verzija. Verzije protokola od 1 do 3 pojavile su se samo interno na MIT-u. Verzija 4 je također bila namijenjena projektu Atena, no krajem 80-tih njeni autori, Steve Miller i Clifford Neuman, objavili su je za širu uporabu. 1993. u obliku RFC dokumenta (*Request For Comment*), broj 1510, pojavila se verzija 5 Kerberos protokola. Specifikacija je proširena 2005. godine u dokumentu RFC 4120 (autori John Kohl and Clifford Neuman), između ostaloga i da prihvati zahvate koji su se pojavili kod Microsoftove implementacije protokola u operacijskim sustavima Windows 2000, Windows XP i Windows Server 2003. [8].

Za autentifikaciju korisnika nekom aplikacijskom poslužitelju koristi se ulaznica koju je korisniku izdao Kerberos poslužitelj. Ta ulaznica je kriptirana s tajnim ključem poslužitelja, tako da ne postoji mogućnost da je korisnik izmijeni.

Najveća prednost Kerberosa, u odnosu na druge protokole za autentifikaciju, jest to što dopušta korisniku da se koristi različitim poslužiteljima na mreži, bez potrebe da više puta zahtjeva pristup i unosi zaporke.

Korisnik ili usluga koji se mogu autentificirati nazivaju se principalima (*principal*). Svaki principal ima jedinstveno ime. Autentifikacija principala je proces dokazivanja svog identiteta komponenti sustava koja zahtjeva taj sigurnosni korak – verifikatoru (*verifier*) [9]. Na taj način se dodjeljuju prava pristupa informacijama i uslugama, u ovisnosti tko je taj principal.

Principali u Kerberos modelu su korisnik, klijent, autentifikacijski poslužitelj (AS – *Authentification Service*), poslužitelj za dodjelu ulaznica (TGS – *Ticket Granting Service*) i poslužitelj koji pruža zatraženu uslugu. Klijent radi u ime korisnika.

Identifikator principala sastoji se od tri komponente:

- principalsko ime (*principal name*)
- ime instance (*instance name*)
- ime područja (*realm name*).

Elementi Kerberosa

Ulaznica – svaki principal koji zatraži pristup određenoj usluzi, ukoliko se autentificira kod autentifikacijskog poslužitelja, dobije ulaznicu za pristup toj usluzi. Ulaznica sadrži podatke bez kojih principal i poslužitelj na kojem se nalazi usluga ne mogu komunicirati (dokazuje identitet principala i kriptirana je privatnim ključem tražene usluge). Ulaznica između ostalog sadrži sjednički ključ koji će biti korišten za autentifikaciju principala verifikatoru, ime principala kojem se sjednički ključ izdaje, te razdoblje valjanosti sjedničkog ključa [9]. Ulaznica vrijedi samo dok razdoblje valjanosti ne istekne.

Autentifikator – klijent ga šalje uz ulaznicu kako bi spriječio da netko drugi ukrade ulaznicu i ponovo je upotrijebi. Sastoji se od više dijelova, a najvažniji su:

- principalski identifikator klijenta
- mrežna adresa
- oznaka trenutnog vremena

Autentifikator se kriptira sjedničkim ključem dobivenim od poslužitelja pa ga poslužitelj usluge ne može otkriti ukoliko nije svojim privatnim ključem otkrio sadržaj ulaznice (gdje se nalazi taj sjednički ključ). Poslužitelj usluge prihvaća ulaznicu ukoliko se poklapaju identifikator i oznaka vremena iz ulaznice i autentifikatora te mrežna adresa u autentifikatoru s mrežnom adresom s koje je paket poslan.

Kerberos klijent – principal koji traži pristup nekoj usluzi i dobiva ulaznicu za nju.

Kerberos poslužitelj – sastoji se od autentifikacijskog poslužitelja i poslužitelja za dodjelu ulaznica. Tu se pohranjuju lozinke i tajni ključevi povezani sa svakim principalom, stoga je vrlo važno da bude što zaštićeniji. Kako bi se povećala pouzdanost Kerberos sustava, često se dodaju i pomoćni poslužitelji u sustav. Ukoliko se računalo na kojem je Kerberos poslužitelj sruši, neće se moći koristiti nijedna usluga ako nije konfiguriran pomoćni poslužitelj [9].

Poslužitelj usluge – bilo koja aplikacija (računalo) od koje klijenti traže usluge koristeći Kerberos ulaznice za autentifikaciju [9].

Sat – zbog vremenskih oznaka izrazito je važno da sistemski satovi unutar računala jednog Kerberos sustava budu usklađeni. Ako satovi nisu usklađeni, postoji opasnost da će poslužitelj odbiti ulaznicu smatrajuću da napadač pokušava uporabiti ulaznicu kojoj je istekao rok valjanosti (a zapravo nije tako). U svakom Kerberos sustavu može se postaviti maksimalna dozvoljena vremenska razlika između računala. Za sinkronizaciju se mogu iskoristiti i posebni vremenski poslužitelji (*timeservers*) ili NTP (*Network Time Protocol*) [9].

ASN.1 notacija

ASN.1 (*Abstract Syntax Notation One*) je, za kompjuterske mreže i telekomunikacije, standardna i vrlo prilagodljiva notacija. Glavna namjena joj je opisivanje struktura podataka za prikaz, kodiranje, prijenos i dekodiranje podataka. Sastoji se od skupa formalnih pravila za opisivanje struktura podataka, neovisno o sustavu na kojem se izvode. ASN.1 je precizna, formalna notacija koja otklanja bilo kakve nedoumice u prikazu podataka. ASN.1 sadrži razna pravila kodiranja, a ovo su standardna:

- BER (Basic Encoding Rules)

- CER (Canonical Encoding Rules)
- DER (Distinguished Encoding Rules)
- XER (XML Encoding Rules)
- PER (Packed Encoding Rules)
- GSER (Generic String Encoding Rules)

ASN.1 notacija skupa s pravilima kodiranja znatno olakšava razmjenu strukturiranih podataka, posebice između aplikacija razdvojenih mrežom, opisujući strukturu podataka neovisno o arhitekturi računala i programskom jeziku.

```
Ticket        ::= [APPLICATION 1] SEQUENCE {
        tkt-vno         [0] INTEGER (5),
        realm           [1] Realm,
        sname           [2] PrincipalName,
        enc-part        [3] EncryptedData -- EncTicketPart
}

-- Encrypted part of ticket
EncTicketPart   ::= [APPLICATION 3] SEQUENCE {
        flags                [0] TicketFlags,
        key                  [1] EncryptionKey,
        crealm               [2] Realm,
        cname                [3] PrincipalName,
        transited            [4] TransitedEncoding,
        authtime             [5] KerberosTime,
        starttime            [6] KerberosTime OPTIONAL,
        endtime              [7] KerberosTime,
        renew-till           [8] KerberosTime OPTIONAL,
        caddr                [9] HostAddresses OPTIONAL,
        authorization-data   [10] AuthorizationData OPTIONAL
}

-- encoded Transited field
TransitedEncoding       ::= SEQUENCE {
        tr-type         [0] Int32 -- must be registered --,
        contents        [1] OCTET STRING
}
```

Slika 2.3 Primjer strukture Kerberos ulaznice u ASN.1 notaciji

Razlike između verzije 4 i verzije 5

U verziji 5 korištenje kriptografije je razdvojeno u posebne module koje programer može promijeniti ili odstraniti iz implementacije, ukoliko je potrebno. Kad je poruka u protokolu kriptirana, uz kriptirani tekst dodaje se identifikator kriptografskog algoritma kako bi je primatelj znao dekriptirati. Tako su označeni i kriptografski ključevi – ako se pojave u poruci uz njih je naznačena duljina i vrsta kriptografskog algoritma za koji se koriste.

Kad se u porukama protokola pojavljuju mrežne adrese, također imaju dodatno označen tip i duljinu adrese kao i ključevi. Ako računalo korisnika ili poslužitelja koristi više mrežnih protokola ili ima više mrežnih adresa, onda su svi protokoli i sve adrese navedene u ulaznici.

Poruke koje se razmjenjuju u verziji 5 su opisane ASN.1 notacijom te kodirane po pravilima standarda ISO8825. Na ovaj način se izbjegavaju problemi koji bi nastali prilikom slanja velikog broja bajtova u verziji 4.

Ulaznica je u verziji 5 proširena i podijeljena na dva dijela: prvi dio ulaznice je kriptiran a drugi ne. Ime poslužitelja u ulaznici nije kriptirano kako bi poslužitelj s više imena mogao

izabrati pravi ključ kojim će kriptirati ostatak ulaznice. Sve ostalo u ulaznici je kriptirano. Valjanost ulaznice je opisana vremenom početka i vremenom kraja, za razliku od Verzije 4 gdje je bila opisana vremenom početka i duljinom trajanja.

U odnosu na Verziju 4, u Verziji 5 imena principala se zapisuju u drugačijem formatu. U verziji 5 format zapisa imena principala je:
komponenta/komponenta/komponenta@realm
Ime se sastoji od dva dijela: imena područja i ostatka, koji su odijeljeni znakom @.

2.4.4. Protokol Yahalom

Protokol Yahalom između dvije strane (Ana i Branko) pomoću autentifikacijskog poslužitelja (AS) raspodjeljuje sjednički ključ za međusobnu komunikaciju (simetrični kriptosustav). Autentifikacijski poslužitelj sa svakim sudionikom ima zajednički tajni ključ [9]. Kad se protokol izvrši, Ana i Branko mogu biti sigurni da je drugi sudionik nedavno bio u sustavu. Ana počinje komunikaciju tako da Branku šalje identifikator i nasumično generirani broj.

A –> B: M1 = A, Na

Branko zatim primljenoj poruci dodaje svoj nasumično generirani broj Nb, sve to kriptira svojim tajnim ključem (koji je također pohranjen na autentifikacijskom poslužitelju) i, skupa sa svojim identifikatorom, pošalje autentifikacijskom poslužitelju.

B –> S: M2 = B, E{(A, Na, Nb), Kb}

Autentifikacijski poslužitelj nasumično generira tajni sjednički ključ Kab, i pošalje Ani poruku koja se sastoji od dvije komponente:

a) Sjednički ključ Kab, Brankov identifikatori i oba nasumična broja, sve to kriptirano Aninim tajnim ključem.

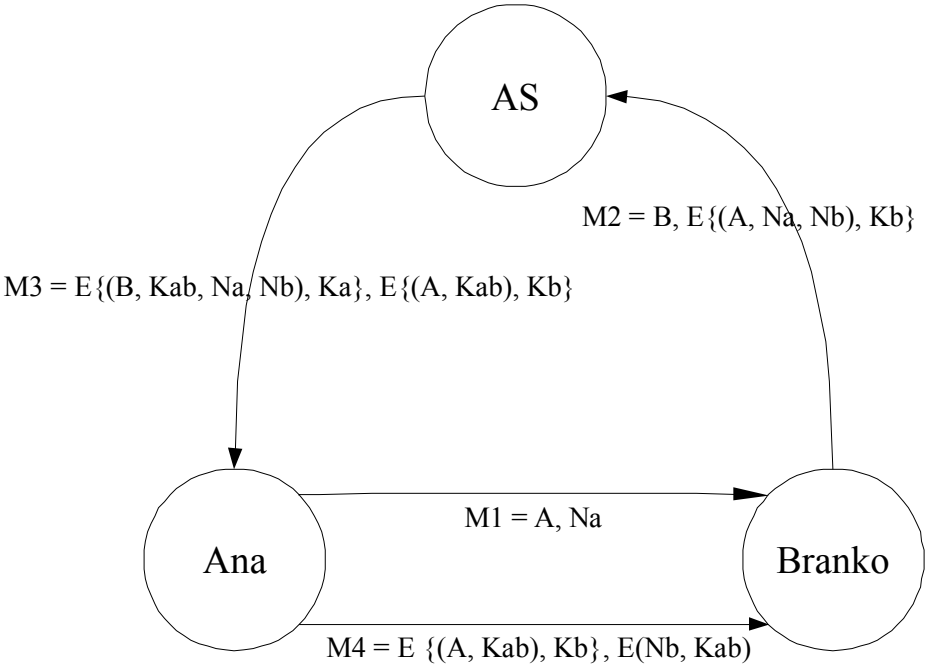
b) Svojevrsna "ulaznica" (kao kod Kerberosa) koju će Ana samo proslijediti Branku. U njoj se nalaze sjednički ključ i Anin identifikator kriptirani Brankovim tajnim ključem.

S –> A: M3 = E{(B, Kab, Na, Nb),Ka}, E{(A, Kab), Kb}

Ana zatim Branku prosljeđuje "ulaznicu" i *nonce* Nb koji je Branko poslao autentifikacijskom poslužitelju kriptiran sjedničkim ključem.

A –> B: M4 = E{(A, Kab), Kb}, E(Nb, Kab)

Nakon završetka protokola izvršena je autentifikacija i pouzdano je da su Ana i Branko nedavno bili aktivni u sustavu.



Slika 2.4 Prikaz protokola Yahalom

U prvotnoj verziji Yahalom protokola druga poruka imala je slijedeći oblik:

B –> S: B, Nb, E{(A, Na), Kb}

Dakle, Nb nije bio kriptiran. Međutim, ta je verzija mogla lako biti napadnuta "čovjekom u sredini". Pretpostavimo da je Eva (uljez) uspjela otkriti sjednički ključ K iz stare poruke (autentifikatora) koju je prisluškujući pohranila – E{(A,K), Kb}. Sada može prekinuti komunikacijske kanale između Branka i Ane te Branka i poslužitelja i "odglumiti" Anu i poslužitelj:

E –> B: M1 = A, Na

B –> E: M2 = B, Nb, E{(A, Na), Kb}

E –> B: M4 = E{(A, K), Kb}, E(Nb, K)

Kod prve poruke Eva glumi Anu pa Branku šalje Anin identifikator a sama generira *nonce* Na. U drugoj poruci, pak, Eva glumi autentifikacijski poslužitelj i prima od Branka poruku u kojoj se nalazi *nonce* Nb u čistom obliku (nije kriptiran). Sad Eva ima sve elemente da Branku pošalje četvrtu poruku umjesto Ane – stari autentifikator iz kojeg Branko otkriva sjednički ključ K i njime dekriptira *nonce* Nb. Branko je uspostavio komunikaciju s Evom misleći da je to Ana. Druga poruka je stoga modificirana i *nonce* Nb kriptiran pa Eva nikako nema elemente za generiranje četvrte poruke te se ne može predstaviti kao Ana.

2.4.5. Protokol Wide Mouth Frog

Wide Mouth Frog je poznati protokol za razmjenu dijeljenog simetričnog ključa između dva korisnika preko zajedničkog centra za raspodjelu ključeva. Centar za raspodjelu ključeva sa svim korisnicima ima zajednički tajni ključ. Protokol se odvija u samo dvije poruke.

Najveći problem ovog protokola je što svi korisnici moraju imati pristup nekom jedinstvenom satu, koji mora biti zaštićen od utjecaja treće strane, kako bi vremenske oznake u porukama imale smisla[7].

Također je potrebno dobro zaštititi autentifikacijsko središte jer ono čuva sve ključeve. Ako autentifikacijsko središte bude ugroženo bit će ugrožena i sigurna komunikacija.

Ana započinje izvođenje protokola slanjem svog identifikatora autentifikacijskom središtu. Uz identifikator šalje i oznaku vremena, identifikator korisnika s kojim želi komunicirati i nasumično generirani sjednički ključ, sve to kriptirano svojim tajnim ključem.

A –> AS: M1 = (IDa, C1)

C1 = E {(Ta, Kab, IDb), Ka}

Autentifikacijsko središte pomoću identifikatora pronađe Anin tajni ključ, dekriptira poruku i provjeri da li je vremenska oznaka istekla. Zatim Branku pošalje poruku, kriptiranu njegovim tajnim ključem, u kojoj se nalazi vremenska oznaka poslužitelja, Anin identifikator i sjednički ključ.

AS –> B: M2 = E {(Ts, Kab, IDa), Kb}

Ana i Branko sada mogu sigurno komunicirati pomoću sjedničkog ključa Kab.

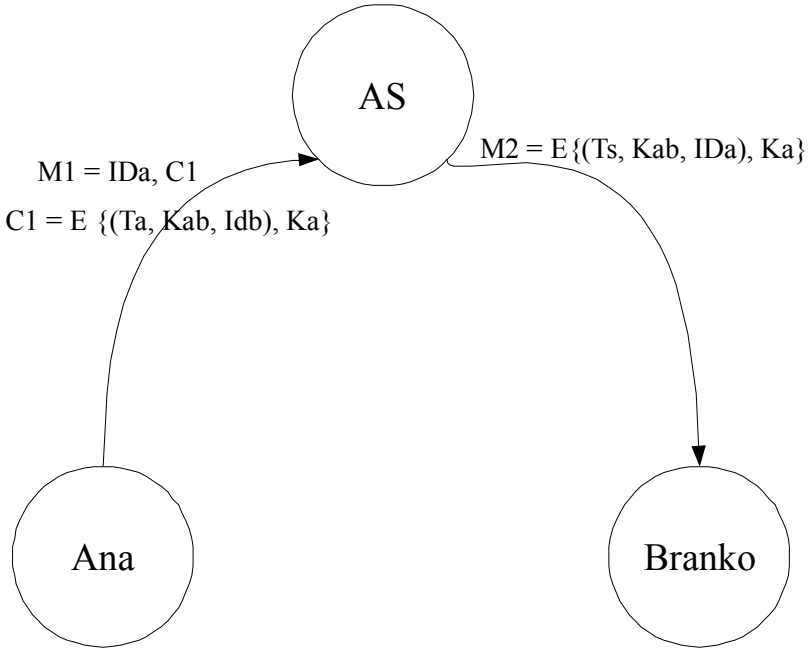
Protokol Wide Mouth Frog je poseban po tome što korisnik, a ne centar za raspodjelu ključeva, generira i određuje sjednički ključ.

Godine 1997. Gavin Lowe objavio je modifikaciju protokola Wide Mouth Frog [11]. Dodao je dvije poruke koje omogućavaju i autentifikaciju. Nakon što je primio zajednički sjednički ključ, Branko nasumično generira broj koji zatim kriptira tim ključem i pošalje Ani:

B –> A: M3 = E (Nb, Kab)

Ana primi poruku, dekriptira je, uveća broj za jedan i taj broj, kriptiran ključem Kab, pošalje Branku:

A –> B: M4 = E (Nb+1, Kab)



Slika 2.5 Protokol Wide Mouth Frog

3. OBOL OPĆENITO

Programski jezik Obol, razvijen na Sveučilištu u Tromsu, namijenjen je isključivo implementaciji sigurnosnih protokola. Ideja cijelog projekta je omogućavanje fokusiranja na opis i analizu samih sigurnosnih protokola tako da ostatak nepotrebnih rutinskih poslova (kao što su komunikacija, primitivne kriptografske transformacije i prezentacija poruka) obavlja programsko okruženje, čime izbjegavamo tipične smetnje koje nastaju pogreškama u tim dijelovima programske implementacije sigurnosnih protokola.

Kako se sigurnosni mehanizmi i protokoli koje opisuje Obol zasnivaju na pretpostavkama i vjerovanjima da su te pretpostavke točne, za logiku na kojoj će se programski jezik temeljiti odabrana je logika BAN, logika za analiziranje autentifikacijskih protokola.

Obol je vrlo intuitivan jezik i u većini je slučajeva protokol zapisan u standardnoj notaciji vrlo lako pretvoriti u notaciju pogodnu za implementaciju u njegovom programskom okruženju.

3.1. Programabilni sigurnosni protokoli

Vrlo je izazovan problem postaviti sigurnosnu politiku u dinamičnim i vrlo promjenjivim okruženjima. Sigurnost je običajno tretirana kao dodatna značajka, te se rijetko podrobno razmatra pri izradi i postavljanju novog sustava. Takav će pristup neizbježno povećati složenost i smanjiti učinkovitost sigurnosnog sustava. Drugi je problem u tome što uobičajeni alati za izgradnju sigurnosnih sustava daju očekivane rezultate samo u točno određenim okruženjima, gdje su sve situacije jasno definirane i predviđene u izgradnji sigurnosnog sustava. Budući da stvarni svijet nije statičan, promjene će se u sustavu sigurno događati, i postupno će početno sigurnosno rješenje biti u raskoraku s dinamičnim okruženjem i pripadajućim aplikacijama. Uza sve to, sigurnosni zahtjevi u kriptografskim funkcijama, kao i u složenim sigurnosnim protokolima, vrlo su složeni i teško ih je ostvariti, pogotovo u dinamičnom okruženju. Kao rješenje tih problema razvija se programabilnost – način ostvarenja sigurnosnih protokola koji nam omogućava prilagođavanje stalnim promjenama uvjeta i okoline. Najveći dio sigurnosti odvija se oko komunikacije – pojmovi kao što su tajnost, integritet, autentičnost i neporecivost.

3.2. Logika BAN

BAN (Burrows, Abadi, Needham) je logika za analiziranje autentifikacijskih protokola. Pretpostavlja se da je autentifikacija funkcija integriteta i svježine podataka te koristi logička pravila da bi definirala ta svojstva kroz sigurnosne protokole.

Analiza počinje tako da početne pretpostavke u protokolu načinimo eksplicitnima ("vjerujemo" da su pretpostavke točne, što se u Obolu ostvaruje operatorom `believe`). Nove pretpostavke dodaju se starima kao posljedica primanja poruka.

Kako bi bilo moguće modelirati složene sustave, svi modeli, uključujući i logičke, zasnivaju se na pretpostavkama. Ako su pretpostavke prejake, na način da previše pojednostavljuju sustav, model više nije iskoristiv. Drugi se problem pojavljuje kad se model premjesti na nove pretpostavke gdje originalne više ne vrijede. BAN logika, na primjer, nema mehanizme

za provjeru jesu li kriptografske funkcije pouzdano izvedene pa jednostavno pretpostavi da jesu, što u stvarnom životu i nije uvijek slučaj.

3.3. Primjena

Obol je dizajniran kako bi se pojačala fleksibilnost sigurnosnih komunikacijskih rješenja i ostvarivanje dinamičnih sigurnosnih politika. Za razliku od ostalih jezika za pisanje sigurnosnih politika, sigurnosne politike pisane u Obolu mogu se dinamički pozivati i izvršavati, te su jasno odvojene od ostalih dijelova sustava. Obol se koristi za sigurnosne protokole ili bilo koji protokol čija struktura uključuje kompoziciju ili transformaciju. Ti protokoli vrlo lako mogu biti nadograđivani zbog modularne strukture koja omogućava dodavanje novih formata poruka, kriptografskih algoritama i komunikacijskih tehnologija.

Obol se sastoji od dvije komponente: programskog jezika i programskog okruženja. U pojednostavljenoj podjeli uloga komponenta programskog jezika zadužena je za ostvarivanje sigurnosnih politika i složenih protokola, dok osnovne zadatke, kao što su temeljni kriptografski algoritmi, operacije slanja i primanja poruke te ostale primitivne operacije, obavlja programsko okruženje. U slučaju potrebe komunikacije programera s programskim okruženjem upotrebljava se poseban skup naredbi – metanaredbe.

Obol omogućava korisnicima opisivanje sigurnosnih zahtjeva aplikacije u formi programa[3]. To je vrlo značajno jer odjeljivanje implementacija omogućava sigurnosnim politikama prilagođavanje raznim okruženjima. Tako, na primjer, ako dođe do promjene protokola koji se inače upotrebljavaju za neku uslugu, korisnik bi morao proizvesti ili instalirati novi program koji je kompatibilan s novim protokolima, što je teško, pogotovo u heterogenim okruženjima koja su i inače teška za implementaciju sigurnosti. No, korisnici Obola nakon promjene protokola mogu samo pokrenuti druge Obol skripte koje su kompatibilne s tim novim protokolima koji se koriste, što je puno lakše nego da postavljaju cjelokupni novi softver.

Ključna karakteristika Obola postignuta je odjeljivanjem implementacije sigurnosnih politika i složenih protokola od samih jednostavnih i osnovnih zadataka, kao što su temeljni kriptografski algoritmi, operacija slanja i primanja poruka, i sl. koje obavlja programsko okruženje [3].

3.4. Varijable, svojstva i tipovi podataka

Varijable u Obolu zovu se *simboli,* te sadržavaju vrijednost i razna svojstva koja pobliže opisuju simbol i njegovu vrijednost. U Obolu se uz vrijednost, a ne uz simbole, veže tip podataka.

Postoje dvije vrste simbola: simbol i anonimna varijabla.

Simbol mora imati eksplicitno dodijeljenu vrijednost koja može biti postavljena samo jednom (operatorima `generate` i `believe`) i greška je koristiti simbol koji nema dodijeljenu vrijednost. Dodjela vrijednosti znači da se za simbol vjeruje da ima tu određenu vrijednost i ostala navedena svojstva.

Anonimne varijable simboli su koji mogu biti korišteni bez eksplicitno dodijeljene vrijednosti te se označavaju jednom zvjezdicom ispred imena: *A.

Osnovna namjena anonimnih varijabli je pohranjivanje neprepoznatih podataka pristiglih preko operatora receive, ili otkrivenih preko operatora decrypt. Anonimne varijable tretiraju se kao binarni podatak, tj. niz okteta, te mogu u kasnijoj uporabi "dobiti" pripadajući tip podataka (naredbom `believe`) i ponašati se kao obični simboli.

Anonimne varijable također ne mogu mijenjati svoju vrijednost kao ni simboli, ali zato postoje pojačane anonimne varijable koje se označavaju s dvije zvjezdice: **A. Takvoj vrsti varijabli može se mijenjati vrijednost i one se mogu više puta upotrebljavati, što je pogodno za programske petlje i slično.

Primjer razlike uporabe anonimnih i pojačanih anonimnih varijabli:

```
(receive A *a (decrypt K *a))
(receive A **a (decrypt K **a))
```

U prvom je primjeru anonimna varijabla *a koja prije nije bila upotrebljavana. Tada će prva (lijeva) pojava varijable *a biti dodjela vrijednosti (znači *a postaje ono što je prvo stiglo u poruci), a druga pojava u naredbi `decrypt` biti će samo provjera je li nam dekripcija uspješna, to jest, dekriptirani podatak mora odgovarati vrijednosti anonimne varijable *a.

U drugom se primjeru dva puta istoj varijabli dodjeljuje vrijednost – dakle, nema usporedbe, nego se stara vrijednost pojačane anonimne varijable samo prepisuje i **a poprima vrijednost niza podataka dobivenih operatorom decrypt. Obol će, pak, ispisati upozorenje ukoliko se dva puta u istoj naredbi istoj pojačanoj anonimnoj varijabli pridijeli nova vrijednost, ali neće zaustaviti izvođenje skripte.

3.4.1. Tipovi podataka

U Obolu, tip je svojstvo dodijeljeno vrijednosti simbola te ga programer može promijeniti. Postoje dvije vrste tipova podataka: jednostavni i složeni.

Jednostavni tipovi podataka su: `string, integer number, binary data.`

Neki od složenih tipova podataka su: `nonce, timestamp, keys, name, address, signature value itd.`

Pretpostavljeni je tip podataka binarni i svi se drugi tipovi moraju moći prevesti u binarni tip.

Pri sigurnosnim protokolima potrebno se je koncentrirati na sljedeća pitanja:

- manipuliranje lokalnim stanjem
- koje podatke treba kriptirati ili dekriptirati
- koje podatke treba verificirati ili digitalno potpisati
- što treba poslati
- što očekujemo primiti

Zajedno sa sintaksom, Obol ima 8 osnovnih operatora koji nam omogućuju koncentriranje na već navedena pitanja: `believe, generate, encrypt, decrypt, sign, verify, send i receive.` Postoje i drugi operatori, ali koriste se za manipuliranje paketima i interakciju s programskim okruženjem.

3.4.2. Svojstva simbola

Ovdje su dana neka svojstva koja se mogu pridijeliti simbolima u Obolu.

Tablica 3.1 Svojstva simbola u Obolu

Ime svojstva	Opis
`alg`	ime algoritma, koristi se s operatorima generate i believe. U gotovo svim slučajevima to je ime kriptografskog algoritma, te može biti dano samo kao ime algoritma (npr. "DES") ili kao puno ime algoritma po JCE standardu – "AES-OBF-PKCS5 Padding". Podržani su sljedeći algoritmi: DES, AES, Blowfish, DSA, RSA, SHA1.
`cipher-iv`	koristi se sa simbolima koji su ključevi za postavljanje inicijalizacijskog vektora kod CBC (Cypher Block Chaining) kriptiranja gdje se dodaje inicijalizacijski vektor kako bi se otežalo analiziranje kriptiranog teksta.
`MAC-type`	Koristi se sa operatorom sign u slučaju potpisa sa simetričnim ključem, kako bi se odabrao MAC standard. Tip je string koji sadržava ime MAC algoritma po JCE standardu
`number-of-bits`	veličina vrijednosti simbola u bitovima
`number-of-bytes`	veličina vrijednosti simbola u bajtovima
`Self`	postavlja točku (adresu i/ili port) na kojoj se očekuje prijem poruke
`signature-digest`	koristi se s asimetričnim ključevima (simbolima). Određuje algoritam sažetka poruke koji se koristi kod digitalnog potpisa. Pretpostavljeni je "SHA1"
`string-encoding`	određuje kodiranje znakova upotrijebljeno na dotičnom simbolu. Npr. "UTF-8"
`Timeout`	vrijeme isteka u milisekundama. Koristi se sa simbolom tipa `host` kako bi se zamijenilo pretpostavljeno vrijeme isteka za operaciju primanja poruka
`Type`	tip simbola
`Channel`	određuje vrstu korištenog kanala komunikacije. Moguće vrijednosti su `tcp`, `udp`, `multicast` i `deliverable`

4. ELEMENTI OBOLA

Obol je protokolni jezik, sintaksa mu se zasniva na sintaksi LISPa, te je osmišljena tako da bude što jednostavnija. Program se sastoji od liste izjava (naredbi). Pri implementaciji protokola potrebno je uzeti u obzir činjenicu da Obol opisuje stanje i protokole samo na jednoj strani komunikacijskog kanala te je sve što se događa na drugoj strani (ili drugim stranama) nagađanje. Tako sudionik u izvođenju sigurnosnog protokola može uzimati samo svoje stanje sa sigurnošću, i to lokalno stanje može biti promijenjeno samo njegovom eksplicitnom akcijom – kao što je npr. primanje poruke.

U Obolu svaka se naredba piše unutar oblih zagrada.

Obol ima 8 osnovnih operatora koji se mogu podijeliti u 2 skupine:

- operatori za upravljanje simbolima
- operatori za slaganje i prepoznavanje poruka

4.1. Operatori za upravljanje simbolima

Operatori za upravljanje simbolima su `believe` i `generate`. Glavna je razlika između tih operatora u porijeklu podataka: `generate` lokalno stvara podatke, dok se vanjski podaci, koji izvana pristignu u sustav, moraju potvrditi naredbom `believe`.

4.1.1. Believe

Naredba `believe`, uz naredbu `generate`, spada u naredbe kojima se mijenja lokalno stanje na jednoj strani komunikacijskog kanala u sigurnosnom protokolu. U Obolu se za neki podatak može „vjerovati“ da ima strukturu – na taj se način niz bitova pretvara u ključ za enkripciju. Koristimo ga kako bismo „pretpostavili“ tip i/ili strukturu podatka koji je primljen. Naredba `believe` također ima ugrađenu automatsku deklaraciju varijabli [3].

Naredba `believe` stvara novi simbol s određenom vrijednošću ili prethodno stvorenom simbolu dodaje novo svojstvo.

Forma naredbe:

```
(believe ime_simbola believe_atributi)
```

Simbol je varijabla za koju ćemo, ili za njezina svojstva, nešto “vjerovati". Jednom kad prođe kroz believe operator, vrijednost je simbola nepromjenjiva, osim ako se radi o pojačanoj anonimnoj varijabli. Svojstva simbola, nakon što je tom istom simbolu dodijeljena njegova vrijednost, mogu biti neograničeno nadograđivana.

Forma `believe_atributi` ima 3 inačice:

izvor_podataka -> navodi se samo izvor podataka, ili vrijednost:

```
(believe A 35624)
```

izvor_podataka tip ((svojstvo_simbola)) -> tip i svojstvo_simbola nisu obavezni, te se može navesti više svojstava simbola

```
(believe P "www.yes.no" host ((port 8080)))
```

((svojstvo_simbola)) -> lista s jednim ili više svojstava koje se dodjeljuju simbolu
```
(believe P ((port 80))
```

Izvor podataka je bilo koji izraz koji vraća simbol kao rezultat. U Obolu vrijednosti ne postoje kao samostalni entiteti, te nema definiranih jednostavnih tipova (kao u drugim programski jezicima). To znači da mi sami moramo simulirati jednostavne tipove podataka pretvarajući takve vrijednosti u jedinstveno imenovane simbole. Sljedeći izrazi vraćaju simbol:

- poznati simboli (vraćaju sami sebe)
- nizovi znakova, npr. "Obol", koji mogu sadržavati posebne znakove (\), kao u C-u i drugim programskim jezicima
- pozitivni cijeli brojevi
- binarni blok podataka u bazi64, npr. |AFR%&?#K|
- operator `load`
  ```
  (load "key/Kab.key")
  ```
- naredba `public-key`, koja vraća javni ključ iz para ključeva asimetričnog kriptografskog algoritma
  ```
  (public-key Kp)
  ```
- naredba `private-key`, koja vraća privatni ključ iz para ključeva asimetričnog kriptografskog algoritma
  ```
  (private-key KP)
  ```
- operator `key-part` koji vraća imenovanu komponentu ključa kao broj ili binarni niz
  ```
  (key-part pub "Modulus")
  ```
- operator `encrypt` koji vraća enkriptirani podatak kao binarni niz
  ```
  (encrypt Kd idA "Kea")
  ```
- operator `sign` koji vraća potpis kao niz bitova
  ```
  (believe *h1 (sign "SHA1" *c1))
  ```

Specifikacija tipa ovisi o izvoru podataka te, uz tipove koji se mogu stvoriti naredbom `generate`, operator `believe` obuhvaća i sljedeće tipove:

Tablica 4.1 Tipovi koji se mogu stvoriti operatorom believe

Ime tipa	Opis
`string`	niz znakova
`number`	pozitivni cijeli broj. Kod operatora `believe` operatora vrijednost broja može biti izražena u oktalnom, decimalnom ili heksadecimalnom sustavu
`binary`	niz bitova
`public key`	javni ključ, zahtijeva svojstvo alg. koje označava uz koji se kriptografski algoritam koristi (npr. "RSA"), te opcionalno

	svojstvo size koje određuje veličinu ključa u bitovima
`private key`	privatni ključ, zahtijeva ista svojstva kao javni ključ
`shared key`	neasimetrični ključ, također uz ista svojstva kao javni ključ
`host`	DNS ime hosta, IPv4 ili IPv6 adresa, može doći uz naznaku (svojstvo simbola) broja porta na koji se treba spojiti kako bi se pristupilo poslužitelju
`uri`	Uniform Resource Indicator tekstualni niz znakova
`file.out, file.in, file.inout`	datoteka upotrebljavana u svrhe komunikacije

4.1.2. Generate

Lokalno stanje kod sudionika komunikacijskog protokola možemo promijeniti i naredbom `generate`, tj. kreiranjem novih podataka. Kad su nam potrebni jednokratni binarni niz (*nonce*), vremenske oznake ili novi ključevi trebamo ih eksplicitno generirati.

Operator `generate` lokalno stvara novi simbol i njemu dodijeljenu vrijednost prema navedenom predlošku, a moguće je i prethodno stvorenom simbolu dodijeliti novo svojstvo. Programsko okruženje stvara vrijednost simbola koja je uvijek složenog podatkovnog tipa.

```
(Generate symbolname type attributes)
```

`Symbolname` mora biti dotad neiskorišteno. Tipovi koji se mogu stvoriti naredbom generate su opisani u slijedećoj tablici.

Tablica 4.2 Tipovi koji se mogu stvoriti operatorom generate

Ime tipa	Opis
`nonce veličina_u_bitovima`	stvara slučajni niz bitova zadane veličine (generate Na nonce 128)
`Shared-key ime_algoritma veličina_u_bitovima`	stvara novi ključ zadane veličine za navedeni simetrični kriptografski algoritam (generate Kab AES 192)
`keypair ime_algoritma veličina_u_bitovima`	stvara novi par ključeva zadane veličine za navedeni asimetrični kriptografski algoritam
`key-agree privatni_ključ javni ključ`	stvara dijeljeni (tajni) ključ za simetrični kriptografski algoritam na temelju para ključeva asimetričnog kriptografskog algoritma. Novi se ključ se potvrditi naredbom `believe`: `(generate Kab key-agree Ka-1 Kab)`

	(believe Kab ((type shared-key)(alg AES)))
timestamp	stvara vremensku oznaku koja predstavlja sadašnji trenutak (generate Ta timestamp)
hash ime_algoritma izvor	stvara sažetak teksta (stringa) ili broja po navedenom algoritmu. (generate md hash SHA1 "a") Naredba sign može također biti upotrijebljena za stvaranje sažetka poruke, ali vrlo je vjerojatno da rezultat neće biti isti kao kod generate naredbe zbog različitog načina sažimanja poruke. Naredba generate hash jednostavno spaja binarne vrijednosti poruke koju treba sažeti te radi sažetak iz dobivenog niza, dok naredba sign konstruira kompletnu poruku skupa s njenim oblicima formatiranja te u sažetak poruke ulaze i metapodaci dodani njenim formatiranjem.

Generate se također koristi u kombinaciji s believe kako bi se kreirale varijable određenog tipa – u sljedećem primjeru prvo se generira 128-bitni *nonce* te se dobivena vrijednost pridjeljuje varijabli R. Nakon toga, sadržaj u R je pretvoren u 128-bitni dijeljeni ključ i pridijeljen varijabli Key [3].

```
(generate R nonce 128)
(believe Key R shared-key ((alg AES)(size 128)))
```

4.2. Operatori za slaganje i prepoznavanje poruka

Poruke se u Obolu tretiraju kao osnovna jedinica komunikacije, te se interpretacijom (i povezanim problemima) i strukturom poruka bavi programsko okruženje. Operatori za slaganje i prepoznavanje poruka su: send, receive, encrypt, decrypt, sign i verify.

Operator send "naređuje" programskom okruženju da pošalje poruku po predlošku na navedeno odredište. Poruka može sadržavati operacije enkripcije i digitalnog potpisa (operatori sign i encrypt).

Operator receive je komplementaran operatoru send te obavještava programsko okruženje otkud treba primiti poruku i, po predlošku poruke, kako poruka treba izgledati. Operacija primanja poruke blokira izvođenje pa se skripta ne može nastaviti izvoditi sve dok odgovarajuća poruka ne bude primljena. Predložak poruke također može sadržavati operacije dekripcije i verifikacije (decrypt i verify operatori), kao i anonimne varijable.

Operator `encrypt` sličan je operatoru `send`, samo što se poruka u kriptiranom obliku šalje "preko ključa" umjesto preko komunikacijskog kanala.

Operator `decrypt` sličan je operatoru `receive`, samo što se poruka prima iz kriptiranog teksta, također preko ključa.

Operator `sign` iz ključa i specifikacije poruke stvara potpis. Vrsta potpisa ovisi o tipu ključa koji se upotrebljava.

Operator `verify` uzima ključ, potpis i specifikaciju poruke te pokušava verificirati potpis.

4.2.1. Send

Slanje poruka središnja je aktivnost svakog protokola.

```
(send odredište specifikacija_poruke)
```

Primatelj poruke definiran je u elementu `odredište`.

Specifikacija_poruke slijed je jednog ili više izvora podataka (*izvor_podataka* definiran kod naredbe believe)

Primjer:

```
(send KDC2   idS N2 *c1 *c2)
```

Kako bi se izbjegle zabune prilikom razlikovanja odredišta i specifikacije poruke, dogovor je da programer u Obolu iza odredišta doda 2 ili 3 razmaka.

Kako bi se poruka što pouzdanije prenijela, moramo znati gdje je treba poslati. Nekoliko je načina za definiranje primatelja poruke, a Obol trenutačno podržava metode koje mogu biti odabrane preko sljedećih tipova simbola:

Tablica 4.3 Opcije za definiranje primatelja poruke

Ime opcije	Opis
`host`	tekstualni niz koji predstavlja DNS adresu ili IPv4/IPv6 identifikator s brojem porta na koji se treba spojiti kako bi se pristupilo željenom odredištu
`Uri`	tekstualni niz koji predstavlja URI. Trenutačna (Java) implementacija programskog okruženja pretvara URI u URL pa pokušava otvoriti konekciju na to što URL pokazuje
`file.out, file.in, file.inout`	tekstualni niz koji predstavlja ime datoteke u lokalnom okruženju. Ovaj posebni tip dozvoljava čitanje, pisanje ili oboje u datoteku ili iz nje. `File.out` jedini je način za spremanje ključa (ili drugih podataka) u datoteku, što kasnije može biti pročitano load operatorom. `File.in` služi za čitanje poruka i rijetko se koristi. `File.inout` dozvoljava i čitanje i pisanje u datoteku te se koristi kako bi se osvježavali podaci u istoj datoteci.

Poruke u Obolu su u ASCII formatu. Svaki element poruke kodiran je u heksadecimalnom sustavu i stavljen u zagrade, te je svaka poruka kao cjelina stavljena u zagrade. Jednostavni ASCII format izabran je kako bi se pojednostavnilo pronalaženje grešaka i provjeravanje vjerodostojnosti poruka. Struktura ugniježđenih zagrada olakšava parsiranje i prevođenje programa [3].

Primjer poruke koja sadrži vremensku oznaku od 4 bajta i AES ključ od 128 bitova:

```
((c3c249ae)(1c7dc3fd915199830a7c1959e7aa9d1f))
```

Poruke su kodirane tako da ne sadrže nikakve informacije o tome kako dekodirati pojedine elemente poruke – ti se elementi ponašaju kao nizovi bitova bez određenog značenja. Poruka je strukturirana tako da je oblikovana dovoljno, jer primatelj mora već imati potrebne informacije kako bi dekodirao poruku. Dekodiranje kod primatelja obavlja se slijedeći uzorak koji je određen u receive naredbi prilikom primanja poruke: ako je prvi element uzorka „nonce“, heksadecimalno kodirani niz bitova, u prvom elementu poruke se također tretira kao nonce, ako je prvi element uzorka niz, prvi element poruke se tretira kao niz, i tako dalje.

4.2.2. Receive

Receive je dosada najkompleksnija naredba. Kako u Obolu nije moguće izraziti komunikacijske kanale kao takve, primitak poruke ne može biti određen ovisno o tome kako ili gdje se poruka pojavi. Zato primitak poruke mora biti definiran u smislu određivanja očekivanog sadržaja poruke [3].

```
(receive naznaka_izvora predložak_poruke)
```

Naznaka_izvora ovisi o tome na koji se način prima poruka, ali uglavnom se koristi kako bi se olakšao izbor poruka. Naznaka_izvora može biti anonimna varijabla, što bi značilo "primi poruku i zapamti odredište za buduću uporabu". Ta ista anonimna varijabla može biti korištena za slanje odgovora na poruku.

Predložak_poruke je slijed jednog ili više izvora podataka (izvor_podataka opisan kod naredbe believe), sa sljedećim iznimkama:

- operator `decrypt` zamjenjuje `encrypt`
- operator `verify` zamjenjuje `sign`
- anonimne varijable mogu se pojavljivati po prvi put

Primanje poruke

Primanje poruka ovisi o 2 čimbenika – odredištu isporuke i prihvatljivom sadržaju. Pogreška u prvom znači da poruka nije isporučena, dok pogreška u drugom čimbeniku znači da nije bila prihvatljiva.

Za primjer ćemo uzeti sljedeću poruku (iz Yahalom protokola):

$A \rightarrow B: \{A,K_{ab}\}K_{bs}, \{N_b\}K_{ab}$

Implementacija poruke je sljedeća:

```
1 (receive *1 *2)
2 (decrypt Kbs *1 A *Kab)
```

```
3 (believe Kab *Kab shared-key ((alg AES)(size 128)))
4 (decrypt Kab *2 Nb)
```

Objašnjenje primjera:

Očekujemo poruku koja se sastoji od dva odvojena elementa koji su spremljeni u zasebne anonimne varijable. Tako složen izraz vraća poruku.

Prva komponenta mora se moći dekriptirati pomoću ključa spremljenog u varijablu K_{bs}, i unutar nje mora biti moguće definirati ponovo dvije komponente, od kojih se prva podudara s vrijednošću spremljenom u varijabli A. Izvršenje ove linije koda uspijeva samo ako su oba uvjeta točna. Druga se komponenta nakon dekriptiranja dodjeljuje anonimnoj varijabli $*K_{ab}$

Sadržaj varijable $*K_{ab}$ pretvara se u dijeljeni ključ i sprema u varijablu K_{ab}.

Deklarira se da se sadržaj iz N_b u 2. anonimnoj varijabli iz poruke mora dobiti dekripcijom pomoću ključa spremljenog u K_{ab}.

4.2.3. Encrypt

Način na koji se enkriptira varijabla ovisi o tipu ključa – programsko će okruženje ovisno o tome pokrenuti određene potprograme za tu vrstu enkripcije.

```
kriptirani_tekst <- (encrypt ključ predložak_poruke)
```

Ključ je simbol koji sadrži tajni ili javni ključ, ili operator `key-part public-key`. Ključ prije enkripcije mora proći „believe“ evaluaciju – mora se eksplicitno „vjerovati da je dobar“ [3]. Na primjer:

```
(believe K (generate shared-key ((alg AES)(size 128))))
(encrypt K “secret”)
```

Predložak_poruke slijed je jednog ili više izvora podataka (izvor_podataka opisan kod naredbe believe).

Operator encrypt vraća simbol koji sadrži kriptirani tekst kao binarni objekt u ASCII formatu (kao kod operatora `send`).

4.2.4. Decrypt

Sigurnost da je operacija uspjela i da smo pouzdano dobili točan rezultat prvi je i osnovni zahtjev dekriptiranja. Tu sigurnost rezultata postižemo dodavanjem redundantnih informacija u paket (kao što je sažetak poruke). Dekriptirani sadržaj mora biti ispitan kako bismo odlučili ispunjava li zahtjeve sigurnosnog protokola.

Operator decrypt ima dvije forme:

```
(decrypt ključ predložak_poruke)
(decrypt (ključ kriptirani_tekst) predložak_poruke)
```

Prva forma dohvaća kriptirani tekst iz trenutnog konteksta, dok druga eksplicitno navodi kriptirani tekst.

Ključ je simbol koji sadrži simetrični ili asimetrični ključ, ili neki od operatora koji izdvajaju

privatni ili javni dio ključa iz para ključeva asimetričnog algoritma.

Predložak_poruke sličan je predlošku poruke opisanom kod operatora `receive`, samo što su ovdje malo pojednostavljeni algoritmi slaganja poruke s predloškom.

Kriptirani_tekst je simbol koji sadržava binarni objekt kao onaj što ga vraća operacija `encrypt`.

Primjer:

```
(receive *1 (decrypt K idC))
```

-> prima poruku i sprema adresu u anonimnu varijablu *1 i očekuje da će u jedinom podatku kad se dekriptira biti vrijednost jednaka vrijednosti već prije poznatog simbola idC

Operator decrypt može izazvati pogrešku na dva načina:

- ukoliko propadne dekripcija, ili
- ukoliko se rezultat ne složi s navedenim predloškom poruke

Dekripcija može propasti zbog uporabe krivog ključa, krivog kriptiranog teksta, modifikacije istog, ili neke druge slične pogreške. Zapravo, nemoguće je točno odrediti zašto propada dekripcija. Neslaganje s predloškom poruka pokazuje da je sama operacija dekripcije uspjela, ali da se rezultat ne poklapa sa zadanim predloškom.

4.2.5. Sign

`Sign` operator vraća objekt koji sadržava digitalni potpis.

```
potpis <- (sign ključ poruka)
```

Ključ je simbol koji sadrži simetrični ključ, privatni ključ, `private-key` operator primijenjen na asimetričnom paru ključeva, ili tekstualnu specifikaciju vrste sažetka.

Poruka je slijed jednog ili više izvora podataka (opisanih kod operatora `believe`).

Operator `sign` može koristiti i simetrične i asimetrične ključeve za potpis. Ukoliko koristi simetrični ključ, utoliko je rezultat MAC potpis, a ako se koristi asimetrični ključ, rezultat je digitalni potpis koji osigurava integritet i neporecivost potpisane poruke.

4.2.6. Verify

Verify operator ima dvije forme:

```
(verify ključ poruka)
(verify (ključ potpis) poruka)
```

Prva forma dohvaća potpis iz konteksta (npr. ako je `verify` ugniježđen u naredbu `receive`), dok druga eksplicitno navodi izvor potpisa.

Ključ je simbol koji sadrži simetrični ključ, privatni ključ, `private-key` operator primijenjen na asimetričnom paru ključeva, ili tekstualnu specifikaciju vrste sažetka.

Poruka je slijed jednog ili više izvora podataka (opisanih kod operatora `believe`).

Potpis je binarni objekt u ASCII formatu, kao onaj koji vraća operator `sign`.

Operator `verify` je samo funkcija provjere, ne vraća nikakvu vrijednost pa se ništa neće dogoditi ako je verifikacija uspješna. Međutim, ako verifikacija ne uspije, to će u većini slučajeva biti smatrano fatalnom greškom i izvođenje skripte prekinut će se.

4.3. Udaljena stanja

Primljena poruka uglavnom se sastoji od poznatih elemenata, kao i nekih nepoznatih. Poznati elementi u uzorku poruke pokazuju da bi oni trebali biti podudarni u primljenoj poruci. Nepoznati elementi poruke, označeni sa zvjezdicom, pokazuje da ta varijabla (element poruke) odgovara elementima uzorka poruke koji su promjenjivi. Varijable označene sa zvjezdicom zovemo „anonimne“ varijable. Vrijednosti anonimne varijable možemo nakon primitka poruke pridijeliti značenje pomoću `believe` naredbe, ili je možemo poslati dalje bez uporabe [3].

4.4. Spremanje i dohvat ključa iz datoteke

Programer u Obolu može kreirati ključ i trajno ga pohraniti na računalo na kojem se trenutačno nalazi u obliku datoteke s nastavkom *.key. Tako spremljen ključ može se dohvaćati iz drugih skripti koje se izvode na lokalnom računalu, te se takav način pohrane ključeva može prikazati kao alternativa komunikaciji pomoću operatora `send` i `receive`.

Primjer spremanja ključa na računalo [3]:

```
(believe f "c:/k.key" file.out)
(generate key shared-key AES 128)
(send f key)
```

Nakon toga, svatko tko na istom računalu izvodi Obol skripte može dohvatiti ključ slijedećom naredbom:

```
(believe Kab (load "c:/k.key") shared-key ((alg AES)))
```

4.5. Ostali operatori

4.5.1. Cond

Opereatorom cond postavlja se uvjet čija istinitost određuje daljnji tijek izvođenja programa. Objedinjuje značenje naredbi if-then-else i switch-case korištenih u drugim jezicima. Nema povratne vrijednosti pa se koristi samo za kontrolu toka izvođenja programa. Cijeli pripadajući blok se nalazi u zagradama, a unutar njega može biti više postavljenih uvjeta, te iza svakog slijedi jedna ili više njemu pripadajućih naredba koje se izvršavaju ovisno o istinitosti uvjeta. Iza uvjeta može biti i prazna naredba – tada istinitost uvjeta nema nikakvog utjecaja na izvođenje programa.

Sintaksa:

```
(cond
((uvjet1) ((naredba1) (naredba2))
```

```
((uvjet2) () )
...
((uvjetX) (naredbaX))
)
```

Uvjet je izraz koji može biti točan ili lažan. Može se sastojati od:

izraza usporedbe: ==, !=, <=, >=, < ili >

not operatora koji će negirati izraz usporedbe kojem prethodi

null operatora koji provjerava da li je vrijednost simbola ili anonimne varijable postavljena na lažno ili ne (istinita je)

operator koji dohvaća neko svojstvo simbola

operatori receive, decrypt i verify

Naredba može biti bilo koji Obol operator s pripadajućim argumentima. Naredbe se izvršavaju redom kojim se pojavljuju (ako je njima pripadajući uvjet ispunjen).

Unutar cond bloka se može nalaziti još jedan ili više cond blokova pa se to naziva ugniježđeni cond (nested cond).

Primjer [2]:

```
      (cond
        ((== 1 1) ())
        ((== 1 1) ((generate **n nonce 128)))
        ((== 1 1) ([print "(== 1 1) test works"]
                     (believe **n 123)
                     (generate **n nonce 128)))
        ((== 1 0) ([print "   (== 1 0) does NOT work"]))
        ((== 1 1) (
                 (cond ((== 1 1)([print " nested (cond) works"])))))
)
```

U ovom primjeru će se izvršiti sve naredbe osim one koja ispisuje " (== 0 1) does NOT work), zbog toga što uvjet nije točan, to jest, 0 i 1 nisu međusobno jednaki.

4.5.2. If…(else)

Pomoću if…(else) operatora ostvaruje se grananje, kao i u većini drugih programskih jezika. Kao i operator cond nema povratnu vrijednost te služi samo za kontrolu toka. Else dio nije obavezan.

Sintaksa:

```
(if (uvjet)
      ((naredba1) (naredba2)…(naredbaX))
else ((naredba1) (naredba2) … (naredbaY))
```

U if naredbu je također moguće ugnijezditi druge složene naredbe, kao što je cond.

Primjer [2]:

```
[print "Testing (if) with random input"]
(generate *coin nonce 1)
(believe *coin ((type number)))
(if (== 0 *coin)
  ([print "picked zero, flipping..."](believe **coin 1))
  else ([print "picked one, flipping..."](believe **coin 0)))
(if (== 0 *coin)
  ([print "flipped coin is now zero (should be one above)"])
  else ([print "flipped coin is now zero (should be zero above)"]))
```

4.5.3. Loop

Loop naredbom se ostvaruje petlja, tj. ponavljanje iste akcije određeni broj puta. Loop ponavlja blok Obol operatora dok neka naredba za kontrolu loop petlje ne izazove izlaz ili dok se ne izvrši operator endloop. Kod naredbe loop je, također, moguće ugnježđivanje ostalih složenih naredbi (cond i if). Postoje 2 načina kako naredbom loop možemo ostvariti petlju:

Korištenjem naredbe endloop za zaustavljanje. Endloop stavljamo kao i svaku drugu naredbu, obično u kombinaciji sa if ili cond, kada se gleda ispunjenje nekog uvjeta da bi se petlja završila.

Primjer [2]:

```
[print "Testing (loop) with (endloop)"]
(believe **n 0)
(loop
  (cond
    ((== 1 1)
      ((cond ((== 1 1)([print "nested cond, in loop"])))))
    ((== 0 *n)
      ([print "changing value"](believe **n 1)))
    ((== 1 *n)
      ([print "changing value"](believe **n 2)))
    ((>= 2 *n)
      ([print "calling endloop" (endloop)))
  )
)
[print "Current value = " *n]
```

Određivanjem broja iteracija petlje unaprijed – pomoću ključne riječi `dotimes`.

Primjer [2]:

```
[print "Testing (loop (dotimes ...) ...)"]
(believe **n 5)
(loop (dotimes *n)
[print "Iteration..."])
```

`Endloop` operator uvijek djeluje na "najbliži" loop operator – tako je spriječena mogućnost, kod ugniježđenih petlji, zatvaranja "vanjskog" loop-a "unutarnjim" `endloop` operatorom.

`Loop-reset-symbol` operator olakšava ispravnu uporabu simbola unutar petlje tako što omogućava uništavanje jedne ili više anonimne varijable.

`(loop-reset-symbol` *`simbol)`*

Simbol može biti samo pojačana anonimna varijabla, i greška je koristiti ovaj operator na "običnim" simbolima ili anonimnim varijablama ili na onima koji nemaju već pridjeljenu vrijednost (simbol mora postojati da bi se uništio). `Loop-reset-symbol` se može koristiti samo unutar `loop` petlje.

5. PROGRAMSKO OKRUŽENJE LOBO I METANAREDBE

5.1. Programsko okruženje

Programsko okruženje Obola zove se LOBO i sadrži prevoditelj koji radi na standardni način: uzme izraz i parametre iz njegove okoline kao ulaznu vrijednost, izvede izraz u toj okolini i vrati izlazne vrijednosti. Kako svaki krovni izraz može sadržavati ugniježđene izraze iste vrste, uvedena je i rekurzivna struktura. LOBO podržava paralelno izvođenje više skripta i ima API koji aplikacije mogu koristiti kako bi mu slali skripte i preuzimali rezultate nakon izvođenja.

Ako se aplikacija ne izvodi u istom adresnom prostoru kao i LOBO, komunikacija se može odvijati preko prividnog poslužitelja (*proxy*). Aplikacija će pozvati inicijalizacijsku metodu na prividnom poslužitelju koji će zatim uspostaviti komunikaciju preko sigurnog kanala s LOBOm i ponuditi mu metodu uzimanje skripta kad su spremne.

Programski jezik ne pravi nikakve pretpostavke o tome kako se komunicira porukama, čak nije bitno jesu li poruke prenesene istim putem tijekom trajanja jedne sjednice protokola. Programsko okruženje ima sačuvane sve poruke koje su stigle pa se zatim pomoću pripadajućeg algoritma određuje hoće li poruka biti primljena od pripadajuće instance skripte. Ne prave se ni pretpostavke kako su poruke prezentirane ili strukturirane. Točan način prijenosa poruka upravljan je od strane programskog okruženja, i modularan je kako bi se novi načini komunikacije mogli dodavati. Načinom komunikacije može se upravljati kroz programsko okruženje (preko skripta), ili preko parametara koji se iz aplikacije predaju programskom okruženju. Ovo svojstvo omogućuje Obol programu adaptaciju na promjene u okolini i načinu komuniciranja [3].

5.2. Metanaredbe

Metanaredbe omogućuju skriptama izravnu komunikaciju s programskim okruženjem, kako bi se definiralo što je potrebno obaviti izvan skripte, od debugiranja**,** provjere jednakosti (*assert*) do interakcije s aplikacijama ili pozivanja drugih skripta. Metanaredbe nemaju nikakve veze s izražavanjem sigurnosnih svojstava protokola, već se koriste za razne druge stvari koje su potrebne da bi se skripta mogla pravilno izvoditi – kao što su debugiranje, formati i interakcija s aplikacijama

5.2.1. Print

Print se javlja u slijedećim oblicima:

```
[print print-spec*]
```

```
[printall print-spec*]
```

Ove će naredbe ispisati tražene informacije na standardni izlaz, a razlikuju se u količini ispisanih detalja. `Print` forma ispisat će samo vrijednost simbola u tekstualnoj ili (kod

binarnih vrijednosti) heksadecimalnoj formi. Printall forma će, pak, ispisati sve informacije koje se povezuju sa navedenim simbolom.

`Print-spec` može se ne pojaviti ili pojaviti nekoliko puta, te se sastoji od citiranih stringova ili poznatih simbola. Ukoliko se `print-spec` ne navede, utoliko će se ispisati samo prazna linija.

Primjena i sintaksa naredbe `print` iste su kao kod ostalih programskih jezika – u navodnike se stavlja niz koji doslovno treba biti ispisan, a izvan navodnika varijable ili izrazi čija vrijednost ide na izlaz. Primjer:

```
[print "Nonce Na jest:", Na]
```

Ispisuje:

```
Nonce Na jest: 0x0cc175b9c0f1b6a831c399e269772661
```

Jer je to vrijednost varijable Na.

5.2.2. Assert

```
[assert A simbol_usporedbe B]
```

Pomoću naredbe assert provjeravamo određeni uvjet bitan za nastavak programa. Ako je izraz u naredbi istinit („true“), izvođenje programa se nastavlja bez ikakvih posljedica, a ako je lažan, izvođenje se prekida i programsko okruženje javlja pogrešku o neistinitom izrazu. Simbol_usporedbe može biti ==, !=, <, >, <= ili >=. Svi tipovi simbola mogu biti testirani na jednakost, ali ostale usporedbe tipa veći i manji imaju smisla samo za tipove `number` i `timestamp`.

5.2.3. Format

Format naredba određuje koji će se format koristiti za prikazivanje poruka.

```
[format ime-formata]
```

Ime-formata mora biti znakovni niz (u navodnicima ukoliko sadrži razmake).

Primjer:

```
[format default]
[format spki]
```

5.2.4. Self

Self naredba specificira gdje i kako se primaju poruke.

```
[self delivery-point properties]
[self delivery-point]
[self delivery-point format-name]
[self delivery-point format-name properties]
```

Delivery-point parametar može biti broj, niz u navodnicima, ili već poznat simbol.

Parametar *format-name* može biti niz u navodnicima ili već poznati simbol.

Parametar *svojstvo* sličan je svojstvima simbola, gdje su podržana sljedeća svojstva i njihove vrijednosti:

Tablica 5.1 Svojstva kod naredbe self

Ime svojstva	Opis
`receive-timeout` *broj*	maksimalni broj milisekundi koji je potrebno čekati na primanje poruke. Ako se postavi, vrijednost može biti prepisana svakom pojavom operatora receive gdje se isto može postaviti ova vrijednost.
`match-ordering` *string*	određuje kako se radi slaganje poruka, to jest, je li bitan redoslijed pristizanja elemenata u poruci ili ne. Ako je redoslijed pristizanja bitan, elementi koji stižu moraju se slagati s elementima koji su navedeni u predlošku poruke naredbe receive, s lijeva na desno. Vrijednost može biti postavljena na "ordered" ili "unordered".
`match-timeout` *broj*	maksimalni broj u milisekundama koji je potrebno čekati da bi se podaci koji pristižu poklopili s predloškom poruke. Ovo svojstvo je značajno kod načina "unordered", što može dovesti do velike složenosti pri slaganju elemenata.

5.2.5. Returns

Returns naredba komplementarna je naredbi input i određuje koje simbole skripta vraća u programsko okruženje (povratne vrijednosti skripte).

Formati naredbe su:

```
[returns simbol]
[returns simbol tip]
```

Simbol je simbol iz skripte koji će biti dostupan izvana, dok je opcionalni parametar tip tretiran kao meta-podatak.

5.2.6. Input

Ova naredba spada u naredbe za interakciju s aplikacijama. Određuje formu ulaznih podataka omogućenih preko okruženja u kojem se izvodi Obol skripta.

```
[input input-symbol type]
[input input-symbol type svojstva]
```

Input-symbol je ime simbola koje je dodijeljeno ulaznom podatku, dok je *type* Obol tip tog simbola. Opcionalni parametar *svojstva* je zapravo lista svojstava, gdje su dva svojstva predodređena: svojstvo "default" koje određuje defaultne vrijednosti i svojstvo-ključna riječ ":doc" koje dokumentira zahtjeve inputa.

Input zahtjevi mogu biti navedeni proizvoljan broj puta u skripti, ali normalno izvršavanje programa neće se nastaviti sve dok svi input zahtjevi ne budu ispunjeni (osim onih s predodređenim vrijednostima). To se svojstvo može iskoristiti za izgradnju interaktivnih Obol skripti, ako se upotrebljavaju input izjave na pravim mjestima u skriptama, navodeći skriptu na stajanje i čekanje novih ulaznih podataka prije nastavka.

5.2.7. Use

Naredbom use unutar jedne Obol skripte koristimo neku drugu Obol skriptu, koja tada ima status podskripte, uz input i return metanaredbe. To nam omogućuje da Obol skripte tretiramo kao potprograme u drugim programskim jezicima, uz ubacivanje ulaznih da bismo dobili izlazne vrijednosti.

5.2.8. Require

Ova metanaredba omogućava skripti da odredi neke zahtjeve. Neke će skripte zahtijevati da programsko okruženje implementira određeni format (npr. neka metoda kriptiranja) ili nagovijestiti da će koristiti određenu podskriptu.

`[require class` *`classname`*`]`

`[require script` *`scriptname`*`]`

`Classname` i `scriptname` predstavljaju ime klase (formata) koji će biti učitan i ime podskripte koja će biti potrebna u izvođenju skripte.

6. PRIMJERI

6.1. Pokretanje Obola

Obol programsko sučelje pisano je u Javi, te dolazi u obliku .jar datoteke (Obol.jar). Prije pokretanje Obola potrebno se u Command promptu pozicionirati u isti direktorij gdje se nalazi te postaviti classpath (set classpath antlr.jar). Obol se može pokrenuti na dva načina:

Slika 6.1 Pokretanje Obola

```
java –jar Obol.jar
```

Pokreće se Obol programsko sučelje spremno za upisivanje naredbi. Na taj način naredbe se upisuju jedna po jedna, te se izvršavaju odmah čim su napisane. Međutim, prethodno upisane naredbe nigdje nisu spremljene, te je cijeli program potrebno upisivati ponovo ukoliko dođe do neke pogreške prilikom izvršenja naredbi.

```
java –jar Obol.jar <"ime skripte"
```

Pokreće prethodno napisanu skriptu (ime skripte uključuje i ekstenziju), isto izvršava red po red, ali je ovako moguće nakon dojave o grešci promijeniti spornu naredbu u skripti, spremiti promjenu, te istu skriptu izvesti bez ponovnog tipkanja naredbi koje nisu bile problematične.

6.2. Primjer 1

Slanje podataka (u ovom slučaju odsječak teksta) s poslužitelja A na poslužitelj B.

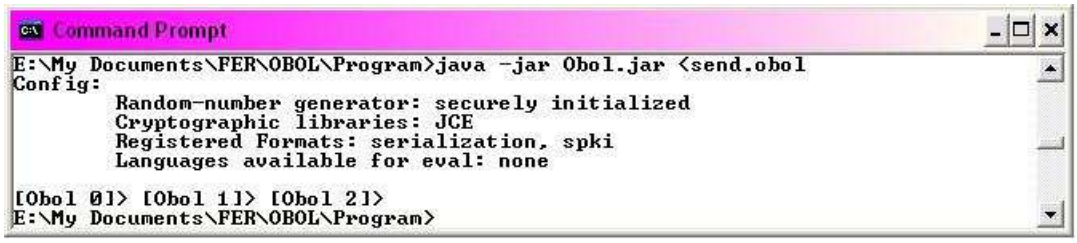
Poslužitelj A:

```
(believe R "localhost:2505" host)
(send R "Ovo je test" "I ovo")
```

Poslužitelj B:

```
[self "localhost:2505"]
(receive *1 *2 *3)
[print *1]
[print *2]
[print *3]
```

Rezultat:



Slika 6.2 Strana koja šalje podatke

Kako u samom programu nema nikakve naredbe koja bi ispisivala nešto na ekran, ispisuju se samo brojevi naredbi.

Slika 6.3 Strana koja prima podatke

Kad su podaci zaprimljeni poslužitelj B na standardni izlaz ispisuje prvo adresu s koje je dobio podatke, a zatim ostatak podataka u obliku teksta.

6.3. Primjer 2: Digitalni pečat

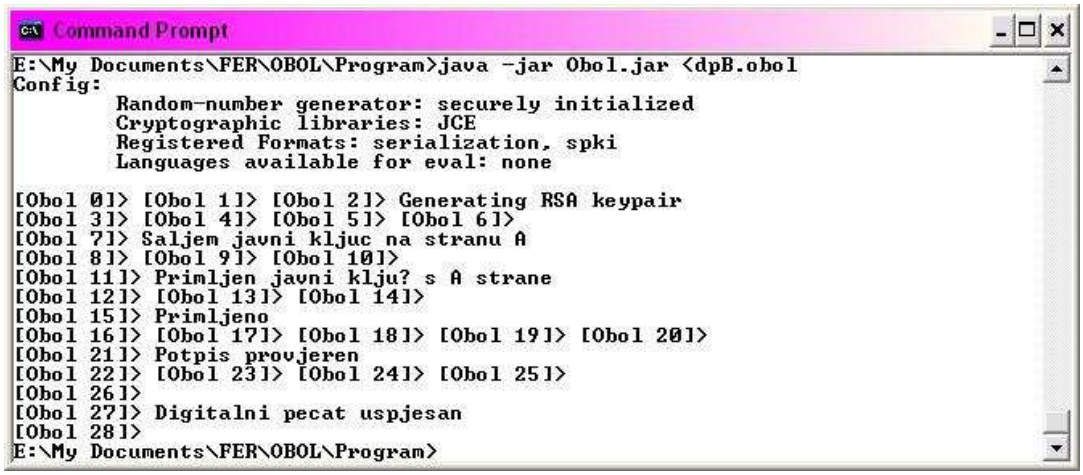
```
;;strana koja šalje pečat
[self "localhost:2506"]
(believe B "localhost:2505" host)
(receive *1 *2)
[print]
[print "Primljen javni kljuc s B strane"]
(believe K_b_pub *2 public-key (alg RSA))
(generate K shared-key AES 128)
(believe *Na K ((type binary)))
(believe *poruka2 "Digitalni pecat uspjesan")
(believe *c1 (encrypt K *poruka2))
(believe *c2 (encrypt K_b_pub *Na))
[print]
[print "Gotova digitalna omotnica"]
(believe *h1 (sign "SHA1" *c1))
(believe *h2 (sign "SHA1" *c2))
(generate K_a keypair RSA 1024)
```

```
(believe K_a_pub (public-key K_a))
(believe K_a_priv (private-key K_a))
(send B K_a_pub)
[print]
[print "Poslan moj javni ključ"]
(believe *c3 (encrypt K_a_priv *h1))
(believe *c4 (encrypt K_a_priv *h2))
(send B *c1 *c2 *c3 *c4)
[print]
[print "Digitalni pecat uspjesno poslan"]
```

Slika 6.4 Strana koja šalje pečat

```
;;strana koja prima pečat
[self "localhost:2505"]
(believe A "localhost:2506" host)
[print "Generating RSA keypair"]
(generate K_b keypair RSA 1024)
(believe K_b_pub (public-key K_b))
(believe K_b_priv (private-key K_b))
[print]
[print "Saljem javni kljuc na stranu A"]
(send A K_b_pub)
(receive *1 *4)
[print]
[print "Primljen javni ključ s A strane"]
(believe K_a_pub *4 public-key (alg RSA))
(receive *1 *c1 *c2 *c3 *c4)
[print]
[print "Primljeno"]
(decrypt (K_a_pub *c3) *h1)
(decrypt (K_a_pub *c4) *h2)
(verify ("SHA1" *h1) *c1)
(verify ("SHA1" *h2) *c2)
```

```
[print]
[print "Potpis provjeren"]
(decrypt (K_b_priv *c2) *Na)
(believe K *Na shared-key ((alg AES)))
(decrypt (K *c1) *rezultat)
[print]
[print]
[print *rezultat]
```



Slika 6.5 Strana koja prima pečat

7. PRAKTIČNI RAD

7.1. Kerberos

Kerberos je sustav za autentifikaciju i dodjelu tajnih ključeva. Njegov osnovni element je Kerberos čvor kojemu korisnik šalje zahtjev za autentifikaciju i dozvolu za pristup nekom poslužitelju.

Kerberos čvor se sastoji od AS (Authentication Server), TGS (Ticket Granting Server) i baze podataka. AS provjerava identitet korisnika (Alice) i pridjeljuje joj tajni sjednički ključ za komunikaciju sa TGS-om. TGS Alice dodjeljuje dozvolu i tajni sjednički ključ za komunikaciju s Bobom. Nakon toga, Alice i Bob se, pomoću generiranog sjedničkog ključa i vremenskih oznaka, autentificiraju i mogu početi razmjenu podataka.

Kod Kerberosa su vremenskim oznakama spriječeni napadi ponovnim odašiljanjem snimljenih starih poruka, kao i napad prekidanjem [1].

Poruke koje se razmjenjuju prilikom traženja dopusnice za poslužitelj (slika na slijedećoj stranici:

Klijent –> AS: M1 = (IDc, N1)

Klijent šalje KDC-u svoj identifikator i slučajni broj N1

AS –> Klijent: M2 = E{(N1, K1, C1), Kc}

C1 = E{(IDc, IDg, Ts1, Te1, K1),Kg}

Kerberos čvor pronađe klijenta u tablici koju čuva lokalno, te njegovim tajnim ključem kriptira ulaznicu za TGS, novi tajni sjednički ključ između klijenta i TGS-a, te slučajni broj koji mu je klijent poslao (kako bi AS dokazao da je on taj koji je primio prošlu poruku od klijenta). U ulaznici se nalazi identifikator klijenta, identifikator TGS-a, vremenske oznake početka i završetka valjanosti ulaznice i isti sjednički ključ koji je dobio klijent, sve to kriptirano tajnim ključem poslužitelja. Klijent raspakira poruku, provjeri broj N1 i spremi ključ K1, a ulaznicu C1 ne dira nego u istom obliku ugradi u slijedeću poruku.

Klijent –> TGS: M3 = (IDs, N2, C1, C2)

C2 = E{(IDc, T1),K1}

Klijent zatim TGSu šalje identifikator poslužitelja (usluge) kojem želi pristupiti, novi generirani slučajni broj, ulaznicu, te svoj autentifikator kriptiran novim sjedničkim ključem. Autentifikator je identifikator korisnika s vremenskom oznakom. Ako je poruka došla na pravu adresu (TGS-u), on će svojim tajnim ključem dekriptirati ulaznicu C1, otkriti tajni sjednički ključ i moći otkriti autentifikator, kojim korisnik dokazuje svoj identitet. Tako se osigurava da samo klijent i TGS mogu doći do tajnog sjedničkog ključa. TGS zatim klijentu šalje natrag primljeni slučajni broj, novi sjednički ključ za komunikaciju sa traženim poslužiteljom i ulaznicu za pristup tom poslužitelju, sve to kriptirano sjedničkim ključem K1.

TGS –>Klijent: M4 = E{(N2, K2, C3), K1}

C3 = {(IDc, IDs, Ts2, Te2, K2), Ks}

Ulaznica identifikatorom kazuje poslužitelju koji klijent želi pristup njegovim uslugama, vremensko ograničenjo u kojem traje ulaznica, sve to kriptirano tajnim ključem poslužitelja, tako da samo dotični poslužitelj može vidjeti podatke u ulaznici.
Klijent provjeri da li je broj N2 dobar, a ključ K2 (sjednički ključ za komunikaciju sa poslužiteljem) spremi.

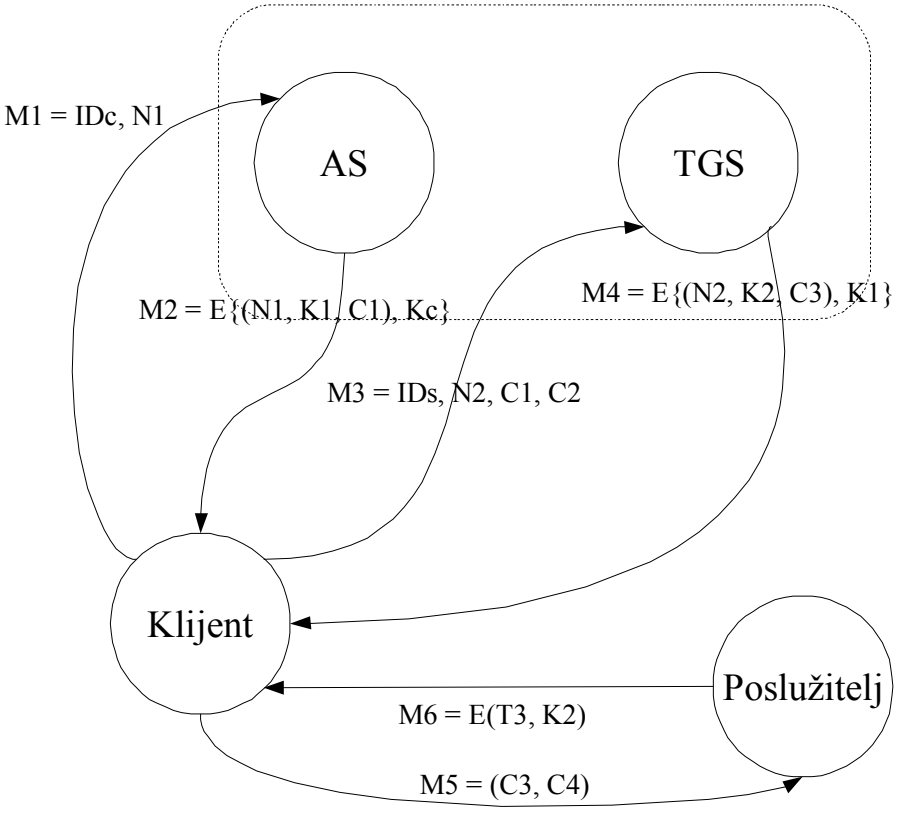
Klijent –> Poslužitelj: M5 = (C3, C4)

C4 = E {(IDc, T2), K2}

Klijent poslužitelju prosljeđuje nepromijenjen sadržaj ulaznice C3 i autentifikator (identifikator + vremenska oznaka) kriptiran sjedničkim ključem K2. Poslužitelj raspakira ulaznicu i autentifikator, provjeri da li identifikatori klijenta odgovaraju i vrati klijentu slijedeću poruku:

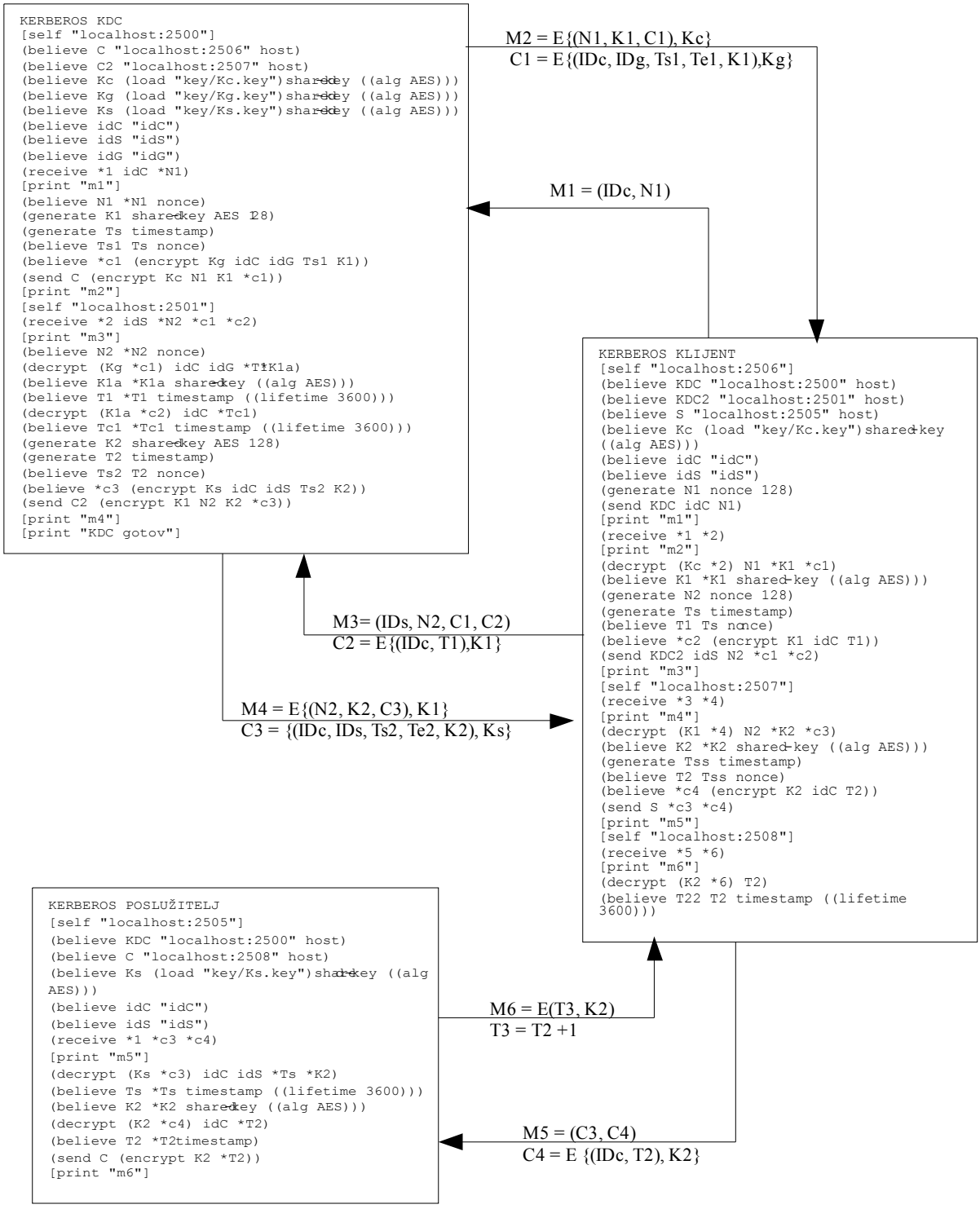
Poslužitelj –> Klijent: M6 = E(T3, K2)

Gdje je T3 = T2 +1. Ako se cijeli postupak provede uspješno, autentifikacija je provedena i klijent sad može nastaviti razmjenjivati poruke sa poslužiteljem.



C1 = E{(IDc, IDg, Ts1, Te1, K1),Kg}
C2 = E{(IDc, T1),K1}
C3 = {(IDc, IDs, Ts2, Te2, K2), Ks}
C4 = E {(IDc, T2), K2}
T3 = T2 +1

Slika 7.1 Kerberos protokol



Slika 7.2 Ostvarenje Kerberosa u Obolu

7.2. Protokol za raspodjelu ključeva u zatvorenom asimetričnom kriptosustavu

Raspodjela ključeva po Needhamu i Schroederu je temelj na kojem je razvijan ovaj protokol. Jedina razlika je u tome što je ovdje, umjesto simetričnog, u uporabi asimetrični kriptosustav te je stoga poznat pod nazivom *Needham-Schroeder Public Key Protocol.*

Centar za raspodjelu javnih ključeva (PKM – *Public Key Manager*) svakom korisniku po prijavi dodjeljuje javni ključ Kei, privatni ključ Kdi i identifikacijsku oznaku IDi. Centar također ima svoj javni (Ke) i privatni (Kd) ključ. Svi korisnici imaju svoje privatne ključeve i javni ključ centra, dok centar ima javne ključeve svih korisnika te svoj privatni ključ. Uz pomoć tih ključeva osigurana je razmjena kriptiranih poruka između centra i pojedinog korisnika. Zadatak ovog protokola je uspostava sigurnosnog kanala i omogućavanje razmjene kriptiranih poruka među korisnicima. Za ispunjenje tog zadatka potrebno je izmjeniti 7 poruka između 2 klijenta i centra za raspodjelu javnih ključeva.

Protokol započinje Ana slanjem zahtjeva, koji sadrži njen i Brankov identifikator, kod zahtjeva i vremensku oznaku, centru za raspodjelu javnih ključeva. Prva poruka je kriptirana javnim ključem centra.

A –> PKM: M1 = E {(Ra, T1, IDa, IDb), Kec}

Centar svojim privatnim ključem otkrije sadržaj poruke i pomoću identifikatora pronađe javne ključeve od Ane i Branka. Ani pošalje Brankov javni ključ, kod zahtjeva i njenu vremensku oznaku, sve to kriptirano Aninim javnim ključem.

PKM –> A: M2= E {(Ra, T1, Keb), Kea}

Ana Brankovim javninm ključem kriptira svoj identifikator i nasumično generirani broj (*nonce*).

A –> B: M3 = E {(IDa, Na), Keb}

Branko zatim centru pošalje svoj zahtjev, analogan Aninoj prvoj poruci, samo s novim kodom zahtjeva i vremenskom oznakom, centru,

B –> PKM: M4 = E {(Rb, T2, IDa, IDB), Kec}

Centar njemu, također na isti način kao i Ani, šalje Anin javni ključ (poruka je kriptirana Brankovim javnim ključem).

PKM –> B: M5 = E {(Rb, T2, Kea), Keb}

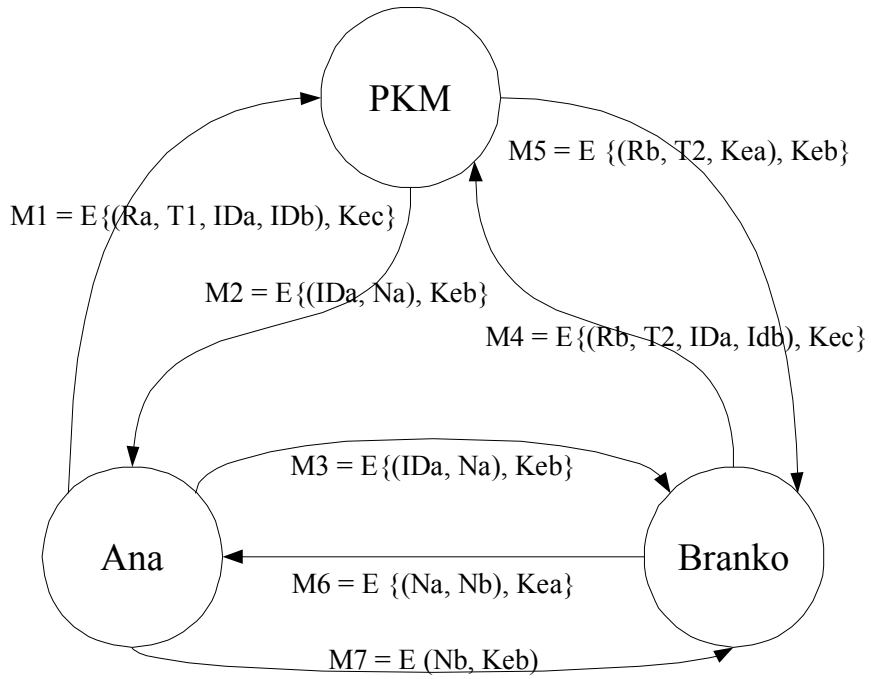
Branko Ani šalje njen *nonce* Na i svoj *nonce* Nb, kriptirano njenim javnim ključem koji je dobio od centra.

B –> A: M6 = E {(Na, Nb), Kea}

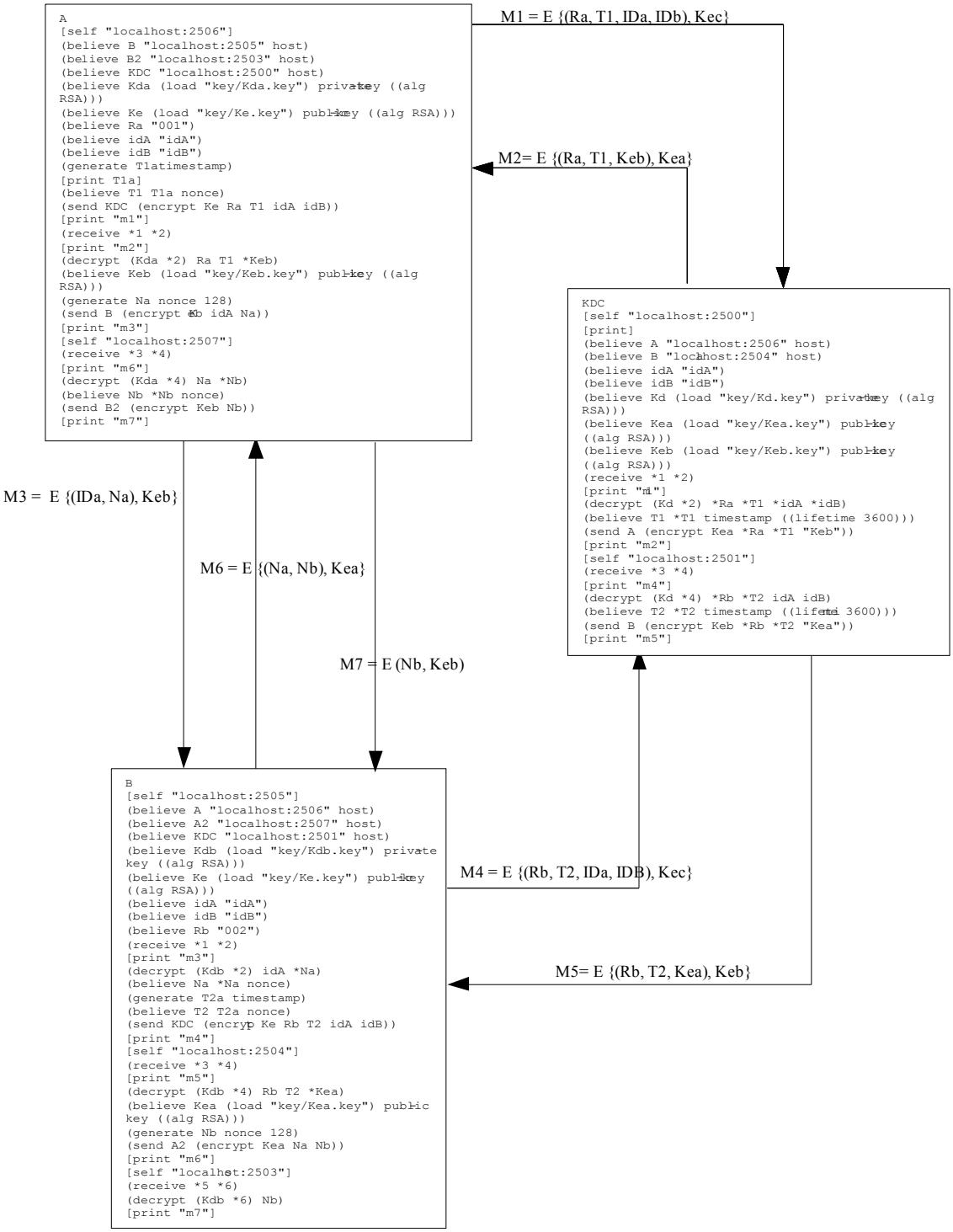
Ana zatim Branku vraća njegov nonce Nb, ovaj put kriptirano Brankovim javnim ključem.

A –> B: M7 = E (Nb, Keb)

Ana i Branko sad mogu sigurno komunicirati pomoću javnih ključeva.



Slika 7.3 Protokol za raspodjelu ključeva u zatvorenom asimetričnom kriptosustavu



Slika 7.4 Protokol za raspodjelu ključeva- ostvarenje u Obolu

7.3. Protokol za obostranu autentifikaciju u zatvorenom asimetričnom kriptosustavu

Pojam autentifikacije obuhvaća identifikaciju i verifikaciju. Protokol za raspodjelu ključeva bi se mogao koristiti i za postupak autentifikacije, međutim nije potrebno jer se korisnici samo moraju predstaviti i dokazati svoj identitet pa se za autentifikaciju koriste jednostavniji protokoli. Kod jednostrane autentifikacije samo se jedan korisnik predstavlja drugom i dokazuje mu svoj identitet. Ipak, za potpunu sigurnost kanala potrebno je dokazati identitet oba korisnika. Takvo dokazivanje identiteta može se napraviti i pomoću dvije provedbe protokola za jednostranu autentifikaciju, što bi zahtijevalo razmjenu 10 poruka, no koristi se poseban protokol za obostranu autentifikaciju koji isti posao napravi u 7 poruka.

Ana prvo šalje autentifikacijskom poslužitelju (AS – *Authentification Server*) zahtjev sa svojim i Brankovim identifikatorom (Ra – kod zahtjeva).

A –> AS: M1 = (IDa, IDb, Ra)

Autentifikacijski poslužitelj (AS) pronalazi Brankov javni ključ šalje ga Ani, skupa s Brankovim identifikatorom (AS kriptira poruku svojim privatnim ključem, čime potvrđuje svoj identitet).

AS –> A: M2= E {(IDb, Keb), Kdc}

Ana Branku šalje poruku kriptiranu njegovim javnim ključem – u poruci se nalazi Anin identifikator i nasumično generirani broj.

A –> B: M3 = E {(IDa, Na), Keb}

Branko zatim šalje poslužitelju svoj i Anin identifikator, Anin slučajni broj i kod zahtjeva, sve to kriptirano tajnim ključem poslužitelja

B –> AS: M4 = E {(IDa, Na, IDb, Rb), Kec}

Poslužitelj vraća Branku poruku od dva dijela.U prvom se nalazi Anin identifikator i javni ključ, dok je u drugom dijelu Brankovim javnim ključem i privatnim ključem poslužitelja kriptiran sjednički ključ, Brankov identifikator i Anin *nonce* Na.

AS –> B: M5 = (C1, C3)

C1 = E {(IDa, Kea), Kd}

C2 = E {(IDb, K, Na), Kd}

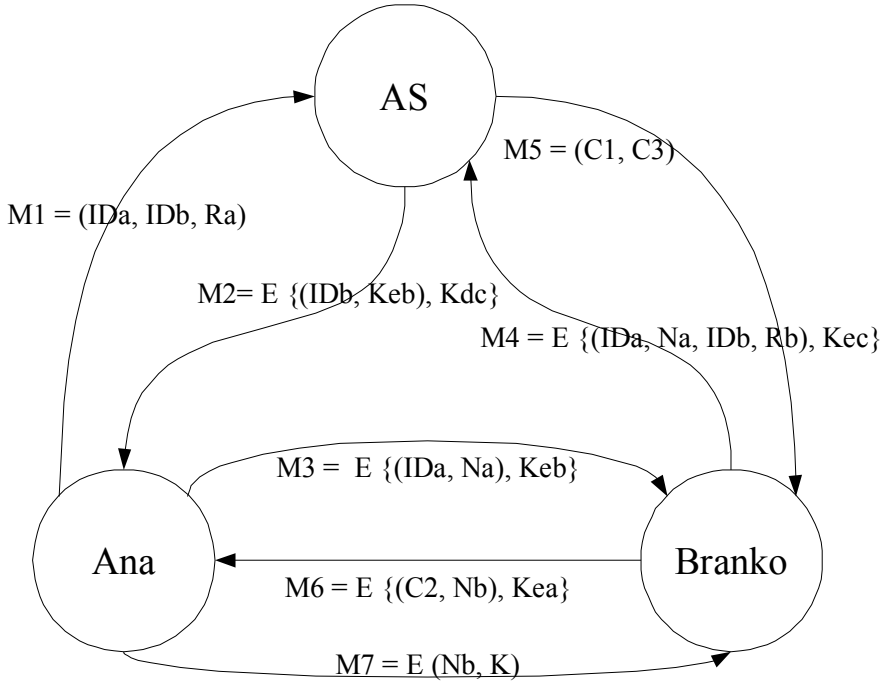
C3 = E (C2, Keb)

Branko Ani prosljeđuje poruku C2 kriptiranu njenim javnim ključem:

B –> A: M6 = E {(C2, Nb), Kea}

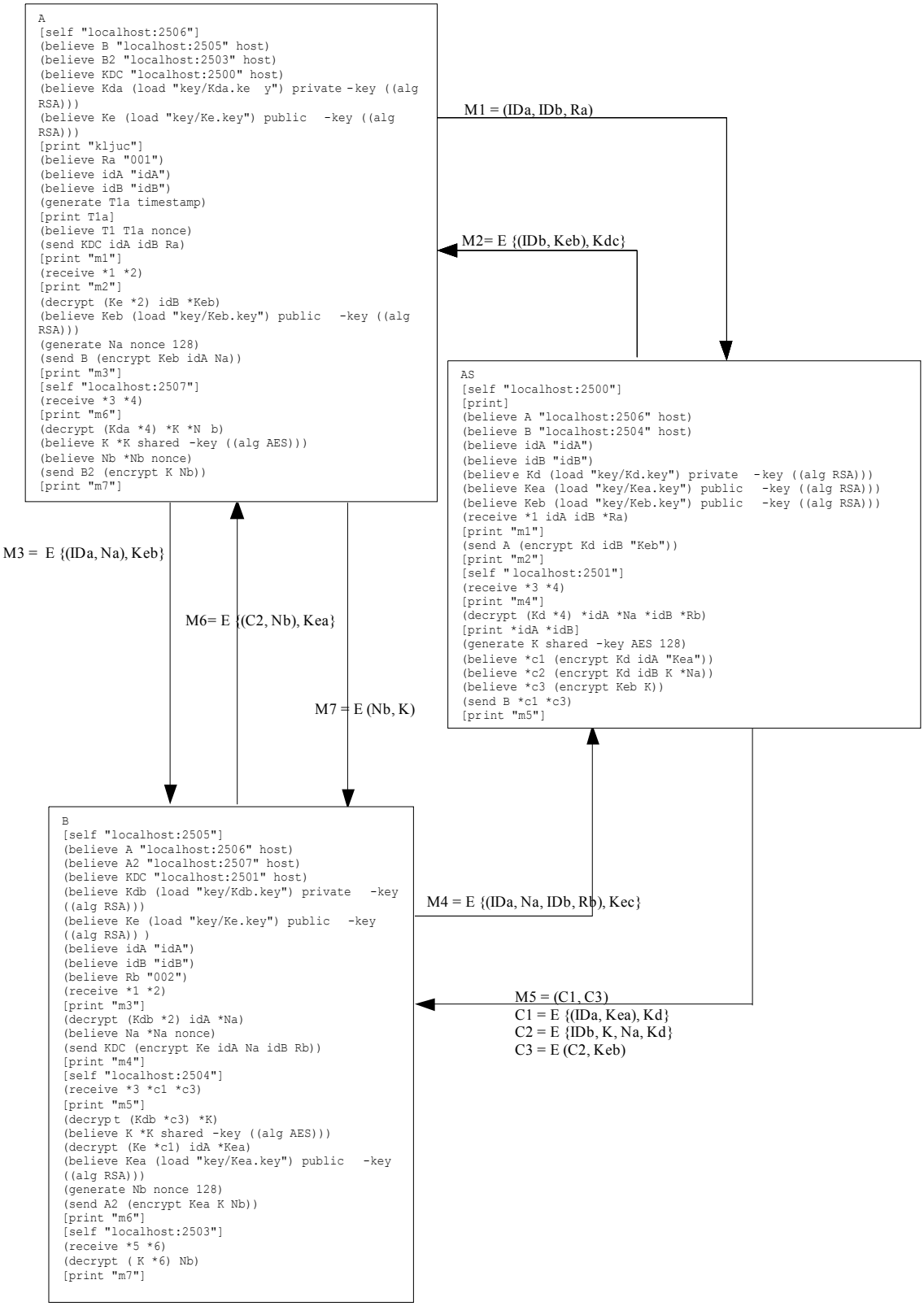
Ana vraća Branku njegov *nonce* Na kriptiran novim sjedničkim ključem i time završava autentifikacija.

A –> B: M7 = E (Nb, K)



C1 = E {(IDa, Kea), Kd}
C2 = E {(IDb, K, Na), Kd}
C3 = E (C2, Keb)

Slika 7.5 Protokol za obostranu autentifikaciju u zatvorenom asimetričnom kriptosustavu



Slika 7.6 Ostvarenje protokola za autentifikaciju u Obolu

7.4. Otway-Rees protokol za autentifikaciju i dodjelu tajnog ključa

Otway-Rees protokol se koristi za autentifikaciju dva korisnika na sigurnoj mreži te dobivanje zajedničkog ključa za buduću komunikaciju. Koristi se simetrični kriptosustav i nakon provedbe protokola korisnici imaju zajednički sjednički ključ za sigurnu komunikaciju, te su međusobno autentificirani. Princip rada je taj da svaki klijent generira svoj "zahtjev za sjednički ključ" koji sadrži njegov vlastiti identifikator i identifikator onoga s kim želi komunicirati, te zahtjev pošalje autentifikacijskom središtu. Neobičnost ovog protokola je u tome što samo jedan korisnik (u ovom slučaju korisnik B) komunicira direktno s autentifikacijskim središtem dok korisnik A koji je inicirao komunikaciju komunicira samo s korisnikom B, te se sva komunikacija korisnika A i AS-a odvija posredstvom kotrisnika B [7].

Protokol je osjetljiv na napade s treće strane – korisnik C može upasti u komunikaciju i urediti da korisnici A i B dobiju različite sjedničke ključeve, što im onemogućava komunikaciju.

Ana šalje Branku zahtjev za početak komunikacije koji se sastoji od njenog i Brankovog identifikatora, koda zahtjeva, i kriptirane poruke (C1) namijenjene poslužitelju.

A –> B: M1 = (R, IDa, IDb, C1)

C1 = E {(Na, R, IDa, IDb), Ka}

Branko prosljeđuje poslužitelju dobivenu kriptiranu poruku. Također mu šalje i svoj identifikator, slučajni broj i kod zahtjeva kriptirane Brankovim tajnim ključem.

B –> AS: M2 = (R, C1, C2)

C2 = E {(Nb, R, IDa, IDb), Kb}

Poslužitelj vraća kod zahtjeva i svakome po jednu "ulaznicu" koja se sastoji od njegova *nonce* broja i nasumično generiranog sjedničkog ključa, sve to kriptirano pojedinim tajnim ključem.

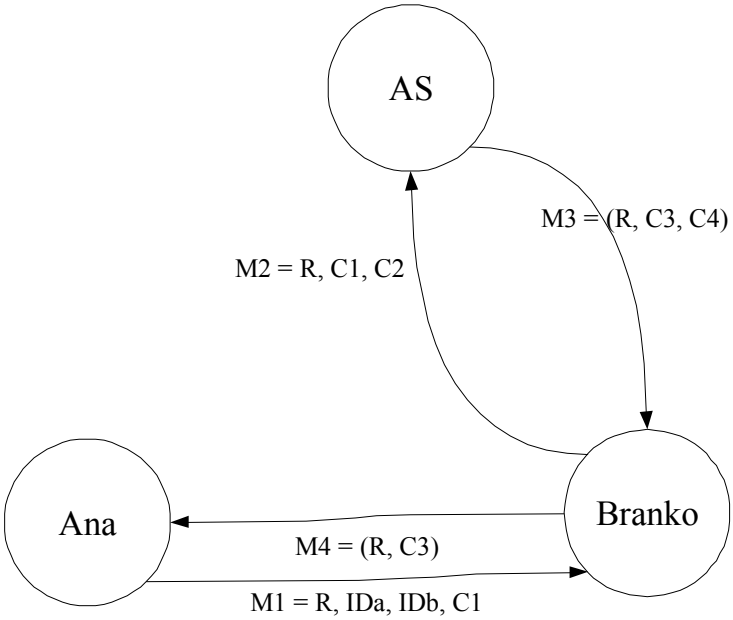
AS –> B: M3 = (R, C3, C4)

C3 = E {(Na, Kab), Ka}

C4 = E {(Nb, Kab), Kb}

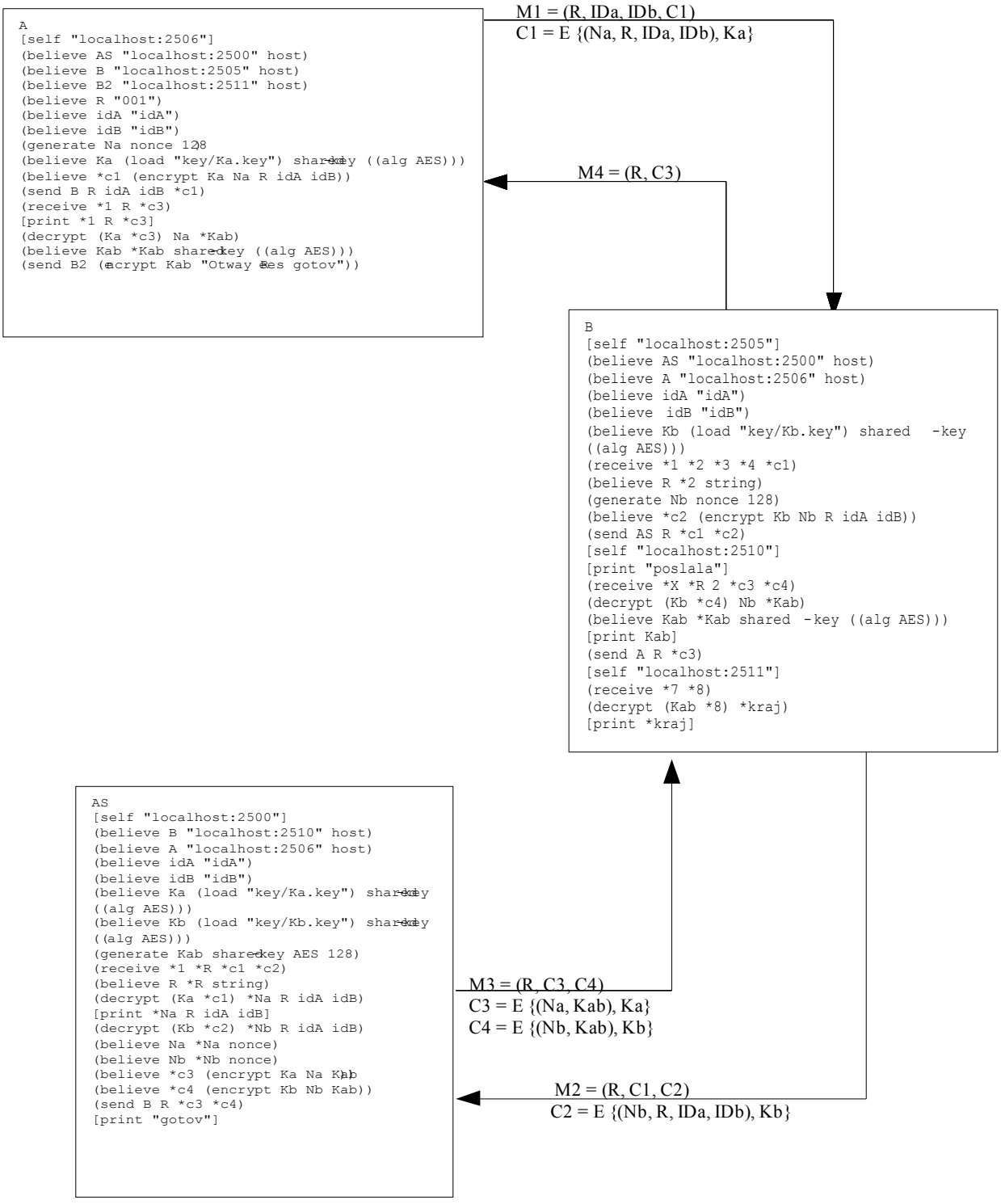
Branko Ani prosljeđuje njenu "ulaznicu" (C3). Branko i Ana mogu komunicirati pomoću sjedničkog ključa Kab.

B –> A: M4 = (R, C3)



C1 = E {(Na, R, IDa, IDb), Ka}
C2 = E {(Nb, R, IDa, IDb), Kb}
C3 = E {(Na, Kab), Ka}
C4 = E {(Nb, Kab), Kb}

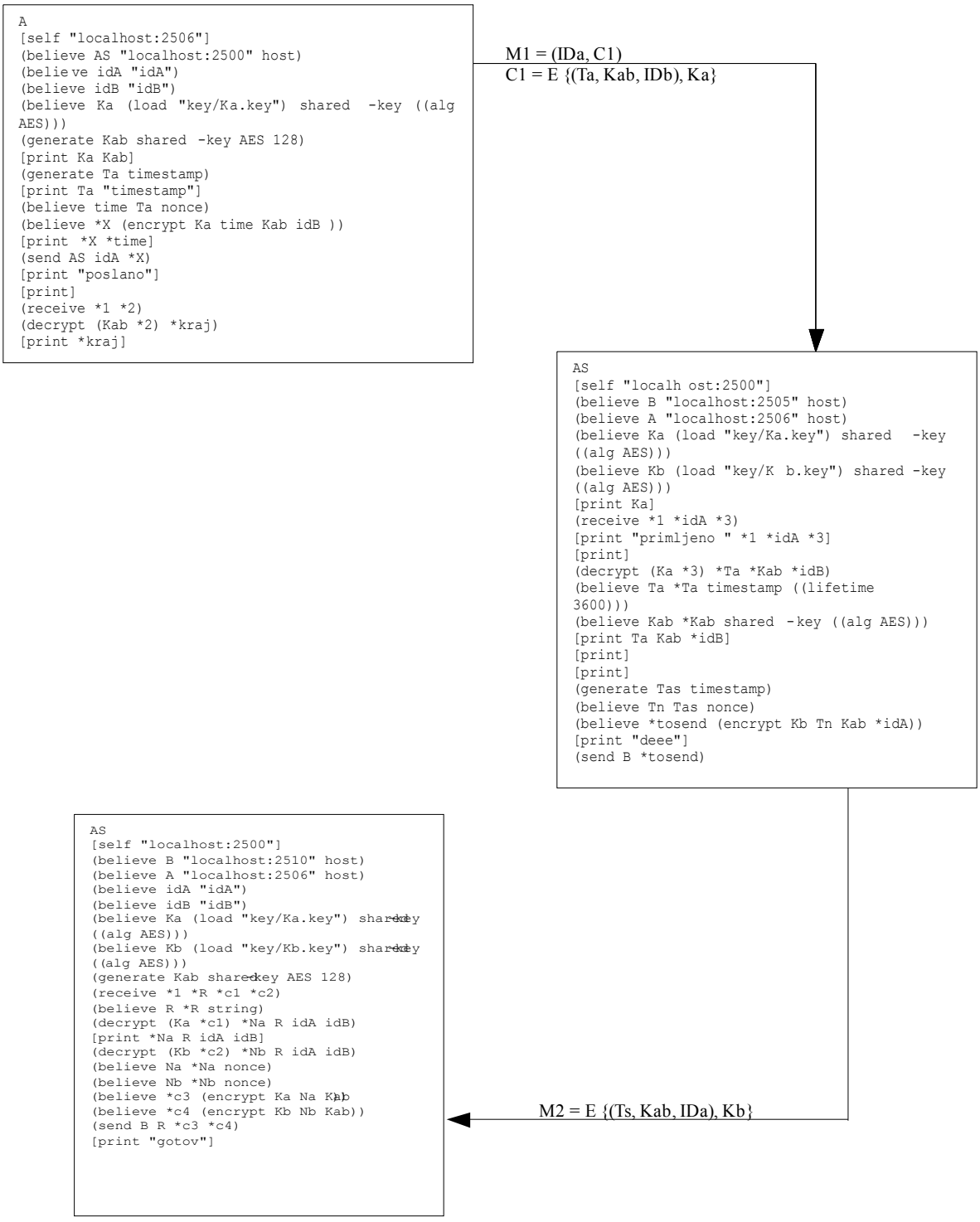
Slika 7.7 Protokol Otway - Rees



Slika 7.8Protokol Otway - Rees ostvaren u Obolu

7.5. Protokol Wide Mouth Frog za autentifikaciju i dodjelu tajnog ključa

Protokol Wide Mouth Frog je opisan u poglavlju 2.4.5.



Slika 7.9 Protokol Wide Mouth Frog ostvaren u Obolu

8. ZAKLJUČAK

U praktičnom dijelu ovog rada u programskom jeziku Obol ostvareno je nekoliko sigurnosnih protokola. Pokazalo se da je Obol zaista intuitivan i jednostavan jezik, koncipiran tako da se standardni zapisi sigurnosnih protokola s malo truda mogu prebaciti u oblik Obol skripte.

Protokoli su, nakon nekoliko dana proučavanja programskog jezika, ostvareni bez većih poteškoća. Međutim, kako su u ovom radu ostvareni sve redom mrežni protokoli njihovo pokretanje i izvođenje veoma je složen proces. Svaki protokol ima najmanje 3 sudionika (računala) koja su simulirana pukim otvaranjem više *Command Prompt* prozora. Vrlo je nezgodan takav način izvođenja jer treba namještati pristupe i ostale parametre kako bi se moglo simulirati slanje i primanje podataka preko mreže, a zahtijeva brzo tipkanje i skakanje iz jednog prozora u drugi. Mogući napredak Obola bi mogao biti u ostvarenju korisničkog sučelja koje bi omogućilo preglednije i učinkovitije istovremeno izvođenje skripti. Također, u ovom radu nije istražena mogućnost ugrađivanja Obola u druge aplikacije (pomoću `input` i `return` metanaredbi), kao i međusobne interakcije više skripti u Obolu, što bi se moglo iskoristiti za buduću izgradnju većih sigurnosnih politika.

9. LITERATURA

[1] Golub M., *Predavanja iz predmeta Operacijski sustavi 2*, Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu, Zagreb, 2005.

[2] Selftest.obol, testna datoteka uz Obol programsko sučelje

[3] Stabell-Kulø T., Segtan Skogan T., Harald Myrvang P., *The design and implementation of Obol*, University of Tromso, 2004.

[4] Na Xu (B.Sc.), An Open and Programmable Security Architecture for GridKIT, Lancaster University, 2005.

[5] Šakić T., *Izrada samostalnog kriptografskog modula*, diplomski rad, FER, Zagreb, 2000.

[6] Zorčec M., *Upravljanje sigurnosnim rizicima*, diplomski rad, FER, Zagreb, 2006.

[7] Antonić A., *Simulacija sigurnosnih protokola*, seminar, FER, Zagreb, 2008.

[8] Vidović V, Ostvarenje Kerberos protokola u različitim programskim okruženjima, diplomski rad, FER, Zagreb, 2006.

[9] Mihalj V., *Sigurna autentifikacija u Unix/Linux okolini*, seminarski rad, FER, Zagreb, 2000.

[10] Paulson L. C., *Relations Between Secrets: Two Formal Analyses of Yahalom protocol*, University of Cambridge, Cambridge, 1997.

[11] Lowe G., *Lowe modified Wide Mouthed Frog*, http://www.lsv.ens-cachan.fr/spore/wideMouthedFrogLowe.html

[12] Harald Myrvang P., *Obol Language Reference*, dodatak doktorskoj disertaciji, University of Tromso

[13] Wikipedia: Kerberos (protocol) http://en.wikipedia.org/wiki/Kerberos_(protocol)

[14] Wikipedia: Abstract Syntax Notation One http://en.wikipedia.org/wiki/ASN.1

[15] Kohl J. T. , Neuman B. C., Ts'o T. Y., *The evolution of the Kerberos authentification service*, MIT, 2003.