SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1646

OVJERA TEMELJENA NA CERTIFIKATIMA U PROTOKOLU IKEv2

Ana Kukec

Zagreb, travanj 2007.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1646

OVJERA TEMELJENA NA CERTIFIKATIMA U PROTOKOLU IKEv2

Ana Kukec

Zagreb, travanj 2007.

Mentor rada: Prof. dr. sc Vlado Glavinić

Sadržaj

1. Uvod

Danas više nije upitno da li je zaštita komunikacije na Internetu potrebna, nego kako je ostvariti. Temelj uspješne zaštite čini prikladno odabrana metoda ovjere koja osigurava međusobnu ovjeru oba sugovornika te zaštitu njihove daljnje komunikacije uspostavom sjedničkog ključa. IPSec je sigurnosna arhitektura i skup protokola koji zadovoljava navedene zahtjeve. Izvedba IPSec-a na trećem sloju OSI modela čini ga vrlo fleksibilnim, budući da štiti sve od prijenosnog sloja pa na više. Također, zaštita pojedinačnih paketa nasuprot zaštite toka okteta čini ga daleko sigurnijim od zaštite na prijenosnom sloju OSI modela (protokol SSL/TLS).

2. Mehanizmi ovjere

U mrežnoj sigurnosti, ovjera je postupak provjere digitalnog identiteta sudionika komunikacije (čovjeka, računala ili aplikacije). Većina mehanizama ovjere je jednostrana (*unilateral authentication*) i omogućava ovjeru korisnika prema jednom ili više poslužitelja. Rjeđi su mehanizmi ovjere koji istovremeno omogućavaju i ovjeru poslužitelja prema korisniku (*mutual authentication*). Najznačajniji i najrjeđi su mehanizmi ovjere koji uz obostranu autentičnost sudionika komunikacije omogućavaju i tajnost komunikacije. Takvi mehanizmi uz ovjeru oba sudionika obavljaju i razmjenu kriptografskog materijala kojim se kriptira komunikacija nakon uspješno obavljene ovjere.

U nastavku poglavlja opisani su različiti mehanizmi ovjere [1], slijedno kako su se razvijali. Svaki novi mehanizam nastao je ispravljanjem nedostataka u prethodnom. Posljednji korak u toj evoluciji predstavlja ovjera temeljena na infrastrukturi javnih ključeva (Public Key Infrastructure, PKI) koja s obzirom na prethodne mehanizme ovjere donosi značajna poboljšanja u sigurnosti, upotrebljivosti i skalabilnosti.

2.1.Lozinke

Lozinka je dijeljena tajna između korisnika i poslužitelja (najčešće operacijskog sustava), a ovjera se svodi na usporedbu primljene lozinke od korisnika i pohranjene lozinke na poslužitelju. Ovjera temeljena na lozinkama je najjednostavniji jednostrani mehanizam ovjere, bilo za korištenje ili implementaciju. Kao takav, on je i najslabiji mehanizam ovjere. Ukoliko napadač otkrije korisničku lozinku, ne postoji način da ga poslužitelj otkrije i spriječi u bilo kojoj od radnji za koju je korisnik autoriziran.

Jedan od problema ovjere temeljene na lozinkama je činjenica da se lozinke od korisnika do poslužitelja putuje kroz nesigurnu mrežu. Korištenje lozinki osobito je nesigurno kod medija koji funkcioniraju na principu razašiljanja tj. unutar lokalne mreže (Ethernet). U tom slučaju napadač vrlo jednostavno može pratiti promet i ukrasti korisničke lozinke korištenjem programa za snifanje. Kod medija koji ne koriste razašiljanje (koncentrator), napad snifanjem je tek nešto kompliciraniji pri čemu napadač program za snifanje mora postaviti na korisničkom računalu.

Drugi problem ovog mehanizma ovjere je skalabilnost. Ukoliko korisnik komunicira s većim brojem poslužitelja moguće je da za sve poslužitelje koristi istu lozinku ili različite. U prvom slučaju, otkrivanjem jedne lozinke dobiva se pristup svim korisničkim računima na različitim poslužiteljima. U drugom slučaju, korisniku je otežano pamćenje velikog broja lozinki.

Nadograđivanje mehanizma kako bi omogućio obostranu ovjeru je moguće, no donosi više problema nego koristi. Korisnik mora pamtiti dvije lozinke (svoju i poslužiteljevu), a poslužitelj mora pamtiti veliki broj korisničkih lozinki. Nedostatak obostrane ovjere dovodi do mogućnosti napada u kojem napadač uspostavlja lažni poslužitelj i na taj način otkriva korisničke lozinke.

Moguća rješenja problema koja će se pojaviti u kasnijim mehanizmima ovjere se odnose na problem snifanja lozinki i omogućavanja sigurne obostrane ovjere. Snifanje lozinki moguće je riješiti kriptiranjem lozinki ili korištenjem lozinke samo jednom.

2.2.Jednom korištene vrijednosti

Mehanizam ovjere u kojem se za svaki pokušaj ovjere koristi drugačija lozinka (*one-time authentication values*) rješava problem snifanja lozinki. Postoje tri osnovna tipa takvih mehanizama:

1. ovjera temeljena na protokolu izazova i odziva (*challenge-response*),
2. ovjera temeljena na protokolu s implicitnim izazovom (*implicit challenge*),
3. ovjera temeljena na sažetku (*hash based authentication*).

Kod protokola izazova i odziva, prije početka ovjere i korisnik i poslužitelj imaju pohranjenu dijeljenu tajnu te u postupku ovjere razmjenjuju tri poruke. U prvoj poruci poslužitelj šalje korisniku slučajno generirani izazov. U drugoj poruci korisnik šalje prema poslužitelju svoj identitet i kriptirani izazov. Nakon primitke takve poruke, poslužitelj pronalazi dijeljenu tajnu koja pripada primljenom identitetu, kriptira izazov tom dijeljenom tajnom i uspoređuje s primljenom lozinkom te šalje korisniku potvrdu o uspješnoj ili neuspješnoj ovjeri. Mehanizam se jednostavno može nadograditi u obostrani mehanizam ovjere bez korištenja nove dijeljene tajne, nego samo razmjenom dodatnog izazova. Ipak, ovaj mehanizam ovjere ne rješava problem skalabilnosti. Korisnik mora za svaki poslužitelj pamtiti dijeljenu tajnu što dovodi i do problema sigurne pohrane lozinki. Ukoliko bi za sve poslužitelje koristio isti ključ, tada bi se neki od poslužitelja mogao predstaviti kao korisnik. Također, postoje jednostavni sustavi koji nemaju sposobnost slanja izazova prema korisniku. U tom je slučaju prikladno korištenje implicitnih izazova.

Primjer mehanizma ovjere koji se temelji na implicitnom izazovu je onaj koji kao izazov koristi vrijeme sustava (*time-based implicit challenge*). Korisnik prema poslužitelju šalje kriptirano vrijeme sustava, a poslužitelj nakon primitka poruke očitava svoje vrijeme sustava i kriptira ga unaprijed podijeljenom tajnom koju ima i korisnik. Budući da je za putovanje poruke do poslužitelja potrebno neko vrijeme i da vremena korisnika i poslužitelja nisu sasvim sinkronizirana, poslužitelj kao ispravnu kriptiranu vrijednost dozvoljava više približnih vrijednosti. Ova karakteristika omogućava izvođenje napada ponavljanjem. Iako napadač ne zna dijeljenu tajnu korisnika, vrlo brzim ponavljanjem poruke koju je korisnik generirao, a u kojoj se nalazi kriptirano vrijeme, poslužitelj može protumačiti napadača kao korisnika koji po drugi puta započinje ovjeru. Dodatno, ovaj mehanizam ovjere je namijenjen jednostavnim sustavima te dvostrana ovjera (iako je teoretski moguća) ne slijedi princip ovog mehanizma koji je prilagođen jednostavnim sustavima kojima predstavljanje eksplicitnog izazova korisniku predstavlja problem.

Rješenje problema koji se pojavljuju u prethodno navedena dva mehanizma ovjere (eksplicitni izazov i odziv te implicitni izazov i odziv) riješen je u mehanizmu temeljenom na funkciji sažimanja (*hash-based authentication*). Primjer takvog mehanizma je S/KEY One-Time Password System. S/KEY ovjera temelji se na nizu jednokratnih lozinki proizvedenih pomoću funkcije sažimanja primijenjene između 500 i 1000 puta. Prva izlazna vrijednost S/KEY algoritma je rezultat funkcije sažimanje koja kao ulaznu vrijednost uzima tajnu korisničku lozinku i javnu sjemenu vrijednost (*seed value*). Takva prva izlazna vrijednost postaje opet ulaz u funkciju sažimanja, itd. Izlaz iz posljednje primjene funkcije sažimanja postaje prva jednokratna lozinka. Prije početka ovjere korisnik šalje prema poslužitelju jednokratnu lozinku, slijedni broj i sjemenu vrijednost. U svakom slijedećem koraku, poslužitelj pamti uspješno verificiranu lozinku iz prethodnog koraka i koristi je kao ulaz u funkciju sažimanja kako bi proizveo slijedeću jednokratnu lozinku. Javna sjemena vrijednost omogućava korisniku da istu lozinku koristi za veći broj poslužitelja. Ukoliko napadač otkrije jednokratnu lozinku, ne postoji način da generira slijedeću lozinku. Da bi je generirao, morao bi primijeniti obrnutu funkciju sažimanja kako bi saznao tajnu korisničku lozinku, što nije moguće. Problem ovog mehanizma

ovjere je taj što ne postoji jednostavan način da podrži obostranu ovjeru. Dodatno, kao i niti jedan od prethodnih, niti ovaj mehanizam ne omogućava tajnost komunikacije nakon uspješne ovjere (generiranje kriptografskog materijala za kriptiranje komunikacije).

2.3.Kerberos

Mehanizam ovjere nazvan po troglavom psu iz grčke mitologije s najpopularnijom implementacijom na MIT-u predstavlja slijedeći korak u evoluciji mehanizama ovjere: temelji se na enkripciji (lozinke ne putuje u obliku čistog teksta) te je otporan na napade uljeza (*eavsdropping*) i napade ponavljanjem (*reply attacks*). Kao i Kerberos, ima tri glavna dijela: korisnik, poslužitelj i ovjerni poslužitelj ili Centar za distribuiranje ključeva (Key Distribution Center, KDC). Kerberos ovjera sastoji se od dva dijela:

- dobavljanje TG ulaznice (Ticket-Granting Ticket) od KDC-a,
- obostrana ovjera klijenta i poslužitelja.

Dobavljanje TG ulaznice obavlja se pomoću dvije poruke. U prvoj poruci se klijent predstavlja KDC-u. Kao odgovor, KDC šalje klijentu TG ulaznicu i sjednički ključ za logiranje:

4. poruka (klijent → KDC): "korisnik"
5. poruka (KDC → klijent): K_a [S_a, TGT: K_k["korisnik", S_a, period-valjanosti]]

Dio parametara Kerberos ovjere koji će biti korišteni u nastavku nalaze se u tablici Parametri Kerberos ovjere (Tablica 1: Parametri Kerberos ovjere). Klijentski ključ (K_a) izvodi se iz korisničke lozinke zbog čega korisnik ne mora pamtiti simetrični kriptografski ključ, već samo svoju lozinku. TG ulaznicu omogućava funkcioniranje KDC-a bez pamćenja i održavanja stanja sjednice za prijavu na sustav za svakog pojedinačnog korisnika. Također, TG ulaznica omogućava KDC-u brz i jednostavan oporavak u slučaju pada sustava, bez ikakvog utjecaja na klijente. Nakon druge poruke, niti korisnički ključ niti lozinka više nisu potrebni – nadalje se koristi sjednički ključ S_a.

Tablica 1: Parametri Kerberos ovjere

K_a	Korisnički simetrični ključ (posjeduju ga korisnik i KDC)
K_b	Poslužiteljski simetrični ključ (posjeduju ga poslužitelj i KDC)
K_{ab}	Sjednički ključ za ovjeru korisnika i poslužitelja (posjeduju ga korisnik, poslužitelj i KDC)
K_k	Ključ KDC-a (posjeduje ga samo KDC)
S_a	Korisnički sjednički ključ (posjeduju ga korisnik i KDC)

Slijedeći korak nakon dobivanja TG ulaznice je dobivanje ulaznice za pristup poslužitelju (*Ticket*). U zahtjevu koji korisnik šalje prema KDC-u nalazi se korisnički identitet, ciljani poslužitelj, TG ulaznica (K_k["korisnik", S_a, period-valjanosti]) i ovjernik (autentifikator) (S_a[vrijeme]) koji služi sinkronizaciji korisničkog i KDC vremena sustava. KDC dekriptira primljenu TG ulaznicu i provjerava period-valjanosti. Ukoliko je TG ulaznica valjana, KDC proizvodi sjednički ključ K_{ab} i oblikuje ulaznicu:

6. poruka (klijent → KDC): “korisnik”, “poslužitelj”, TGT: K_k[“korisnik”, S_a, period-valjanosti], S_a[vrijeme]
7. poruka (KDC → klijent): S_a[poslužitelj, K_{ab}, TICKET: K_b[“korisnik”, K_{ab}, period-valjanosti]]

Pomoću ulaznice primljene od KDC-a korisnik i poslužitelj se ovjeravaju u zadnje dvije poruke Kerberos ovjere. U predzadnjoj poruci korisnik prema poslužitelju šalje ulaznicu (K_b[“korisnik”, K_{ab}, period-valjanosti]) i vrijeme kriptirano sjedničkim ključem K_{ab}. Nakon primitka takve poruke, poslužitelj vjeruje da sjednički ključ K_{ab} posjeduju samo korisnik, poslužitelj i KDC. Ukoliko je primljeno kriptirano vrijeme približno jednako vremenu sustava poslužitelja (za sinkronizaciju se obično koristi SNTP (Simple Network Time Protocol)), ovjera je uspješno obavljena u zadnje dvije poruke:

8. poruka (klijent → poslužitelj): TICKET: K_b[“korisnik”, K_{ab}, period-valjanosti], K_{ab}[vrijeme]
9. poruka (poslužitelj → klijent): K_{ab}[vrijeme+1]

Budući da pojedini korisnik može poslužitelju predstaviti pojedini ovjernik samo jednom, napadač ne može izvesti napad ponavljanjem pete poruke, u kojoj se šalje zahtjev za pristup poslužitelju. Korisnik dekriptira primljenu vrijednost vremena koju je poslužitelj povećao za 1. Ukoliko je ta vrijednost jednaka vremena njegova sustava, poslužitelj se uspješno ovjerio. Nakon uspješne obostrane ovjere, sjednički ključ K_{ab} omogućuje zaštitu daljnje komunikacije po pitanju tajnosti i integriteta.

Problem Kerberos ovjere je ranjivost KDC sustava. Ukoliko napadač otkrije ključ K_k KDC-a, napadač se može predstaviti poslužitelju kao bilo koji od korisnika čije ključeve KDC posjeduje. Nakon uspješnog napada na KDC sustav, potrebno je promijeniti sve korisničke ključeve i ključeve poslužitelja. Dok je mijenjanje ključeva poslužitelja ponešto jednostavnije, mijenjanje korisničkih ključeva je vrlo zahtjevan posao zbog težine lociranja korisnika i dojavljivanja novih ključeva. Problem mobilnosti korisnika (i njihovog kriptografskog materijala) rješava kriptografija javnog ključa.

2.4.Certifikati

Ovjera Kerberos riješila je većinu problema prijašnjih mehanizama ovjere (problem snifanja lozinki, problem napada ponavljanjem, omogućila je uspostavu kriptografskog materijala za kriptiranje daljnje komunikacije nakon uspješne ovjere, omogućila je obostranu ovjeru..)

Ovjera temeljena na certifikatima je slijedeći korak u evoluciji mehanizama ovjere i rješava temeljni problem ovjere Kerberos – ranjivost KDC-a. U kriptografiji, certifikat temeljen na javnom ključu (*public key certificate*, *identity certificate*) je certifikat koji pomoću digitalnog potpisa povezuje javni ključ s vlasnikom certifikata. Sustav čiji sudionici prilikom komunikacije međusobno povjerenje grade na osnovu povjerenja u treće tijelo naziva se infrastruktura javnog ključa (Public Key Infrastructure, PKI). Takvo treće tijelo se u tipičnom PKI-u naziva Certification Authority (CA). Iako ni ovjera temeljena na certifikatima ne rješava sudionike u komunikaciji problema ovisnosti o trećem sudioniku (u ovom slučaju to je Certification Authority, CA), CA ne sudjeluje izravno u protokolu između dva sudionika. Ukoliko CA nije dostupan, sudionici ipak mogu uspješno obaviti ovjeru.

Primjeri protokola koji koriste ovjeru temeljenu na certifikatima su Secure Sockets Layer/Transport Layer Security (SSL/TLS), Internet Key Exchange version 2 (IKEv2), S/MIME, PGP, OpenPGP. Načini na koje pojedini protokole koriste certifikate se ponešto

razlikuju, no osnovni princip je uvijek jednak. U ovom radu će biti objašnjena upotreba certifikata u IKEv2 protokolu.

3. X.509 certifikati

Ovjera temeljena na certifikatima rješava većinu problema preostalih mehanizama ovjere: omogućava obostranu ovjeru, uspostavu kriptografskog materijala za zaštitu daljnje komunikacije, sigurna je od napada snifanjem i ponavljanjem i sl). Ukoliko je ovjera certifikatima temeljena na povjerenju prema trećem tijelu tj. certifikacijskom centru (Certification Authority, CA), ovjera je sigurna od napada posrednikom (*man-in-the-middle* attack) [2].

PKI je sustav koji omogućava sigurno povezivanje javnih ključeva s identitetima vlasnika certifikata, a temeljen je na povjerenju prema certifikacijskom centru. Zasniva se na kriptografiji javnog ključa i digitalnom potpisu. Kriptografija javnog ključa osigurava tajnost podataka, što znači da kriptirane podatke u PKI-u može pročitati samo onaj za koga su podaci i namijenjeni tj. samo onaj tko posjeduje odgovarajući privatni ključ. Digitalni potpis (sažetak poruke potpisan privatnim ključem pošiljatelja) osigurava cjelovitost i autentičnost podataka zbog čega je poslana poruka sigurna od mijenjanja (cjelovitost), a primatelj je siguran da je poruku poslao onaj tko se predstavlja kao pošiljatelj (autentičnost).

PKI odgovara na slijedeća važna pitanja:

- Tko je vlasnik para asimetričnih ključeva?
- Koje vrste poruka/aplikacija pošiljatelj smije štiti javnim ključem?

Odgovor na pitanje tko ima odgovarajući vremenski važeći ključ daju certifikat i lista opozvanih certifikata (Certificate Revocation List, CRL). Informacija o vrstama poruka/aplikacija koje se smiju potpisati javnim ključem zapisana je u certifikacijskoj politici, u certifikatu.

Za navedene probleme PKI pruža skalabilno rješenje – mogućnost komunikacije velikog broja sudionika. Temelj skalabilnosti PKI-a čini certifikacijska staza (*certification path*).

U ovom poglavlju će biti detaljnije objašnjen profil certifikata, ali ne i preostali dijelova PKI-a (npr. PKI arhitekture i sl), budući da to nije potrebno za razumijevanje ostatka ovog rada. IKEv2 ulazi u detalje sadržaja certifikata, kako korisničkih tako i CA certifikata, no ne i u specifičnosti PKI sustava.

Certifikat povezuje javni ključ s vlasnikom certifikata i njegovim privatnim ključem te čini središnji dio PKI-a. Kao što sadrži informacije o identitetu vlasnika certifikata (*Subject*), sadrži i informacije o izdavatelju certifikata (*Issuer*). Nakon stvaranja sažetka (*hash*) certifikata pomoću funkcije sažimanja, izdavač potpisuje stvoreni sažetak svojim privatnim ključem. Takav princip temeljen na povjerenju prema certifikacijskom centru čini temelj PKI-a. Svi sudionici PKI-a vjeruju certifikacijskom centru, znaju njegov javni ključ i na temelju njega obavljaju verifikaciju primljenog certifikata. Digitalnog potpisa jamči autentičnost pošiljatelja, a period valjanosti upisan u certifikatu utvrđuje da li je certifikat još uvijek valjan. Ukoliko je certifikat još uvijek valjan, potrebno je provjeriti i listu opozvanih certifikata. Kako bi se onemogućio napad posrednikom napad, verifikacija certifikata obavlja se preko certifikata izdavatelja (*issuer certificate*, *CA certificate*).

Postoji više tipova digitalnih certifikata (npr. PGP certifikat, SPKI certifikat). U ovom radu će biti opisan široko prihvaćeni format za certifikate temeljene na javnom ključu - X.509 format certifikata. Format pruža veliki broj obaveznih, ali i opcionalnih svojstava te je važno da ih i PKI razvijatelji i korisnici razumiju, kao i da razumiju mogućnosti posljedica i neefikasne

upotrebe certifikata u slučaju neispravnog ili neprikladnog korištenja pojedinih svojstava koje X.509 certifikati pružaju.

Mnogi implementacijski detalji mogu se najbolje protumačiti prateći razvoj X.509 certifikata. X.509 certifikat je nazvan prema dokumentu u kojem je inicijalno bio specificiran – CCITT (danas ITU-T) Recommendation X.509 iz 1988 godine. X.509 preporuka opisuje ovjeru za X.500 protokole. X.509 framework započeo je kao striktno hijerarhijski sustav sastavljen od certifikacijskih tijela (Certification Authorities) čija je osnovna zadaća izdavanje certifikata. Od verzije 3, X.509 podržava uz hijerarhijski model i druge topologije – mostove i potpune mreže (*mesheve).*

Budući da X.500 standard nikada nije bio do kraja implementiran, X.509 se usmjerio na općenite primjene – prema PKI-u i Internetu, a standardizaciju je nastavio IETF. IETF profil X.509 certifikata i X.509 CRL-a za upotrebu na Internetu (za WWW, mail, autentifikaciju i IPsec), zbog čega se X.509 certifikati ponekad nazivaju i PKIX certifikati. Prvi dokument koji je PKIX izdao bio je rfc2459, a zbog premalo kompatibilnosti s ITU-T X.509 standardom, kao i zbog nedovoljne razumljivosti te nespecificiranih segmenata formata, ubrzo je zamijenjen novim standardom – rfc3280 [3]. Jedan od velikih problema rfc2459 bio je vezan upravo uz IPsec. Primjerice, PKIX (Public Key Infrastructure X.509) inicijalno nije prihvatio ekstenzije certifikata koje se tiču primjene certifikata u IPsec aplikacijama kakve je definirao ITU-T. Spomenuti problem samo je jedan od problema koje PKIX radna grupa još uvijek rješava. U prilog važnosti i složenosti PKIX problematike ide i činjenica da je PKIX radna grupa IETF-a aktivna od 1995 do danas što je čini najdulje aktivnom IETF grupom. Danas je aktualna četvrta generacija X.509 certifikata koja još uvijek nosi oznaku verzije 3 (X.509v3) budući da se od treće generacije certifikata razlikuje tek brojem ekstenzija u certifikatu. Aktualna verzija CRL-a je druga.

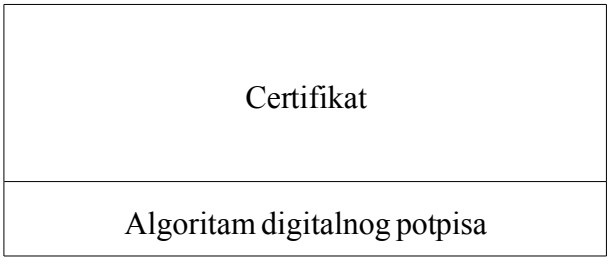
3.1.Struktura X.509 certifikata

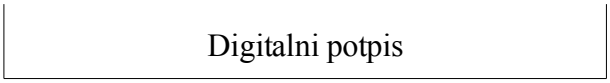
X.509 certifikat se sastoji od tri logičke cjeline [3][4]:

1. Certifikat tj. podaci X.509 certifikata koje je potrebno digitalno potpisati,
2. Identifikator algoritma kojim se izvodi digitalni potpis,
3. Digitalni potpis.

Ključni element samog certifikata tj. podataka certifikata koje je potrebno digitalno potpisati je identitet vlasnika certifikata i privatnog ključa koji pripada javnom ključu iz certifikata. Identitet vlasnika može biti sadržan u X.500 obliku (Distinguished Name) u Subject polju certifikata ili može biti zapisan u subjectAltName polju certifikata (rfc822 e-mail adresa, FQDN i sl). Sadržaj preostale dvije logičke cjeline (identifikator algoritma kojim se izvodi digitalni potpis i sam digitalni potpis) razumljive su same po sebi.

Promatrajući X.509 certifikat na taj način (kao ovojnicu), možemo ga opisati slijedećom tablicom (Slika 1: Tri osnovna dijela X.509 certifikata):





Slika 1: Tri osnovna dijela X.509 certifikata

Certifikat. Dio X.509 certifikata koji se digitalno potpisuje, a detaljnije je objašnjen u nastavku.

Algoritam digitalnog potpisa. Označava algoritam koji izdavač koristi pri stvaranju digitalnog potpisa za certifikat. Primjer je md5WithRSAEncryption što znači da je polje Certifikat ulaz u md5 funkciju sažimanja, čiji se izlaz kriptira RSA algoritmom (privatnim ključem certifikacijskog centra). Budući da se ne nalazi unutar polja Certifikat, polje Algoritam digitalnog potpisa se ne potpisuje digitalno. Zbog toga je kod verifikacije certifikata nužno provjeriti da li je ovo polje identičnom polju unutar samog certifikata (polje Algoritam ID).

Digitalni potpis. Polje koje sadrži digitalni potpis izračunat nad ASN.1 DER kodiranim sadržajem polja Certifikat.

Primjer polja Algoritam digitalnog potpisa i Digitalni potpis iz X.509 certifikata nalaze se u tablici (Ispis 1: Primjer polja Algoritam digitalnog potpisa i Digitalni potpis):

Ispis 1: Primjer polja Algoritam digitalnog potpisa i Digitalni potpis

```
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```

U nastavku teksta će biti detaljnije objašnjeni dijelovi certifikata, a u dodatku se nalazi prikazan X.509 certifikat te opisane ASN.1 građevne jedinice X.509 certifikata.

Verzija (version). Verzija je opcionalno polje koje upućuje na sintaksu certifikata, pri čemu je podrazumijevana postavka sintaksa verzije 1. Svaka sljedeća verzija s obzirom na prethodnu sadrži neka dodatna polja. Verzija 1 ne sadrži jedinstvene identifikatore vlasnika i izdavača i ekstenzije. Verzija 2 sadrži jedinstvene identifikatore, ali ne sadrži ekstenzije, za razliku od verzije 3 koja sadrži i ekstenzije. Svaka implementacija temeljena na [3] minimalno mora podržavati barem verziju 3.

Serijski broj (serialNumber). Serijski broj certifikata je cijeli broj koji je jedinstveni identifikator certifikata unutar nekog certifikacijskog tijela. Dvojac koji se sastoji od identiteta izdavača i serijskog broja iz certifikata jedinstveno globalno identificiraju certifikat.

ID algoritma (algorithmIdentifier). Polje ID algoritma sadrži algoritam digitalnog potpisa (npr md5WithRSAEncryption) i mora biti jednako polju SignatureAlgorithm iz nepotpisanog dijela X.509 certifikata.

Izdavač (Issuer). U polju izdavača zapisan je identitet izdavača prema X.500 standardu, a naziva se Distinguished Name (X.501 type Name, DN). U skladu s [3] DN u polju izdavača ne smije biti prazan string.

Period valjanosti (Validity). Period valjanosti je vremenski period tokom kojeg će CA održavati informacije o statusu certifikata, a određen je s dva elementa: notBefore i notAfter koji mogu biti kodirani kao UTCTime ili GeneralizedTime. Period valjanosti je period koji

započinje s notBefore, a završava s notAfter vremenskim trenutkom. UTCTime i GeneralizedTime obrađeni su u dodatku.

Vlasnik certifikata (Subject). Identitet vlasnika certifikata može biti zapisan u Subject polju ili unutar neke od subjectAltName ekstenzija. Ukoliko je vlasnik certifikata certifikacijski centar, radi se o certifikatu koji će biti korišten za potpisivanje drugih certifikata. Unutar takvog certifikata, Subject polje mora biti popunjeno s DN-om identičnim DN-u u polju izdavača. Struktura DN-a, obradjena je u dodatku.

Javni ključ (Subject Public Key Info). Polje se sastoji od dva dijela: javnog ključa i algoritma s kojim se javni ključ koristi (RSA, DSA ili Diffie-Hellman).

Jedinstveni identifikatori (Unique Identifiers). Polje koje se može pojaviti samo u verzijama 2 i 3, a sadrži jedinstveni identifikator vlasnika (subjectUniqueID) i/ili izdavača (issuerUniqueID) certifikata. Jedinstveni identifikatori su uvedeni kako bi se omogućilo ponovno iskorištenje identiteta vlasnika i/ili izdavača tokom vremena. Iako je prema [3] definirano da svaka PKI implementacija mora biti sposobna obraditi ovo polje, ne preporuča korištenje jedinstvenih identifikatora i ponovno korištenje identiteta vlasnika i/ili izdavača.

Ekstenzije. Objašnjene su u nastavku rada.

3.1.1. X.509v3 ekstenzije

Rane PKI implementacije (verzija 1 i 2 certifikata) pokazale su da osnovna (do sada objašnjena) polja X.509 certifikata nisu dovoljna za efikasnu upotrebu certifikata za različite primjene na Internetu. Iz tog je razloga u certifikat uvedeno polje koje se sastoji od niza ekstenzija. One postoje tek u trećoj u četvrtoj generaciji certifikata (obje generacije čine verziju 3, a razlikuju se brojem ekstenzija).

Ekstenzije sadrže informacije o izdavaču, vlasniku i samom javnom ključu, a mogu se grupirati u nekoliko skupina:

- **Podaci o vlasniku.** Da li je vlasnik certifikata certifikacijski centar (CA) ili krajnji čvor PKI stabla?
- **Podaci o imenima.** Da li su ana.kukec@domena.com i c=Hr; O=FER, cn=Ana Kukec iste osobe?
- **Atributi ključeva.** Za zaštitu kojih aplikacija i podataka se javni ključ smije upotrijebiti?
- **Dodatne informacije.** Gdje se mogu dohvatiti određeni certifikati? Gdje se može dohvatiti neka CRL lista?

Ekstenzije koje odgovaraju na ovakva temeljna pitanja nazivaju se javne ekstenzije. Unutar neke organizacije moguće je definirati i privatne ekstenzije. Budući da ni javne ekstenzije još uvijek nisu općeprihvaćene, nije preporučljivo upotrebljavati privatne ekstenzije jer donose probleme s interoperabilnošću. Uz to, ekstenzije se dijele na kritične i nekritične. Ukoliko je ekstenzija označena kao kritična, primatelj certifikata mora obraditi ekstenziju kako to propisuje standard ili odbaciti certifikat.

Postoji ukupno 14 ekstenzija, od kojih su 4 najčešće korištene na Internetu, no prema [3] samo jedna od njih mora biti nužno označena kao kritična. Svaka od njih sastoji se od identifikatora, zastavice koja pokazuje da li je ekstenzija kritična i od same vrijednosti ekstenzije.

Iako su ekstenzije osmišljene s ciljem kvalitetnije upotrebe certifikata, zbog čestih nedorečenosti u specifikacijama one donose probleme interoperabilnošću. Različiti standardizacijski dokumenti ponekad ih tumače na različite načine, a u implementacije također postoje razlike. Ekstenzije još uvijek nisu globalno prihvaćene. Primjerice, PKIX nikada nije prihvatila extKeyUsage ekstenzije vezane uz IPSec (IPSec End System, IPSec Tunnel, IPSec User).

U nastavku slijedi opis najčešće korištenih ekstenzija unutar IKEv2 protokola pri čemu svaka od njih odgovara na jedno od prethodno navedena četiri pitanja.

[5] definira pravila kako obraditi pojedina polja certifikata (uglavnom ekstenzije) unutar IKEv2 protokola što će biti detaljnije obrađeno u nastavku.

BasicConstraints ekstenzija. Ekstenzija koja spada u skupinu ekstenzija koje sadrže informacije o vlasniku certifikata. Važno je razlikovati dvije vrste certifikata: CA certifikate i klijentske certifikate. CA certifikat je certifikat koji ima dozvolu potpisati klijentski certifikat. Klijentski certifikat (*end entity certificate*) nema takvu sposobnost i krajnji je čvor PKI stabla. CA certifikat može biti potpisan od strane nekog drugog certifikacijskog centra ili može biti samo-potpisan (*self signed, root certificate*). Samo-potpisan certifikat je onaj u kojem je vlasnik certifikata jednak izdavaču (u hijerarhijskom PKI stablu, on je vrh PKI stabla). U slijedećoj tablici (Tablica 2: BasicConstraints ekstenzija u različitim vrstama X.509 certifikata) su prikazane vrijednosti basicConstraints ekstenzije u različitim vrstama X.509 certifikata.

Tablica 2: BasicConstraints ekstenzija u različitim vrstama X.509 certifikata

Samo-potpisan (root) certifikat	BasicConstraints = TRUE
CA certifikat	
Klijentski (end-entity) certifikat	BasicConstraints = FALSE

basicConstraints ekstenzija sadrži i opcionalno polje pathLenConstraint koje definira najveći broj CA certifikata (koji nisu root certifikati) u verifikacijskoj stazi (verifikacijska staza je detaljnije objašnjena u nastavku). Zadnji certifikat (najčešće je to klijentski certifikat) ne ulazi u pathLenConstraint broj CA certifikata. Ukoliko je pathLenConstraint vrijednost postavljena na nulu, znači da se certifikat mora verificirati CA certifikatom koji se u PKI stablu nalazi neposredno iznad samog klijentskog certifikata. Ukoliko opcionalno polje pathLenConstraint ne postoji u basicConstraints opciji, ne postoji ograničenje na duljinu verifikacijske staze.

basicConstraints ekstenzija nije kritična, a PKIX preporučuje označiti je kao kritičnu u CA certifikatima, dok je ne preporučuje u klijentskim certifikatima.

SubjectAltName ekstenzija. Ekstenzija pripada grupi ekstenzija koja sadrže podatke o imenima koja se pojavljuju unutar certifikata. U X.509 verzijama 1 i 2, jedini način pohrane imena bio je X.500 Dinstinguished Name. Da je nekim slučajem X.500 uznapredovao, vjerojatno bi do danas to ostalo tako. Svaki korisnik (bilo osoba, računalo ili aplikacija) imala bi X.500 Directory identitet. No, za razliku od X.500 protokola čiji su razvoj, primjena i implementacija stale, Internet je nastavio razvoj. Budući da X.500 DN imena nisu prikladna za upotrebu na Internetu, pojavila se potreba za alternativnim načinima pohrane imena. Na Internetu je prikladnije pohraniti imena kao što su DNS imena, e-mail adrese, IP adrese, URL-ovi i sl. te je stoga ova ekstenzija osobito prikladna kod klijentskih certifikata. Ukoliko je vlasnik certifikata osoba, prikladno je koristiti primjerice e-mail ili IP adresu. Slično, ako je vlasnik certifikata prilaz ili NAT uređaj iza kojeg se nalazi VPN, prikladno je koristiti URL ili DNS ime.

SubjectAltName ekstenzija je najčešće upotrebljavana ekstenzija kod ovjere temeljene na certifikatima u IKEv2 protokolu. IKEv2 protokol kao korisničke identitete (unutar IKEv2 ID payloada) koristi FQDN, rfc822addr e-mail adrese, IP adrese, X.500 Distinguished Name i X.500 General Name. Kao što je već objašnjeno DN nije prikladan za upotrebu na Internetu, a General Name također nije preporučljiv – previše je liberalan (može pohraniti ime bilo kojeg tipa), a za potrebe unutar IKEv2 protokola zamjenjiv je s FQDN, rfc822addr e-mail adresama te IP adresama koje se mogu definiranati unutar subjectAltName ekstenzija. Uz navedene IKEv2 tipove ID payloada, postoji i tzv. ID_KEY_ID payload koji služi za prijenost identifikacijskih podataka specifičnih za nekog proizvođača. Budući da odgovarajuća subjectAltName ekstenzija koja bi prenijela takav tip payloada ne postoji, a X.500 General Name nije preporučljiv za korištenje prema [8], ID_KEY_ID identifikacijski teret u IKEv2 protokolu se ne koristi ukoliko je ovjera temeljena na certifikatima.

SubjectAltName ekstenzija ne mora biti kritična, osim ako je Subject polje prazno. Tada subjectAltName ekstenzija mora biti popunjena i označena kao kritična.

KeyUsage ekstenzija. X.509 verzija 1 i 2 sadrže javni ključ i identifikator algoritma u kojem će se ključ koristiti, no ne sadrže nikakve druge informacije o ključevima. Većina certifikacijskih centara posjeduje više parova asimetričnih ključeva koje koriste za različite primjene. Ovo je jedna od ekstenzija koja omogućuje razlikovanje takvih parova ključeva. Točnije, ona identificira primjene za koje se pojedini javni ključ može upotrijebiti: keyCertSign, CRLSign, non-Repudation, DigitalSignature, keyEncipherment, dataEncipherment, keyAgreemnet, encipherOnly i decipherOnly.

CrlDistributionsPoint ekstenzija. Postupak verifikacije certifikata započinje pronalaskom nadležnog CA certifikata kako bi se iz njega pročitao javni ključ za verifikaciju digitalnog potpisa u certifikatu. Dodatno, potrebno je dohvatiti najnoviji CRL i provjeriti da li je možda certifikat opozvan. Zato je prikladno da CA kod izdavanja certifikata u njega pohrani ekstenziju koja će sadržavati pokazivač na CRL. Postoji više takvih ekstenzija, a crlDistributionsPoint je najpoznatija i daje informaciju o tome gdje i na koji način dohvatiti odgovarajući CRL. Pripada skupini ekstenzija koje daje dodatne informacije o certifikatu.

Ekstenzija može sadržavati jedan ili više pokazivača na lokacije CRL-ova. Svaki od pokazivača (distribution point) sastoji se od imena lokacija CRL-a (najčešće X.500 Directory ili LDAP directory), zastavice (reasons flag) i izdavača CRL-a.

Za razliku od basicConstraints, keyUsage i subjectAltName ekstenzija koju svaka IKEv2 implementacija mora znati obraditi, crlDistributionsPoint je opcionalna ekstenzija za IKEv2 protokol.

3.2.Certifikacijski centri

Do sada opisivani certifikati sadrže digitalni potpis napravljen na temelju RSA kriptiranog sažetka certifikata javnim ključem certifikacijskog centra. Takvi certifikati koji se izdaju vlasnicima koji nisu certifikacijski centri nazivaju se korisnički ili klijentski certifikati (*end-entity certificates*). Sigurno je da takvi certifikati imaju basicConstraints ekstenziju postavljenu na *false* jer čine završne čvorove PKI stabla.

Certifikati koji su izdani certifikacijskim centrima nazivaju se CA certifikati (*certification authority certificate*). Takvi su certifikati dio verifikacijske staze u procesu verifikacije certifikata i važno je da sadrže sve potrebne informacije za konstrukciju verifikacijske staze i dohvat odgovarajućeg CRL-a.

Osim što certifikati mogu biti izdani drugom certifikacijskom centru ili korisniku, certifikacijski centar može pojedini certifikat izdati sam sebi. U takvom su certifikatu vlasnik i izdavač

certifikata jednaki što znači nemogućnost verifikacije takvog certifikati. Jedini način zaštite samo-potpisanog certifikata je fizička zaštita. Iz tog razloga se root certifikati isporučuju sa softverom kojeg štite (npr. s web preglednicima).

Bilo samo-potpisani ili ne, CA certifikati se dijele u dvije skupine:

- javni CA certifikati koje izdaje javni CA,
- privatni CA certifikati koje izdaje privatni CA.

Javni certifikacijski centri izdaju certifikate bilo kojem korisniku ili CA-u na Internetu. Najpoznatiji javni CA-ovi su VeriSign ili Thawte, a certifikate koje izdaju mogu se kupiti preko Interneta. Pri tome su certifikati grupirani u tri klase ovisno o potrebama korisnika. Klasa 1 su certifikata za zaštitu elektroničke pošte (Class 1 Digital ID) i najjeftniji su certifikati (trenutna cijena je 20 USD za takav VeriSign certifikat). Drugu klasu čine certifikati koji štite kôd (Class 2 Code Signing Certificates), a treću čine certifikati za zaštitu web sjedišta (Class 3 Web Site Certificates).

Privatni certifikacijski centar izdaje certifikate samo korisnicima ili drugim CA-ovima unutar jedne organizacije. Takvom certifikacijskom centru vjeruju samo članovi te organizacije. Primjer upotrebe privatnog certifikacijskog centra je zaštita elektroničke pošte unutar neke organizacije (npr. korištenjem S/MIME-a). Takav CA bi izdao certifikate svakom zaposleniku organizacije, a svi zaposlenik bi konfigurirao svoj klijent za elektroničku poštu tako da prepoznaje certifikat privatnog certifikacijskog centra.

3.2.1. Postavljanje privatnog CA centra

Postoji veliki broj CA alata koji olakšavaju stvaranje privatnog certifikacijskog centra. Najpoznatiji primjer takvog alata je OpenSSL knjižnica (trenutna verzija je 0.9.8). Iako je inicijalno CA funkcionalnost koju pruža OpenSSL knjižnica bila zamišljena kao pokazni primjer, danas ona čini temelj poznatih CA paketa – OpenCA i pyCA, a i koristi je većina malih privatnih certifikacijskih centara unutar jedne organizacije. Ipak, za komercijalne primjene i veći broj korisnika OpenSSL CA funkcionalnost nije dovoljna i zahtijeva nadogradnju (npr. LDAP *storage*). U nastavku je opisan primjer stvaranja privatnog certifikacijskog centra pomoću OpenSSL ca alata.

Postavljanje privatnog certifikacijskog centra sastoji se od četiri koraka:

1. Priprema okoline,
4. Generiranje self-signed (root) CA certifikata,
5. Generiranje zahtjeva za certifikatom (Code Signing Request, CSR),
6. Generiranje klijentskog certifikata,
7. Generiranje CRL-a i opozivanje certifikata.

Priprema okoline za CA. Priprema okoline za OpenSSL certifikacijski centar uključuje stvaranje potrebnih direktorija i datoteka te uređivanje konfiguracijske datoteke za OpenSSL. U tu je svrhu dovoljno koristiti ljusku nekog od operacijskih sustava Unix ili Linux i uređivač teksta.

Prvi korak je odabir direktorija u kojem će se nalaziti CA (*root CA directory*). U ovom primjeru to će biti /opt/privateCA. Unutar njega je prikladno stvoriti dva direktorija – certs (za kopije izdanih klijentskih certifikata) i private (za pohranu privatnog ključa certifikacijskog centra). Povjerenje cijelog certifikacijskog centra zasniva na njegovom privatnom ključu i jedini način njegove zaštite je fizička zaštita. Pristup privatnom ključu certifikacijskog centra smije imati

samo osoba koja izdaje klijente certifikate i CA te se u praksi privatni ključ certifikacijskog centra najčešće pohranjuje na računalu ili u sklopovlju koje nije priključeno na Internet. U našem primjeru, privatni ključ certifikacijskog centra zaštićen je postavljanjem odgovarajućih dozvola na direktorij u kojem se ključ nalazi (direktorij private). Danas je zadovoljavajuća i preporučljiva duljina asimetričnih ključeva 2048 bitova. Preostale datoteke iz /opt/privateCA mogu i zapravo moraju biti dostupne javnosti zbog distribucije certifikata unutar CA (tj. unutar organizacije).

Uz navedene direktorije, potrebno je stvoriti još nekoliko datoteka. Datoteka serial sadrži serijski broj (u heksadecimalnom zapisu i s barem dvije znamenke) certifikata koji se trenutno izdaje ili onog koji će slijedeći biti izdan. Ona osigurava da se ne izdaju dva certifikata s istim serijskim brojem i incijalno se popunjava sa serijskim brojem 1. Datoteka index.txt sadrži popis izdanih certifikata s ispisanim imenom vlasnika i serijskim brojem.

Za stvaranje opisane okoline dovoljne su slijedeće naredbe (Ispis 2: Stvaranje okoline za privatni certifikacijski centar):

Ispis 2: Stvaranje okoline za privatni certifikacijski centar

```
# mkdir /opt/privateCA
# cd /opt/privateCA
# mkdir private certs
# chmod g-rwx, o-rwx private
# echo '01' > serial
# touch index.txt
```

Nadalje, potrebno je urediti i konfiguracijsku datoteku za OpenSSL. Ona sadrži informacije o tome kako će se odvijati slijedeća tri koraka (generiranje CA certifikata, generiranje CSR-a i generiranje klijentskog certifikata). Moguće je koristiti i podrazumijevanu konfiguracijsku datoteku, no u tom slučaju je znatno povećana količina informacija koje je potrebno interaktivno unijeti prilikom već navedena slijedeća tri koraka. Najčešće lokacije podrazumijevane konfiguracijske datoteke su /etc/ssl/openssl.cnf, /usr/local/ssl/lib/openssl.cnf, /usr/share/ssl/openssl.cnf i sl.

Konfiguracijska datoteka (nazovimo je openssl.cnf) sastoji se od dva dijela: ca sekcija i req sekcija. Sukladno tome, postoji openssl ca alat i openssl req alat. Alat openssl ca je minimalna CA aplikacija, a koristi se za potpisivanje CSR zahtjeva te generiranje CRL-ova. Uz to, alat openssl ca uređuje bazu izdanih certifikata i datoteku u kojoj je zapisan status certifikata. Primjer ca sekcije datoteke openssl.cnf (Ispis 3: Sekcija ca iz konfiguracijske datoteke za openssl) je:

Ispis 3: Sekcija ca iz konfiguracijske datoteke za openssl

```
[ ca ]

default_ca    = privateCA

[ privateCA ]
dir           = /opt/privateCA
certificate   = $dir/cacert.pem
database            = $dir/index.txt
```

```
new_certs_dir = $dir/certs
private_key   = $dir/private/cakey.pem
serial        = $dir/serial

default_crl_days    = 7
default_days  = 365
default_md          = md5

policy        = privateCA_policy

x509_extensions     = certificate_extensions

[ privateCA_policy ]
commonName          = supplied
stateOrProvinceName  = supplied
countryName   = supplied
emailAddress  = supplied
organizationName    = supplied

[ certificate_extensions ]
basicConstraints    = CA:false
subjectAltName      = DNS:moja.domena.com
```

Značenje konfiguracijskih parametara su:

- dir: root CA direktorij,
- certificate: staza do self-signed CA certifikata,
- database: datoteka sa sastusnim podacima o izdanim certifikatima,
- new_certs_dir: direktorij s izdanim certifikatima,
- private_key: staza do privatnog ključa certifikacijskog centra,
- serial: datoteka sa serijskim brojem certifikata koji će se slijedeći izdati,
- default_crl_days: međuperiod između generiranja dva CRL-a,
- default_days: vrijeme trajanja izdanog certifikata,
- default_md: funkcija sažimanja za stvaranje sažetka certifikata (polja Certifikat iz X.509 certifikata),
- policy i x509_extensions: specificiraju naziv sekcije u kojoj se nalaze policy odnosno x509_extensions konfiguracijski parametri.

Sekcija policy sadrži konfiguracijske parametre koji definiraju sadržaj Subject polja certifikata tj. definiraju Distinguished Name. Redoslijed konfiguracijskih parametara u određuje i redoslijed pojedinih (istoimenih) dijelova Subject polja certifikata. Vrijednost navedenih

konfiguracijskig parametara može biti: *match* (vrijednost tog konfiguracijskog parametra u CSR zahtjevu i u izdanom certifikatu moraju biti jednaka), *supplied* (CSR zahtjev mora sadržavati istoimeno polje) ili *optional* (CSR zahtjev ne mora sadržavati istoimeno polje). Važno je obratiti pažnju na to da se unesu smislene vrijednosti parametara. Primjerice, countryName mora biti uneseno prema standardu IS03166 za kodove zemalja (dvoslovni identifikatori).

Sekcija x509_extensions sadrži konfiguracijske parametre koji za poslijedicu imaju umetanje ekstenzija u certifikat. Ukoliko sekcija ne postoji, generira se certifikat verzije 1. Za generiranje certifikata verzije 3, sekcija mora postojati pa makar bila i prazna. U gornjem primjeru nalaze se definirane basicConstraints i subjectAltName ekstenzije. Ekstenzija basicConstraints postavljena je na false što znači da izdani certifikat ne može biti CA certifikat, nego čini kraj PKI stabla. Ekstenzija subjectAltName pohranjuje dodatni identitet vlasnika certifikata u obliku FQDN-a.

Drugi dio konfiguracijske datoteke (Ispis 4: Sekcija req iz konfiguracijske datoteke za openssl) je sekcija req koju koristi openssl req alat, a koristi se za stvaranje i obradu CSR-ova u PKCS#10 formatu. Dodatno, koristi se za stvaranje root CA certifikata (uz opciju -x509).

Ispis 4: Sekcija req iz konfiguracijske datoteke za openssl

```
[ req ]

default_bits        = 2048
default_keyfile            = /opt/privateCA/private/cakey.pem
default_md                 = md5

prompt              = no
distinguished_name            = root_ca_distinguished_name

x509_extensions            = root_ca_extensions

[ root_ca_distinguished_name ]
commonName                 = rootCA-common-name
countryName         = HR
emailAddress        = rootCA@zemris
organizationName    = rootCA

[ root_ca_extensions ]
basicConstraints    = CA:true
```

Značenja konfiguracijskih parametara sekcije req su:

- default_bits, default_keyfile i default_md: određuju duljinu privatnog ključa, stazu do dateke s privatnim ključem i funkciju sažimanja za izradu digitalnog potpisa,
- prompt i distinguished_name: definiraju način na koji openssl req alat popunjava Distinguished Name iz certifikata. Ukoliko je prompt postavljen na vrijednost *no*, parametar distinguished_name upućuje na sekciju u kojoj se nalazi podaci za popunjavanje DN polja certifikata.

Preostali konfiguracijski parametri već su objašnjeni kod objašnjavanje parametara iz ca sekcije, samo što se sada primjenjuju na CSR zahtjev i *root* certifikat, a ne na klijentski certifikat kao što je to slučaj s parametrima iz sekcije ca.

Opisane konfiguracijske parametre iz sekcija ca i req koriste alati openssl req i openssl ca. Pri tome se bilo koji od parametara iz konfiguracijske datoteke može pregaziti upotrebom (većinom istoimenih) opcija kod zadavanje naredbe u ljusci operacijskog sustava.

Izdavanje root CA certifikata. U ovom se koraku izdaje *root* certifikat (cacert.pem smješten u /opt/privateCA/cacert.pem) i pripadajući privatni ključ (smješten u /opt/privateCA/cakey.pem) pomoću alata openssl req (Ispis 5: Stvaranje CA certifikata):

Ispis 5: Stvaranje CA certifikata

```
# openssl req -x509 -newkey rsa -out cacert.pem -outform PEM
-config openssl.cnf
```

Nakon unosa naredbe, alat openssl ca traži interaktivno unošenje lozinke (*pass phrase*) kojom se kriptira privatni ključ (podrazumijevani algoritam je DES-EDE3-CBC). Lozinku je potrebno svaki put interaktivno upisati kod izdavanja klijentskog certifikata ili CRL-a. Uz privatni ključ, nastaje i root CA certifikat cacert.pem digitalno potpisan privatnim ključem cakey.pem. I privatni ključ i CA certifikat pohranjeni su u PEM formatu - base64 zapis omeđen s linijama zaglavlja.

OpenSSL omogućuje ispis certifikata i ključeva (ili njihovih dijelova) u čitljivom obliku (Ispis 6: Ispis certifikata u čitljiv obliku (za razliku od base64 zapisa)):

Ispis 6: Ispis certifikata u čitljiv obliku (za razliku od base64 zapisa)

```
# openssl x509 -in cacert.pem -noout -text
# openssl x509 -in cacert.pem -noout dates
# openssl x509 -in cacert.pem -noout purpose
```

Umjesto -config opcije, lokaciju openssl.cnf datoteke moguće je definirati pomoću varijable okoline (Ispis 7: Definiranje lokacije konfiguracijske dattoeke za OpenSSL pomoću varijable okoline):

Ispis 7: Definiranje lokacije konfiguracijske dattoeke za OpenSSL pomoću varijable okoline

```
# OPENSSL_CONF=/opt/privateCA/openssl.cnf
# export OPENSSL_CONF
```

CA certifikati su certifikati na temelju kojih se verificiraju klijentski certifikati. Glavna verifikacijska funkcija openssl-a (x509_cert_verify) kao argumente prima klijentski certifikat (koji je potrebno verificirati) te CA certifikat i CRL (na temelju kojih se klijentski certifikat verificira). Neki čvor na Internetu često vjeruje većem broju certifikacijskih centara i kada prima klijentski certifikat vrlo vjerojatno će pokušati po redu verificirati primljeni certifikat sa CA certifikatima koje posjeduje dok ne pronađe odgovarajući CA certifikat. Zato je važno pripaziti kako i gdje će biti pohranjeni CA certifikati (treba ih razlikovati u odnosu na klijentske certifikate).

CA certifikati mogu biti spremljeni u zasebne datoteke pri čemu PEM format omogućuje spremanje većeg broja certifikata (i ključeva) u istu datoteku. Druga je, i mnogo praktičnija, mogućnost koristiti tzv. standardni CA direktorij. Takav direktorij sadrži veći broj CA

certifikata čije ime je x509 sažetak proizveden slijedećom naredbom (Ispis 8: Stvaranje X.509 sažetka certifikata):

Ispis 8: Stvaranje X.509 sažetka certifikata

```
# openssl x509 -hash -in cacert.pem
```

Primjerice, ako je na taj način proizveden sažetak 1e13c860, u CA direktorij (direktorij proizvoljnog imena) potrebno je smjestiti certifikat s promijenjenim imenom u 1e14c860.0.

Izdavanje zahtjeva za certifikatom. Zahtjev za certifikatom sadrži identifikacijske podatke o korisniku i druge podatke koji certifikacijski centar umeće u sam certifikat koji se izdaje na temelju primljenog CSR-a. Prilikom generiranja CSR-a, generiraju se i privatni i javni ključ korisnika. Javni ključ se umeće u CSR, a privatni ključ ostaje kod korisnika.

Zahtjev za certifikatom može generirati korisnik ili certifikacijski centar u ime korisnika koji je prethodno od certifikacijskog centra zatražio tu uslugu i certifikacijskom centru ostavio svoje identifikacijske i druge podatke koji će biti umetnuti u CSR i certifikat. Ukoliko ga generira korisnik, privatni ključ nikada ne putuje Internetom, što je važna sigurnosna činjenica. Primjerice, kod javnog certifikacijskog centra VeriSign, za jednostavnije certifikate (klasa 1, za zaštitu elektroničke pošte), CSR generira certifikacijski centar. Složenije CSR-ove (za certifikate klase 2 i 3) generira sam korisnik.

U našem primjeru privatnog certifikacijskog centra pretpostavljamo da korisnik generira CSR i zato pri tome ne koristimo konfiguracijsku datoteku openssl.cnf. Konfiguracijsku datoteku koristi samo certifikacijski centar kod izdavanja *root* certifikata (alat openssl req) i kod izdavanja klijentskog certifikata (alat openssl ca), Ispis 9: Stvaranje CSR-a za klijentski certifikat:

Ispis 9: Stvaranje CSR-a za klijentski certifikat

```
# openssl req -newkey rsa:2048 -days 9999 -keyout
clientkey.pem -keyfrom PEM -out req.pem
```

U postupku generiranja CSR-a nastaju dvije datoteke. Privatni ključ klijenta je kriptiran DES-EDE3-CBC algoritmom za koji je ključ tj. lozinka (*pass phrase*) unesena interaktivno nakon upisivanje gornje naredbe, a zatim je spremljen u datoteku clientkey.pem u PEM formatu. Ta lozinka nije od globalnog značaja za certifikacijski centar, ali je važna za korisnika. Ukoliko se otkrije lozinka, narušen je integritet tog specifičnog klijentskog certifikata, dok je u slučaju otkrivanja lozinke kojom je zaštićen privatni ključ certifikata narušen integritet cijelog certifikacijskog centra – svih već izdanih certifikata i onih koji će se tek izdati. CSR zahtjev je spremljen u datoteci req.pem u PEM obliku, a polje Certifikat iz X.509 certifikata u CSR-u je zakriptirano s klijentskim privatnim ključem (clientkey.pem). U CSR se umeće i fraza izazova (*challenge phrase*) koju korisnik unosi interaktivno, a koristi se kod opozivanja certifikata. Primljeni CSR certifikacijski centar dekriptira javnim ključem iz certifikata, čime je osigurana autentičnost CSR-a (ne može ga poslati nitko osim dotičnog korisnika koji jedini posjeduje odgovarajući privatni ključ).

Moguć je i ispis generiranog CSR-a u čitljivom obliku (Ispis 10: Ispis CSR-a):

Ispis 10: Ispis CSR-a

```
# openssl req -in req.pem -text -noout
```

Izdavanje certifikata. U prethodna tri koraka postavljanja privatnog certifikacijskog centra obavljeno je sve što je potrebno prije izdavanja klijentskog certifikata. U prvom koraku je

pripremljena okolina, uključujući i bazu u kojoj se nalaze popis izdanih certifikata te datoteku sa serijskim brojem certifikata koji je potrebno izdati (vrijednost u njoj je 01 jer je tek potrebno izdati prvi certifikat). U drugom koraku je generiran *root* CA certifikat i pripadni privatni ključ certifikacijskog centra. U trećem koraku je generiran CSR zahtjev koji sadrži javni ključ korisnika i identifikacijske podatke korisnike, a kriptiran je privatnim ključem korisnika. Također, u tom je koraku generiran i pripadni privatni ključ korisnika. Slijedeći korak je izdavanje certifikata korisniku. Takav klijentski certifikat sadrži identifikacijske i druge podatke koje je certifikacijski centar primio u CSR zahtjevu. Oni se digitalno potpisuju privatnim ključem certifikacijskog centra.

Klijentski certifikat izdaje CA tj. alat openssl ca na temelju ca sekcije iz konfiguracijske datoteke openssl.cnf (Ispis 11: Stvaranje klijentskog certifikata):

Ispis 11: Stvaranje klijentskog certifikata

```
# openssl ca -in req.pem -in req.pem -config openssl.cnf
```

Alat openssl ca prvo traži interaktivno unošenje lozinke. Ta lozinka je lozinka tj. simetrični ključ kojim je kriptiran privatni ključ CA centra. Zatim se na temelju CSR zahtjeva generira certifikat: popunjava se podacima iz CSR zahtjeva i digitalno potpisuje privatnim ključem CA centra. Najvažniji dio certifikata je javni ključ, a pripadajući privatni klijentski ključ se nalazi već korisnika (korisnik ga je sam generirao prilikom generiranja CSR zahtjeva). Generirani klijentski certifikat nalazi se u direktoriju certs: /certs/01.pem (naziv certifikata je uvijek oblika <serial>.pem).

Ispis certifikata u čitljivom obliku moguće je s (Ispis 12: Ispis klijentskog certifikata):

Ispis 12: Ispis klijentskog certifikata

```
# openssl x509 -in cacert.pem -noout -text
```

Izdavanje CRL-a i opozivanje certifikata. Za generiranje CRL liste (koja sadrži popis opozvanih certifikata) koristi se alat openssl ca. Iako se tokom postupka generiranja CRL-a ne koristi privatni ključ, openssl ca ipak zahtijeva unos lozinke (*pass phrase*) za privatni ključ kako bi se osiguralo da CRL generira ovlaštena osoba (Ispis 13: Stvaranje CRL-a):

Ispis 13: Stvaranje CRL-a

```
# openssl ca -gencrl -out privateCA.crl
# openssl crl -in privateCA.crl -text -noout
```

Kao i CRL, certifikat je javno dostupan. Zato potreba za opozivanjem certifikata predstavlja problem, budući da se nakon izdavanja kopije certifikata nalaze distribuirane na većem broju mjesta koja CA ne može saznati. Potrebu za opozivanjem certifikata potrebno je prijaviti certifikacijskom centru koji onda opoziva certifikat (Ispis 14: Opozivanje certifikata):

Ispis 14: Opozivanje certifikata

```
# cp certs/01.pem revokecert.pem
# openssl ca -revoke revokecert.pem -config openssl.cnf
```

U postupku opozivanja certifikata ne dešavaju se nikakve promjene na certifikatu, nego na statusnoj bazi (index.txt u našem primjeru). Umjesto oznake V (*valid*), certifikat se označava s

R (*revoked*). Također, certifikat se dodaje na CRL listu koju klijenti periodički dohvaćaju i prilikom verifikacije certifikata provjeravaju.

4. Sigurnosna arhitektura za IP

IPSec (Internet Protocol Security) je standard i skup protokola (opcionalan za IPv4, a obavezan za IPv6) koji obuhvaća mehanizme za zaštitu prometa na razini trećeg sloja OSI modela (kriptiranjem i/ili ovjerom IP paketa) te predstavlja sigurnosnu arhitekturu za IP protokol.

Implementacija na razini trećeg sloja znači da IPSec podrazumijeva prvenstveno promjene u jezgri operacijskog sustava, čime su automatski zaštićene sve aplikacije (u IP paketu sve od TCP zaglavlja pa nadalje). Ovo je jedna od temeljnih razlika između IPSec-a i SSL-a tj. između zaštite prometa na razini trećeg i četvrtog sloja. Ipak, u cilju ostvarenja potpune IPSec funkcionalnosti, nije dovoljno mijenjati samo jezgru operacijskog sustava, nego i aplikacije i njihov API. U suprotnom, čak i ako se korisnik ovjerava certifikatom, API aplikaciji može dojaviti jedino IP adresu sugovornika, ne i njegov identitet. Stoga, iako se implementacija IPSec-a u jezgri operacijskog sustava navodi kao osnovna prednost IPSec-a pred primjerice SSL-om, ona nije u potpunosti točna. Druga razlika između IPSec-a i SSL/TLS-a zasniva se na činjenici da IPSec štiti svaki paket pojedinačno, a SSL/TLS štite struju okteta. Svakako je sigurnije štiti svaki pojedinačni paket i to je osnovna prednost IPSec-a pred SSL/TLS-om. Problem zaštite struje okteta leži u činjenici da TCP ne sudjeluje u kriptiranju podataka i i ne može zamijetiti namjerno umetnute lažne oktete u struju okteta. Sve takve oktete TCP će prihvatiti i proslijediti prema SSL-u. SSL će uočiti lažne pakete, no ne postoji način da dojavi TCP-u informaciju o tome. Stoga će TCP sve stvarne nadolazeće oktete odbaciti (zbog jednakih slijednih brojeva).

IPSec osigurava ispunjenje slijedećih sigurnosnih zahtjeva:

- tajnost (*confidentiality*; isključivo ovlaštena osoba može pristupiti podacima),
- integritet ili besprijekornost (*integrity*; nemogućnost promijene podataka od strane neovlaštene osobe),
- autentičnost (*authentication*; verifikacija identiteta pošiljaoca),
- raspoloživost (*availability*; dostupnost podacima unatoč neočekivanim događajima, npr. DOS napad i sl.).

Spomenuti zaštitni mehanizmi IPsec-a zasnivaju se na:

- Sigurnosnim protokolima: *Encapsulating Security Payload* (**ESP**) i *Authentication Header* (**AH**)),
- Specifičnoj arhitekturi: *Security Association* (**SA**) unosi u *Security Association Database* (**SAD**) u jezgru OS-a, *Security Policy* (**SP**) unosi u *Security Policy Database* (**SPD**) u jezgru OS-a te PAD baza (Peer Authorization Database) koja čini povezanicu između daemona za razmjena ključeva i SPD baze.
- Algoritmima za ovjeru i kriptiranje (npr. DES, 3DES, RSA, DH, SHA1, MD5 i dr).
- Protokolima za uspostavu ključeva: Internet Key Exchange (IKE), Internet Key Exchange v2 (**IKEv2**), Kerberized Internet Negotiation of Keys (KINK), Just Fast Keying (JFK) i dr.

Kada sugovornici žele ostvariti sigurnu vezu zaštićenu IPSec-om, njihova IPsec implementacija mora znati odgovor na dva pitanja: što treba zaštiti i kako to zaštiti.

Komunikacija između sugovornika definirana je izvorišnom i odredišnom IP adresom, tipom protokola višeg sloja (promet koji želimo zaštiti), smjerom komunikacije, tipom sigurnosnog protokola (AH ili ESP, opcionalno protokol IPcomp ili protokol encap), načinom rada i dr. Navedeni skup parametara naziva se sigurnosna politika (SP; *Security Policy*) i zapisuje se u jezgru operacijskog sustava. Jednom smjeru komunikacije pripada jedan takav zapis u bazi (SPD; *Security Policy Database*). SP-ovi uvijek dolaze u parovima - za dvosmjernu komunikaciju potrebna su dva SP unosa u SPD bazu.

Opisali smo kako definirati što želimo zaštiti, no kako definirati na koji način to želimo zaštiti? Način na koji određenu komunikaciju želimo zaštiti zapisan je u sigurnosnom udruženju (SA; *Security Association*). SA je također jednosmjerna veza, a pripada joj i identifikator - SPI (*Security Parameters Index*) koji je zapisan u zaglavlju AH ili ESP paketa. Određeno SA udruženje prepoznaje se na temelju SPI-a za taj smjer, na temelju IP adrese i IPsec protokola (AH ili ESP). SA-ovi kao i SP uvijek dolaze u parovima (po jedan za svaki smjer), a zapisani su u jezgri operacijskog sustava, u bazi koja se naziva SAD (*Security Association Database*).

Koncept SA-a temelj je IPsec arhitekture i predstavlja sigurnu poveznicu između sugovornika kojoj pripada dijeljena tajna informacija te skup kriptografskih algoritama koji štite promet koji se tom poveznicom prenosi. Sugovornici će uspješno uspostaviti komunikaciju samo ako se uspiju dogovoriti oko skupa kriptografskih algoritama, ali i drugih parametara kojima će štiti promet koji žele razmijeniti. Kriptografski se algoritmi pri tome dijele na:

- enkripcijske: DES-CBC, 3DES-CBC, CAST-128, RC5, RC5-CBC, RC5-CTS, AES i dr.
- ovjerne: MD5, HMAC, HMAC-MD5-96, HMAC-SHA1-96, HMAC-RIPEMD-160-96, AES-XCBC-MAC-96.

Zamijetimo još samo da je broj SA-ova barem jednak broju SP-ova ili veći. Naime, više SA-ova može pogađati jedno pravilo, a uz to SAD baza je podložna promijenama i u nekom trenutku može postojati više udruženja (SA-ova) između sugovornika nego na početku ostvarivanja komunikacije (jedan od razlog ovome je ponovna uspostava kljuleva (*rekeying)* SA-ova.

Uz SAD bazu koja sadrži parametre pridružene već uspostavljenom SA udruženju (*keyed session*) i SPD bazu koja sadrži sigurnosne politike koje definiraju razmještaj IP prometa ulaznog ili izlaznog za računalo (*host*) ili sigurnosni prilaz (SGW) koji implementiraju IPsec, sadrži i PAD bazu. PAD baza (*Peer Authorization Database*) ima unose koji su poveznica protokola za automatsku razmjenu ključeva i SPD baze. Do izražaja dolazi kod ovjernog dijela protokola za uspostavu ključeva te iako spada u IPSec arhitekturu, za razliku od SAD i SPD baza ne implementira se u jezgri operacijskog sustava – ona je logička baza i implementira se unutar *daemona* za razmjenu ključeva.

Sigurnosni protokoli AH i ESP su protokoli trećeg sloja OSI modela i osiguravaju zaštitu toka IP paketa. AH protokol osigurava besprijekornost i autentičnost IP datagrama. ESP protokol uz spomenutu ovjeru podataka koja je opcionalna, primarno osigurava tajnost IP datagrama. AH ili ESP protokol je zapravo jedan od kriterija "politike" unesene u jezgru operacijskog sustava. Uspješna komunikacija između sugovornika će biti uspostavljena samo onda kada oba sugovornika za tu specifičnu komunikaciju imaju u jezgri postavljeni zahtjev za istim sigurnosnim protokolom: AH ili ESP. Uobičajeno je da se koriste nezavisno, no postoji i mogućnost njihovog istovremenog korištenja. Starija IPSec arhitektura je omogućavala zaštitu sigurnosne veze s oba protokola istovremeno (*SA bundles*), dok novija omogućava isključivo zaštitu jednim sigurnosnim protokolom (na temelju tablice prosljeđivanja i prikladno

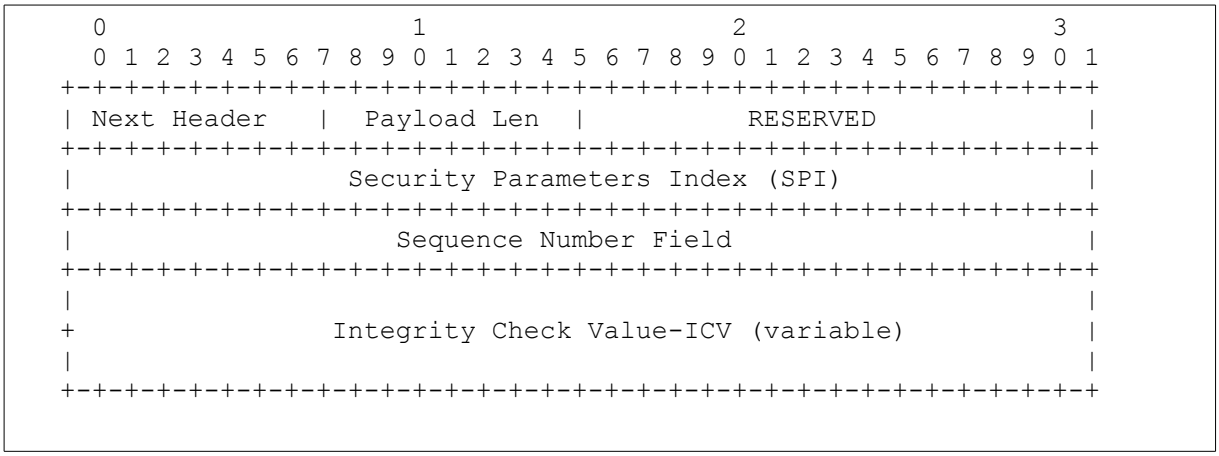
postavljane sigurnosne politike koja uzrokuje po jedan prolazak kroz IPSec granicu za svaki od sigurnosnih protokola).

AH zaglavlje (Ispis 15: IP datagram zaštićen protokol AH) osigurava besprijekornost i autentičnost IP paketa. Dodatno, na temelju klizećeg prozora (*sliding window*) i odbacivanja starih paketa, osigurava zaštitu od napada ponavljanjem paketa (*reply attacks*). Dijelovi AH zaglavlja su:

- *Next Header* - identificira protokol podatkovne jedinice koja slijedi (ovisno o načinu rada (*transport, tunnel*) to može biti protokol prijenosnog sloja ili mrežnog sloja),
- *Payload Length* - veličina AH paketa,
- RESERVED - rezervirano za buduće potrebe (sada je popunjeno nulama),
- *Security Parameters Index* (SPI) - jednoznačno pripada sigurnosnim parametrima veze u jednom smjeru (zajedno s odredišnom adresom i tipom sigurnosnog protokola (AH ili ESP) jednoznačno identificira zaštićenu vezu (*Security Association*) od Alice prema Bobu ili obrnuto),
- *Sequence Number* - redni (monotono rastući) broj paketa koji štiti od napada ponavljanjem,
- *Authentication Data* - svi podaci potrebni za osiguravanje autentičnosti IP paketa.

U *Authentication Data* polju AH zaglavlja nalazi se ICV vrijednost (*Integrity Check Value*).

Ispis 15: IP datagram zaštićen protokol AH



Nad većinom polja IP paketa zaštićenog protokolom AH izvodi se zaštita pomoću ICV vrijednosti (kao rezultat HMAC algoritma). Za razliku od "obične" funkcije sažimanja, ICV vrijednost kao rezultat HMAC algoritma računa se i na temelju podataka koji se štite i na temelju dijeljene tajne. Kao takav, pruža zaštitu besprijekornosti i autentičnosti podataka prema čemu je sličan digitalnom potpisu. Za razliku od digitalnog potpisa, ne osigurava zaštitu od poricanja.

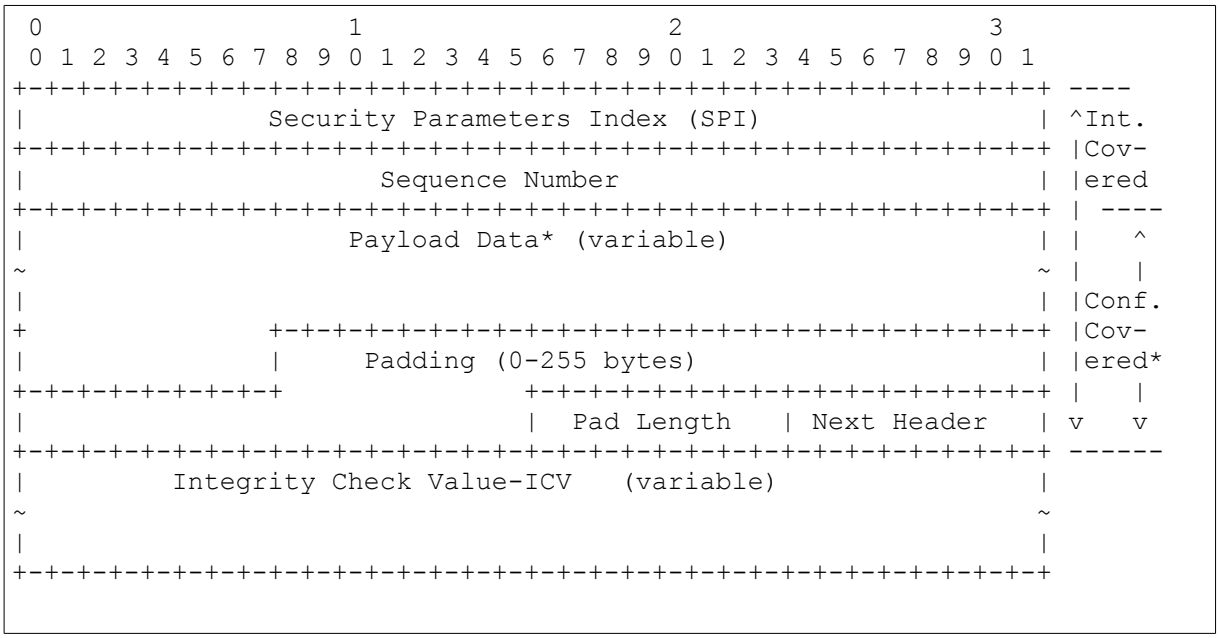
Jasno je, izračunavanje ICV vrijednosti na temelju sadržaja i ključa je mnogo sigurnije nego koristiti "čistu" jednosmjernu sumu tj. izračunati sažetak samo na temelju podataka koji se štite, ali ne i ključa. Budući da se u IPsec-u kao osnova MAC algoritma upotrebljava funkcija sažimanja, MAC se naziva HMAC. Uobičajena terminologija IPsec-a je: zaštita autentičnosti i besprijekornosti na temelju HMAC-MD5 i HMAC-SHA1 algoritama (preostali su opcionalni). Kada Bob primi poruku zaštićenu nekim od tih algoritama obavlja provjeru ICV vrijednosti.

Izračunava HMAC ICV vrijednost nad unaprijed dogovorenim poljima IP paketa dogovorenim autentifikacijskim algoritmom te provjerava da li su dobivena i izračunata vrijednost jednake.

AH štiti gotovo sve dijelove IP paketa, isključeni su samo oni koji se pri svakom skoku kroz mrežu (u usmjernicima) mijenjaju - TTL (*Time To Live*) i suma zaglavlja (*header cheksum*) te još i TOS (*Type Of Service*), *flgs* i *frag offset*. Važno je uočiti da AH štiti i polja s izvorišnom i odredišnom adresom što za posljedicu ima "probleme" kod AH+NAT funkcionalnosti.

ESP (Ispis 16: IP paket zaštićen protokolom ESP) osigurava autentičnost, besprijekornost i tajnost paketa. Na desnoj strani uz zaglavlje obilježeno je područje koje je zaštićeno s obzirom na zahtjev tajnosti (*Conf. Coverage*) i područje koje je zaštićeno s obzirom na zahtjev autentičnosti (*Auth. Coverage*).

Ispis 16: IP paket zaštićen protokolom ESP



Dio koji je kriptiran (dakle, zaštićen po pitanju tajnosti) obuhvaća sam sadržaj (*Payload Data*; *payload* IP paketa je npr. TCP ili UDP segment (segment prijenosnog sloja)), *Padding* i polje koje u sebi nosi tip protokola poruke višeg sloja koja je enkapsulirana u tom IP paketu. Na ovaj način, osim što su podaci kriptirani, napadač čak ni ne zna što je enkapsulirano unutar paketa (jer je polje *Next header* također zakriptirano).

Kriptiranje se obavlja nekim od simetričnih algoritama oko kojih se sugovornici moraju dogovoriti. Moguće je koristiti ESP i bez kriptiranja što je označeno kao ENCR_NULL algoritam. Koristiti ESP s NULL algoritmom ne zadovoljava zahtjev tajnosti (njime se ESP-om, ukoliko ga koristimo u kombinaciji s AH, ostvaruje autentičnost i besprijekornost, ali samo zaglavlja) i iz toga aspekta najčešće nema nikakvog smisla koristiti ga osim u ispitne svrhe.

Odabir simetričnog algoritma potpuno je proizvoljan, važno je jedino da se sugovornici dogovore koje će koristiti. Svojedobno se na IPsec mailing listi razvila diskusija o tome kako RFC-ovi o IPsec arhitekturi [7] čak niti ne spominju da li koristiti u implementaciji primjerice ECB ili CBC što bi moglo izazvati probleme u interoperabilnosti. Odgovor se uvijek podrazumijeva - potrebno je razmisliti što je sigurnije. ECB način kriptiranja ima karakteristiku da bez inicijalizacijskog vektora (IV) algoritam daje ponovljene instance iste poruke kriptirane istim ključem. CBC nema okvakvu karakteristiku i stoga se upotrebljava u implementacijama. Detaljnije informacije mogu se naći u RFC2451: *The ESP CBC-Mode Cipher Algorithms*.

Zaštita s obzirom na zahtjev autentičnosti je opcionalna i sadržana je u polju Authentication Data koje se prema izboru dodaje na kraj zaglavlja. Za razliku od AH protokola, u ovom slučaju, ovjeravaju se samo ESP zaglavlje i kriptirani sadržaj (*Payload*), a ne cijeli IP paket.

Uz dva spomenuta protokola IPsec nudi mogućnost korištenja još dva IP protokola: ipcomp (*IP Payload Compression Protocol*; IP protokol s brojem 108) i encap (*IP encapsulation*; IP protokol s brojem 98). Spomenute protokole nije moguće koristiti zasebno za neku komunikaciju. Primjerice, nije moguće u jezgru unijeti zahtjev samo za kompresijom nekog toka IP paketa. IPcomp za neku komunikaciju može biti zatražen dodatno uz ESP ili AH sigurnosni protokol (prema [8] i [7]).

4.1. Protokol IKEv2

Ulazna vrijednost za kriptografske i sigurnosne algoritme vidljiva korisniku, kao što je već spomenuto, može biti unesena ručno ili pomoću *daemona* za razmjenu ključeva. Primjerice, u slučaju ručne razmjene ključeva i unošenja ključeva pomoću setkey naredbe (iz ipsec-tools paketa KAME projekta), takav ključ je znakovni niz ili heksadecimalni broj. Takva razmjena i unošenje ljučeva donosi mnoge probleme:

- dijeljeni ključ je potrebno na siguran način razmijeniti između sugovornika preko nesigurne mreže,
- za komunikaciju sa svakim udaljenim sugovornikom mora se čuvati drugačiji dijeljeni ključ (veliki broj unaprijed dodijeljenih ključeva),
- postoji mogućnost napada prisluškivanjem i otkrivanja dijeljenog ključa na temelju prikupljenih podataka (nema *perfect forward secrecy* zaštite),
- ne postoji mogućnost autentificiranja certifikatama ili korištenja proširene autentifikacije (EAP),
- SA poveznica se u jezgru unosi ručno s točno definiranim enkripcijskim i autentifikacijskim algoritmom te duljinama ključeva.

Protokol za razmjenu ključeva, iako je opcionalan dio IPSec arhitekture, iz sigurnosnih i praktičnih razloga je neophodan za praktične primjene. Trenutno je u praksi najčešće korišten protokol IKE (verzija 1), no on je zamijenjen novijom verzijom – protokolom IKEv2. Dok do prije dvije godine nije postojala još niti jedna slobodna (*opensource*) implementacija IKEv2 protokola, danas postoje četiri: racoon2 (KAME projekt), strongSwan 4.0 (nasljednik FreeS/WAN projekta) i ikev2 (ZEMRIS).

IPSec implementacija se sastoji od IPSec stoga i *daemona* za uspostavu ključeva kao opcionalnog dijela. Uobičajeno je da je IPSec stog implementiran u jezgri operacijskog sustava, dok je *daemon* u korisničkom prostoru. Prva slobodna implementacija IPSec-a za Linux napravljena je unutar FreeS/WAN projekta, a sadržavala je IPSec stog KLIPS i *daemon* nazvan pluto. Jednu od najpoznatijih IPSec implementacija (posebice po *daemonu* za razmjenu ključeva) napravio je KAME projekt. Njihov *daemon* naziva se racoon, a implementacija je izvorno napravljena za FreeBSD i NetBSD. OpenBSD je napravio vlastitu implementaciju ISAKMP/IKE *daemona* i nazvao ga je isakmpd. Pluto, isakmpd i racoon su implementacije verzije 1 protokola. Prva verzija protokola bila je opisana u nekoliko rfc-ova (2407, 2408, 2409), a predstavljala je protokol koji obavlja obostranu autentifikaciju sugovornika i uspostavlja sigurnosno udruženje (Security Associations, SAs). Dodatno, ima i druge

karakteristike – NAT traversal, različite vrste ovjere (certifikati, EAP, dijeljena tajna), dodjeljivanje adrese udaljenom čvoru (*road warrior* scenarij) koji nisu bili specificirani u navedena tri dokumenta. IKEv2 protokol započet je s nekoliko ciljeva:

- specifirati IKE protokol unutar jednog dokumenta,
- zadržati sve dobre funkcionalnosti protokola,
- pojednostaviti protokol,
- riješiti uočene probleme protokola.

Nažalost, svaki od navedenih problema u IKEv2 je rješen tek djelomično. Standardizacijski postupak IKEv2 protokola i nove IPSec arhitekture tekao je velikim dijelom istovremeno, no uz premalo suradnje što je dovelo do propusta u specifikaciji. Dodatni propusti nastali su zbog toga što je radna grupa prilikom specifikacije zanemarila dodatne specifikacije dokumente IKEv1 (uzimala je u obzir samo 2407, 2408 i 2409), a dok su preostali (NAT traversal, EAP, Configuration Payloadi) uključeni naknadno i gotovo doslovno prepisani iz starog protokola. Iako je ideja bila opisati protokol u jednom dokumentu, ovi propusti doveli su do potrebe stvaranja novih dokumenata. Također, iako je protokol vrlo pojednostavljen, pokazalo se da je mogao biti još jednostavniji – upravo zbog ovih dodatnih dijelova protokola.

IKEv2 je zadržao dobre karakteristike originalnog IKE protokola (skrivanje identiteta, *perfect forward secrecy*, dogovoranje kriptografskih protokola), a unatoč svim propustima ipak je efikasniji, sigurniji, robusniji i fleksibilniji s obzirom na verziju 1 protokola. IKEv2 protokol zahtijeva IPSec arhitekturu opisanu u [7] što predstavlja jedan od velikih problema prilikom implementacije, budući da je u postojećim jezgrama operacijskih sustava još uvijek implementacije stare verzije IPSec arhitekture (prema rfc2401). To ima za posljedicu nemogućnost implementacije nekih dijelova IKEv2 protokola. Iako je implementacija ikev2 *daemona* na kojoj se temelju ovaj rad napravljena na staroj verziji IPSec arhitekture, u radu se objašnjava IKEv2 protokol vezan uz noviju verziju IPSec arhitekture. Također, uz dva osnovna dokumenta koja sadrže opis IPSec arhitekture [7] i IKEv2 protokola [8], tekst i implementacija temelje se i na [5] i [6].

4.1.1. Scenariji primjene protokola

Dva osnovna IPsec načina rada su *transport* i *tunnel*. Jednako kao i sigurnosni protokoli (AH ili ESP) i način rada je parametar politike koji se unosi o jezgru operacijskog sustava (Security Policy), a sugovornici se prije komunikacije moraju dogovoriti oko korištenja istog načina rada (uspješna komunikacija će se upostaviti samo ako oba sugovornika imaju u jezgri unesen isti način rada).

Prvi spomenuti način rada (*transport)* podrazumijeva zaštitu s kraja na kraj (*end-to-end*). Ukoliko sugovornici žele na ovaj način ostvariti sigurnu vezu, tada oba sugovornika moraju imati IPsec implementaciju. Podatkovna jedinica koju razmjenjuju enkapsulira samo IP teret. Posljedica ove činjenice je ta da u zaštićenom paketu u ovom načinu rada postoji samo jedna izvorišna i odredišna adresa IP adresa budući da se između sugovornika ne stvara sigurni tunel (ovo je jedna od osnovnih razlika između *transport* i *tunnel* načina rada). U AH *transport* načinu rada originalni je IP paket tek neznatno promijenjen - između originalnog IP zaglavlja i tereta IP paketa (transportnog segmenta) umetnuto je AH zaglavlje. Jasno je, u originalnom zaglavlju polje *Next header* više ne pokazuje na protokol transportnog sloja nego na AH protokol, dok polje *Next header* u AH zaglavlju sada pokazuje na prijenosni protokol. Na slici je vidljivo da je ovjeren i zaštićen po pitanju integriteta gotovo cijeli IP paket uključujući izvorišnu i odredišnu adresu. Jasno je da zbog toga AH ovjera ne može funkcionirati ukoliko se na putu IP paketa od Alice prema Bobu događa prepisivanje adresa (NAT; *Network Address*

Translation) - ako neki mrežni uređaj prepiše ili izvorišnu ili odredišnu adresu, zaštitna suma više neće biti jednaka.

Kod ESP *transport* načina rada također se između originalnog IP paketa i IP tereta dodaje ESP zaglavlje, a IP teret se kriptira. Opcionalno se na kraj IP paketa dodaju ovjerni podaci. Izvorišna i odrediša adresa se ne nalaze pod zaštitnom sumom.

Drugi način rada, *tunnel* način podrazumijeva *net-to-net* komunikaciju (dakle, za računala iza prilaza) što znači da se mora obaviti enkapsulacija cijelog IP paketa. Kako bi se formirao IPsec tunel, u zaglavlju enkapsuliranog IP paketa (*inner header*) postoji jedan par izvorišna - odredišna IP adresa, a u vanjskom IP zaglavlju (*outer header*) drugi par. U ovom slučaju se sugovornici nalaze iza prilaza između kojih se ostvaruje siguran tunel - sugovornici nemaju implementiran IPsec, a prilazi su ti koji imaju implementiran IPSec. Uobičajeno je takve prilaze u IPsec terminologiji nazivati sigurnosnim prilazima (SGW; *Security Gateways*). Enkapsulacija cijelog IP paketa u ovom načinu rada znači da, primjerice kod AH protokola, nakon originalnog IP zaglavlja (*outer header*) i AH zaglavlja slijedi cijeli IP paket (unutarnje IP zaglavlje (*inner header*) te prijenosni segment), a ne samo prijenosni segment kao što je to slučaj kod *transport* načina rada. Jedina polja koja nisu zaštićena nalaze se u vanjskom zaglavlju (TTL, ToS, *flgs*, *frag offset*, *header cheksum* i naravno *Authentication Data* koji je rezultat AH protokola). Analogna je ovome i razlika između ESP *transport* i ESP *tunnel* podatkovne jedinice trećeg sloja OSI modela - kriptiran nije samo prijenosni segment nego cijeli IP paket.

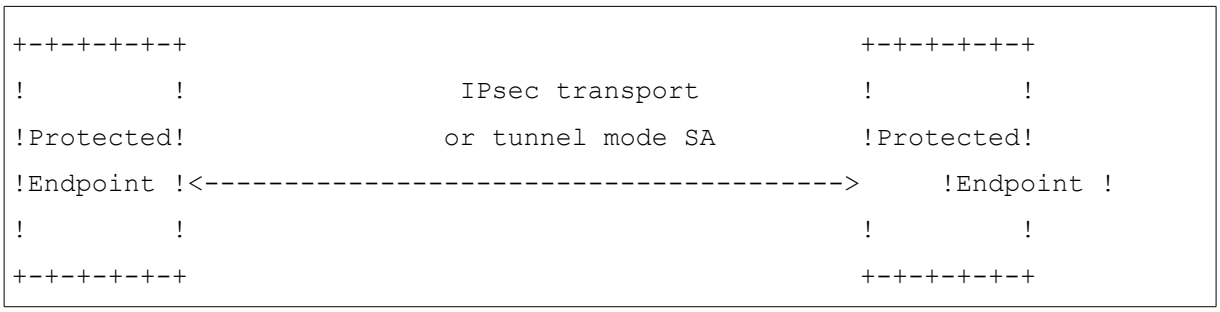
Upotreba jednog od dva spomenuta načina rada ovisi o topologiji mreže i zahtjevima korisnika. Postoji nekolicina IKEv2 scenarija (ovisno o topologiji mreže tj. dijela mreže koja se nalazi između sugovornika) na temelju kojih se odlučuje koji način rada odabrati.

Može se raditi o komunikaciji između:

1. krajnjih korisnika (s kraja na kraj) - *transport* ili *tunnel* način rada,
2. dva sigurnosna prilaza (*Security Gateways*) - *tunnel* način rada,
3. krajnjeg računala i sigurnosnog prilaza - *tunnel* način rada.

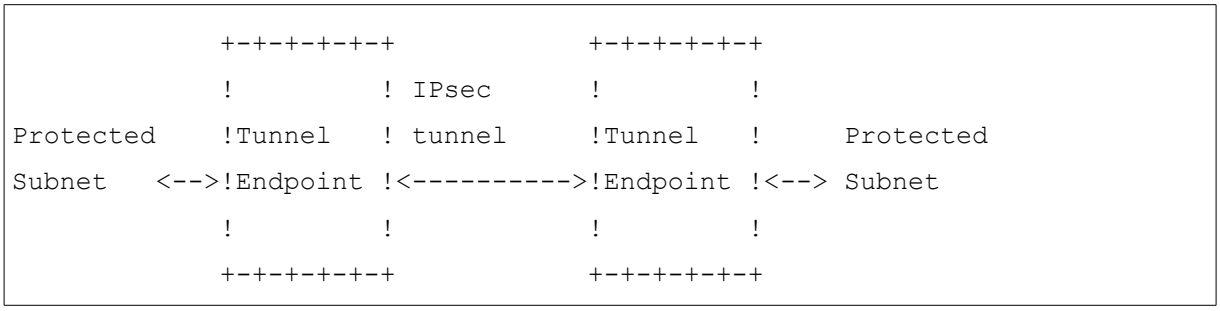
U prvom je slučaju (Ispis 17: Scenarij s kraja na kraj (end-to-end)) uobičajeno upotrebljavati već opisani *transport* način rada. No, moguće je upotrijebiti i *tunnel* način rada. Ukoliko se odabere *tunnel* način rada, izvorišna i odredišna adresa i u vanjskom i u unutrašnjem zaglavlju (ukoliko postoje oba zaglavlja) su jednake.

Ispis 17: Scenarij s kraja na kraj (end-to-end)



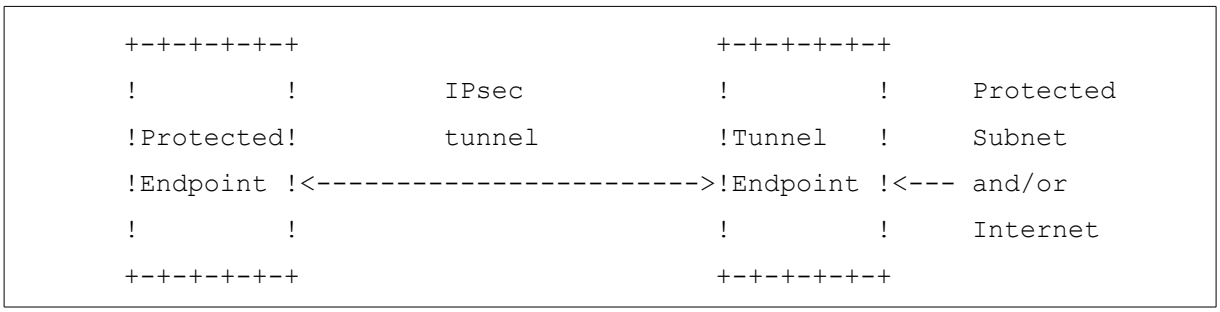
U drugom slučaju (Ispis 18: Scenarij komunikacije između dva SGW-a (net-to-net)), kod komunikacije između dva sigurnosna prilaza, nužno je upotrijebiti *tunnel* način rada - u svrhu stvaranja tunela kroz nesigurnu mrežu, u vanjskom zaglavlju postoji jedan par izvorišne i odredišne adrese, a u unutarnjom zaglavlju drugi par. Sigurnosni prilazi implementiraju IPsec i iza njih se nalazi zaštićena mreža računala koja nemaju implementiran IPsec. Unutrašnja adresa je ta koja pripada krajnjem korisniku.

Ispis 18: Scenarij komunikacije između dva SGW-a (net-to-net)



Treći način rada (Ispis 19: Scenarij komunikacije klijenta i SGW-a (end-to-net)) naziva se *host-to-net*, a najčešći je primjer *road-warrior* način rada. *Road-warriori* su klijenti koji mijenjaju IP adrese (primjerice, prijenosna računala koja se preko nesigurne mreže prijavljuju u matičnu tvrtku). Između klijenta i SGW-a ostvaruje se IPsec tunel (u oba smjera) - radi se o *tunnel* načinu rada. Kako bi *road-warrior* klijent i SGW ostvarili tunel, SGW mora klijentu dodijeliti IP adresu (bilo statički, bilo dinamički). Neka je *road-warrior* klijent koji ima svoju trenutnu adresu i još jednu, adresu koju dobiva od SGW-a. Sugovornik roadwarrior klijenta se nalazi u zaštićenoj mreži iza SGW-a i za razliku od roadwarriora i SGW-a nema implementiran IPsec. Ukoliko se *roadwarrior* i SGW dogovore oko korištenja istog sigurnosnog protokola (AH ili ESP) te istog načina rada (*tunnel*) i drugih parametara politike koje imaju unesene u jezgru, uspješno se ostvaruje komunikacija. SGW će roadwarrioru dodijeliti IP adresu iz mreže u kojoj se nalazi Bob i ta se adresa upisuje na mjesto vanjske izvorišne adrese. U unutarnjem zaglavlju na mjestu izvorišne adrese ostaje zapisana trenutna IP adresa od roadwarriora. U ovakvom scenariju, *roadwarrior* klijent je u politici u jezgri SGW-a predstavljen s IP adresom 0.0.0.0. Kada u SGW pristigne paket od klijenta s nepoznatom adresom, smatra se da je njegova adresa 0.0.0.0 i poštuje se pripadno pravilo iz jezgre. Ako se paket uspješno autentificira, SGW čita iz paketa trenutnu adresu klijenta te na temelju preostalih potrebnih parametara iz paketa i pročitane trenutne adrese klijenta u jezgru unosi pravilo za komunikaciju s tim *roadwarrior* klijentom. Uobičajeno je reći da SGW u ovakvom scenariju mora imati uključenu opciju generiranja politike (*generate policy on*).

Ispis 19: Scenarij komunikacije klijenta i SGW-a (end-to-net)



Iz perpsepektive ovjere, *end-to-net* način rada je interesantan zbog česte primjene EAP autentifikacije klijenta i obavezene RSA autentifikacije SGW-a.

4.1.2. Uloga protokola IKEv2

IKEv2 protokol obavlja:

- obostranu ovjeru sugovornika,
- uspostavu sigurnosnih udruženja (*Security Associations*).

Ovjera se (za svaki smjer posebno) odvija kroz potvrdu identiteta sugovornika primljenog u IKEv2 poruci. IKEv2 podržava dva osnovna tipa autentifikacije:

- PSK: *pre-shared key* - na temelju dijeljenog ključa i HMAC algoritma te prf funkcije (*pseudo random function*),
- RSA: RSA-potpisan PKCS-nadopunjen MAC (*Message Autentification Code*),
- DSS (*Digital Signature Standard*): DSS-potpisan sažetak,
- ECDSS (*Elliptic Curve Cryptography Digital Signature Standard*): ECC baziran DSS.

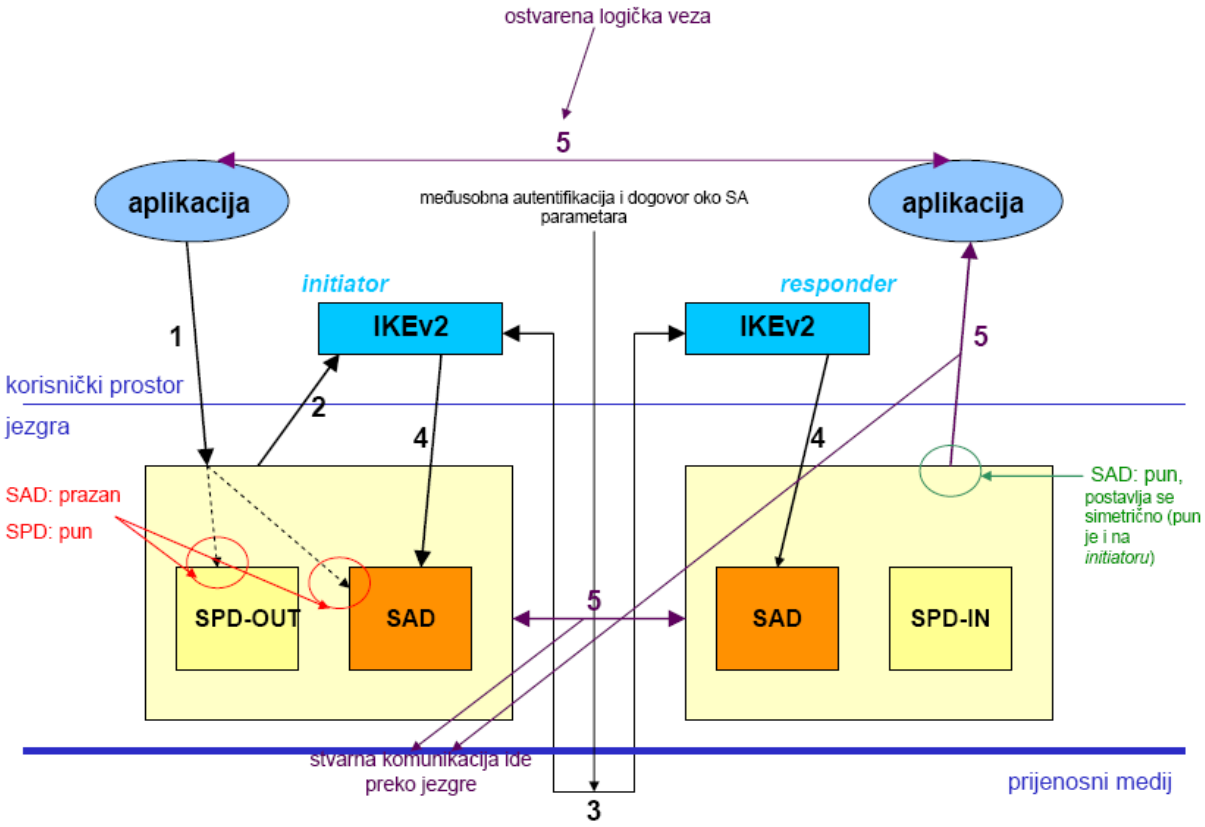
PSK ovjera se temelji na dijeljenoj tajni povezanoj s identitetom (svaki sugovornik ima spremljen takav par "dijeljena tajna – sugovornik"). Takva dijeljena tajna ne mora zaista biti dijeljena, a najčešće i nije – važno je samo da svaka IKEv2 implementacija ima za svakog sugovornika ima spremljenu dijeljenu tajnu povezanu s identitetom kojeg od sugovornika prima u IKEv2 poruci. Na prvi pogled, PSK ovjera ne donosi ništa novo u usporedbi s ručnom razmjenom ključeva (budući da se svakom sugovorniku ionako mora ručno unijeti dijeljena tajna). Unatoč tom nedostatku, IKEv2 pruža ostale prednosti pred ručnom razmjenom ključeva: dinamičko osvježavanje kriptografskog materijala proizvedenog iz dijeljene tajne (*rekeying*), periodičku ponovnu autentifikaciju (reauthentication), skrivanje identiteta, PFS zaštitu i dr.

RSA ovjera rješava problem ručnog unošenja dijeljene tajne temeljeći se na već ustanovljenim infrastrukturama za razmjenu ključeva (PKI, SPKI, PGP i sl) i najčešća je ovjera IKEv2 metoda. Obuhvaća skup različitih metoda ovjeree temelje na asimetričnom RSA algoritmu, a razlikuju prema dokumentima koji sadrže asimetrični kriptografski materijal na temelju kojeg se obavlja ovjera (sirovi asimetrični ključevi, certifikati i sl.):

- PKCS #7 wrapped X.509 certificate,
- PGP Certificate,
- DNS Signed Key,
- X.509 Certificate – Signature,
- Kerberos Token,
- Certificate Revocation List (CRL),
- Authority Revocation List (ARL),
- SPKI Certificate,
- X.509 Certificate – Attribute,
- Raw RSA Key,
- Hash and URL of X.509 certificate,
- Hash and URL of X.509 bundle.

Od navedenih RSA metoda ovjere specifikacija detaljnije opisuje jedino X.509 Certificate – Signature i Hash and URL of X.509 certificate. Ove dvije metode će biti opisane u nastavku rada. Dodatno, biti će opisan profil IKEv2 tereta vezanih uz RSA ovjeru te PAD baza.

Uz obostranu ovjeru IKEv2 nakon dogovora oko svih parametara za zaštitu sigurnosnog udruženja te nakon uspješne obostrane ovjere u jezgru operacijskog sustava sprema sigurnosna udruženja (SA). Na slici (Slika 2: Uspostava IKEv2 sigurnosnog udruženja) prikazan je postupak uspostavljanja sigurnosnog udruženja:



Slika 2: Uspostava IKEv2 sigurnosnog udruženja

Inicijator šalje prema odgovaratelju IP paket. Inicijator ima u jezgri zapisano da se svi paketi za odgovaratelja moraju zaštiti (tj. inicijator ima u jezgri u SPD bazi SP unos koji definira zaštitu tog prometa).

1. Budući da inicijator ima u jezgri samo sigurnosnu politiku (SP), a ne i SA udruženje, aktivira se njegov IKEv2 *daemon.*
2. IKEv2 daemon inicijatora započinje komunikaciju s IKEv2 *daemonom* odgovaratelja kako bi razmijenili dijeljene ključeve i dogovorili se oko skupa algoritama kojima će zaštiti SA poveznicu.
3. *Daemoni* su razmijenili dijeljene ključeve i dogovorili se oko kirptografskih algoritama te možemo reći da istovremeno zapisuju taj ključ i algoritme u jezgru operacijskog sustava - formiraju novi unos u SAD bazi određen IP adresama sugovornika, SPI brojem, akcijom, tipom protokola i razinom zaštite, dogovorenim kriptografskim algoritmima i dr. Takav SA

koji je IKEv2 uspostavio, naziva se IKE SA, a uz njega se u inicijalnom koraku stvara i CHILD SA - sigurnosno udruženje kojom se razmjenjuje traženi promet.

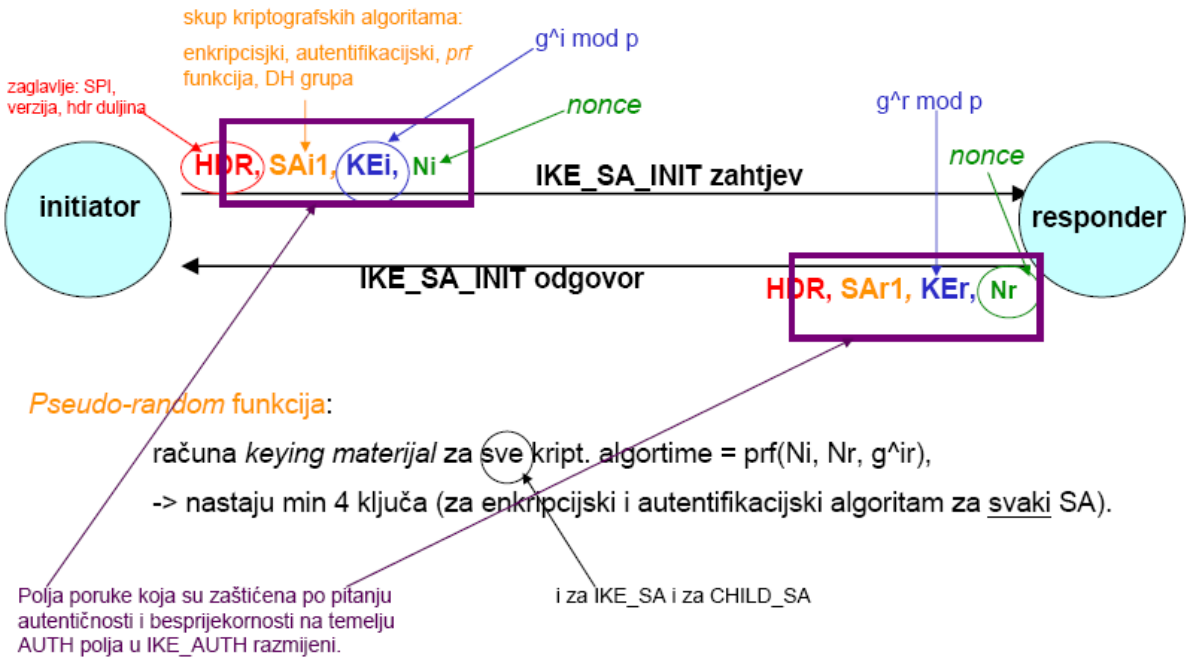
4. Nakon koraka 4, ukoliko se ne događaju nikakve nepredvidive situacije (npr. napad) *daemon* može mirovati (sve do promijene ključeva). U tom koraku, između uspostave SA nakon generiranja ključeva i uspostave novog SA nakon generiranja novih ključeva, kroz nesigurnu se mrežu prenosi zaštićen traženi promet (sigurnom poveznicom koja će se stvoriti kroz uspostavljenu IKE SA - naziva se, kao što je rečeno, CHILD SA). Bob na temelju pridruženog SP unosa u jezgri, prepoznaje koji promet treba zaštiti, a prema SPI broju i drugim parametrima pronalazi pripadni SA.

IKE SA je prvo sigurnosno udruženje koja se uspostavlja između sugovornika, a preko koje se prenosi IKE promet. Takav SA je logički (ne postoji u jezgri u operacijskog sustava) i jednosmjeran. Sva SA udruženja koja se stvaraju kroz IKE SA (prilikom promjene ključa (*rekeying*)) nazivaju se CHILD SA i preko njih se prenosi traženi promet. Zahtjev za ponovnim izračunom ključeva *(rekeying)* bilo koji od sugovornika može primiti od jezgre (kod isteka tzv. *soft expiry* parametra) ili od drugog sugovomika (ukoliko je njemu istekao tzv. *softlimit time* parametar i pošalje zahtjev za *rekeyingom*). IKE SA ostaje aktivna još neko vrijeme (do isteka tzv. *hard expiry* parametra). *Hard expiry* kod IKE SA naziva se vrijeme reautentifikacije (budući da se mora ponovo proći kroz proces ovjere).

4.1.3. Tipovi poruka

IKE poruke razmjenjuju se UDP protokolom (najčešće preko porta 500). Zamijetimo za početak da je kod IPsec komunikacije s *daemonom* za razmjenu ključeva potrebno zaštiti i traženi promet, ali i IKE promet (poruke koje se razmijenjuju između dva *daemona*). IKE poruke se uvijek razmjenjuju u parovima (*exchanges*) pri čemu se za sugovornike upotrebljavaju termini inicijator (*initiator)* i odgovaratelj (*responder*) [5]. IKE SA i prva CHILD SA uspostavlja se kroz četiri poruke:

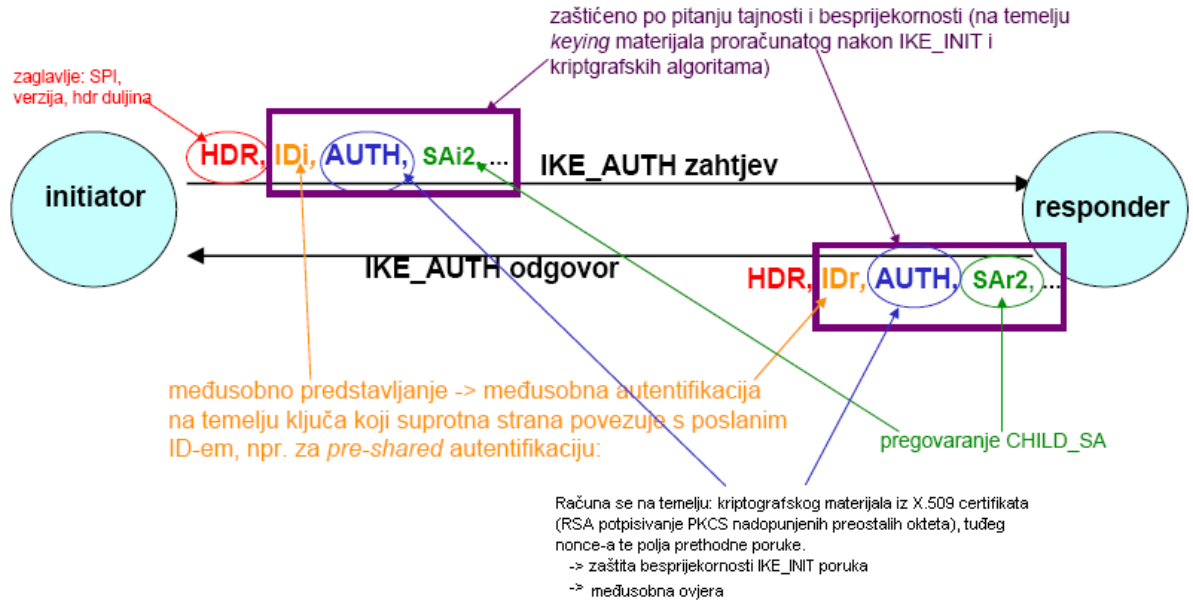
1. IKE_SA_INIT *exchange,*
2. IKE_AUTH *exchange.*



Slika 3: IKE SA INIT razmjena

Prvim parom poruka (IKE_SA_INIT, Slika 3: IKE SA INIT razmjena), inicijator i *responder* se dogovaraju oko kriptografskih algoritama i drugih parametara kojima će štiti traženi promet (uobičajeno je za skup takvih protokola upotrijebiti pojam kriptografskog odijela (*cryptographic suite*), a za cijeli skup parametara ponuda (*proposal*), razmjenjuju jednom korištene brojeve (*nonces)* i obavljaju Diffie-Helman razmjenu (asimetričnim kriptosustavom razmjenjuju dijeljenu tajnu ili asimetrične ključeve iz certifikata). IKE_SA_INIT par poruka je kriptiran (zaštićen je enkripcijskim algoritmom definiranim za zaštitu IKE prometa), no nisu još ovjerene jer se podaci potrebni za ovjeru tek prenose ovim porukama, a inicijator i odgovaratelj će se međusobno predstaviti jedan drugome tek u sljedećem paru poruka.

Drugi par poruka (IKE_AUTH, Slika 4: IKE AUTH razmjena) služi ovjeri prethodno razmijenjenih poruka (izvodi se HMAC na temelju sadržaja iz prethodnih poruka i dijeljene tajne). Ovim se parom poruka sugovornici predstavljaju jedan drugome (šalju svoje identitete i/ili certifikate) i uspostavlja se prvo CHILD SA udruženje kojim se prenosi traženi promet. IKE_AUTH poruke su i kriptirane i ovjerene na temelju ključeva koji su razmijenjeni prvim parom poruka (IKE_SA_INIT). Postoje situacije u kojima je za IKE SA potrebno razmijeniti više od četiri poruke, npr:



Slika 4: IKE AUTH razmjena

- U slučaju detekcije prevelikog broja poluotvorenih veza (što se tipično događa kod DOS napada) potreban je dodatan par poruka zbog razmijene tzv. *cookie* vrijednosti - ovo je princip zadovoljavanja sigurnosnog zahtjeva raspoloživosti,
- U slučaju EAP autentifikacije (*Extensible Authentication Protocol)* također je potrebno razmijeniti dodatne poruke.

Navedene četiri poruke moraju se razmijeniti točno ovakvim navedenim redoslijedom, a nakon te četiri poruke počinje se razmjenjivati traženi promet zaštićen SA usruženjem (kriptiran i ovjeren). Sljedeću IKE poruku, nakon što je IKE SA jednom uspostavljen, može poslati bilo koji od sugovornika (neovisno o tome da li on započeo komunikaciju slanjem IKE SA INIT zahtjeva). Poruke nakon četiri početne poruke mogu biti:

- CREATE_CHILD_SA zahtjev ili odgovor,
- INFORMATIONAL poruke.

CREATE_CHILD_SA zahtjev je zahtjev za stvaranjem novog SA udruženja (uslijed *rekeyinga*). Očito je, ovu će poruku poslati bilo inicijator bilo odgovaratelj - onaj kome prvome istekne *soft expiry*. Ovim se parom poruka opcionalno može obaviti dodatna Diffie-Hellman razmjena (kako bi se omogućila jača *perfect forward secrecy* zaštita). Razmjenjuju se *nonce* brojevi (kako bi se uspostavio jedinstveni ključ za taj CHILD SA i opcionalno se dogovaraju tzv. prometni selektori (*traffic selectors*) koji definiraju IP adrese, pristupe (*ports*) i protokole koji će se prenositi tom CHILD SA. PFS (*perfect forward secrecy*) jedna je od karakteristika automatske razmijene ključeva - ukoliko napadač uspije otkriti trenutni dijeljeni ključ, ne može proračunati niti do tad upotrebljavane djeljene ključeve, niti one koji će se tek generirati. Ova karakteristika zasniva se na Diffie-Hellman razmijeni. Upravo je PFS svojstvo razlog zbog kojeg se uz DH eksponente razmijenjuju i jednom korišteni brojevi. Ukoliko bi se svaki puta (za svaki novi SA - u CREATE_CHILD_SA porukama) upotrebljavao jedinstveni privatni DH broj, potreba za jednom korištenim brojevima (koji moraju biti jedinstveni za svaki SA) zapravo uopće ne bi postojala. No, kako bi se izbjegli "skupi" DH eksponent DH brojevi se ponekad ponovo iskorištavaju (*concept of reusing DH exponents*).

INFORMATIONAL razmjena može se dogoditi nakon inicijalne četiri poruke i obično detektira neočekivana stanja. Primjeri neočekivan stanja i NOTIFY tereta koji se umeću u INFORMATIONAL poruku su:

- kod DOS napada – INVALID_COOKIE,
- nepoznati SPI broj – INVALID_IKE_SPI,
- neodgovarajući zatraženi skup kriptografskih algoritama – NO_PROPOSAL_CHOSEN,
- poruke za brisanje SA - DELETE INFORMATIONAL.

4.1.4. RSA ovjera temeljena na certifikatima

Za razliku od PSK autentifikacije kod koje se uvijek koristi samo osnovni tereti IKE SA INIT i IKE AUTH poruka, RSA ovjera omogućava razmjenu kriptografskog materijala:

- nekim vanjskim sredstvom (*out-of-band exchange*),
- pomoću CERT tereta te CERTREQ tereta.

Uloga CERT tereta je prijenos certifikata u nekom od specificiranih oblika, dok CERTREQ teret sadrži informacije koje sugovorniku služe kao sugestija kod odabira certifikata kojima će odgovoriti na CERTREQ teretu.

U slučaju RSA ovjere oba sugovornika te razmjene CERTREQ i CERT tereta prve četiri poruke izgledaju (Ispis 20: IKE SA INIT i IKE AUTH uz RSA ovjeru):

Ispis 20: IKE SA INIT i IKE AUTH uz RSA ovjeru

```
      Initiator                          Responder
     -----------                        -----------
      HDR, SAi1, KEi, Ni    -->
                            <--    HDR, SAr1, KEr, Nr, CERTREQ
```

```
      HDR, SK {IDi, CERT, CERTREQ, [IDr,]
                  AUTH, SAi2, TSi, TSr}     -->
                                    <--    HDR, SK {IDr, CERT, AUTH,
                                                  SAr2, TSi, Tsr}
```

RSA ovjera se zasniva na već ustanovljenim infrastrukturama za razmjenu ključeva (PKI, SPKI, PGP i sl) čime je olakšana razmjena ključeva, identifikacijskih podataka i sl. Ovo čini osnovnu prednost RSA ovjere pred PSK ovjerom. vjeru Za ovaj zahtjev IKEv2 specifikacije [8] zapravo ne postoji nikakvo opravdanje i jedan je od propusta u specifikaciji. Budući da sve novije EAP metode obavljaju obostranu ovjeru i proizvode MSK vrijednost, ne postoji nikakva potreba da se odgovaratelj još jednom dodatno ovjeri RSA ovjerom. Ovaj specifikacijski propust je posljedica činjenice da su prve EAP metode bile jednosmjerne i nesigurne, no to danas više nije slučaj. Posljedica zahtjeva da odgovaratelj nužno mora u tom scenariju koristiti RSA ovjeru, ograničava njegovu slobodu izbora metode ovjere. [6] predlaže korištenje EAP metoda s tzv. *channel binding* podrškom umjesto odgovarateljeve RSA ovjere.

4.1.5. Tereti CERT i CERTREQ

CERTREQ teret (Tablica 3: Izgled i sadržaj CERTREQ tereta) sadrži informacije kojim se sugovornika obavještava o certifikacijskim tijelima kojima vjerujemo (u tablicama su samo tereti implementirani u ikev2 *daemonu*):

Tablica 3: Izgled i sadržaj CERTREQ tereta

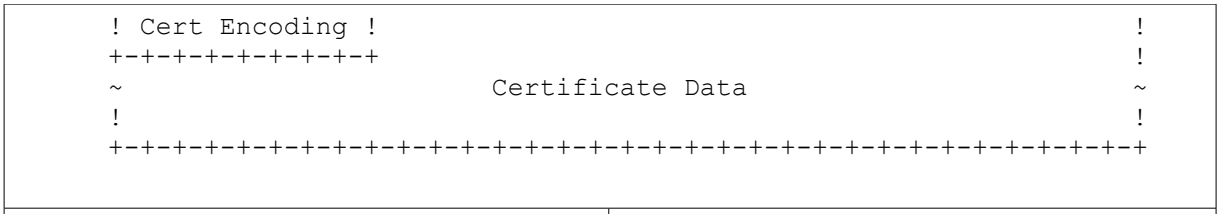
```
                            1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       ! Next Payload  !C!  RESERVED   !         Payload Length        !
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       ! Cert Encoding !                                               !
       +-+-+-+-+-+-+-+-+                                               !
       ~                    Certification Authority                    ~
       !                                                               !
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Tip CERTREQ tereta:	Sadržaj CERTREQ tereta:
X.509 Certificate – Signature Hash and URL of X.509 certificate 12	SHA-1 sažetak javnog ključa CA certifikata

Svaki od sugovornika stvara CERTREQ teret na temelju CA certifikata koje posjeduje: iz certifikata čita javni ključ i stvara SHA-1 sažetak. Primatelj CERTREQ tereta pretražuje skup certifikata koje posjeduje, proizvodi sažetke iz javnih ključeva tih certifikata u potrazi za certifikatom koji je izdao neki od certifikacijskih centara iz CERTREQ tereta te odgovara takvim certifikatom. Certifikat odabran na temelju CERTREQ tereta šalje se prema sugovorniku unutar CERT tereta (Tablica 4: Izgled i sadržaj CERT tereta).

Tablica 4: Izgled i sadržaj CERT tereta

```
                            1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       ! Next Payload  !C!  RESERVED   !         Payload Length        !
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



Tip CERT tereta:	Sadržaj CERT tereta:
X.509 Certificate – Signature	DER kodiran X.509 certifikat čiji se javni ključ koristi za verifikaciju primljenog AUTH payloada
Hash and URL of X.509 certificate	SHA-1 sažetak DER kodiranog X.509 certifikata i URL koji se razlučuje u originalni certifikat

Primatelj CERT tereta može:

- ignorirati primljeni CERT payload ako posjeduje lokalno pohranjen sugovornikom certifikat ili ako kriptografski materijal sugovornici razmjenjuju na neki vanjski način.
- Nastaviti s verifikacijom primljenog certifikata.

IKEv2 specifikacija ne definira da se CERT i CERTREQ nužno moraju razmijeniti u parovima, no [5] preporuča da se CERT teret šalje uvijek i jedino kao odgovor na CERTREQ teret. Na ovaj specifikacijski zahtjev je važno obratiti pažnju prilikom implementacije mogućnosti da sugovornici koriste različite metode ovjere (npr. inicijator koristi PSK, odgovaratelj RSA). Zamijetimo da u trenutku kada icijator i odgovaratelj šalju CERTREQ teret, oni još ne znaju koju metodu ovjere njihov sugovornik namjerava koristiti. Dodatno, oni još niti ne znaju tko je njihov sugovornik jer se identiteti razmijenjenju tek u IKE AUTH porukama unutar ID tereta. Zato je prikladno slati ID teret po defaultu, neovisno o tome da li sami koristimo RSA ili PSK ovjeru. Ignoriranje certifikata u slučaju lokalno posjedovanog certifikata primjer je propusta IKEv2 protokola. Ukoliko netko posjeduje sugovornikov certifikat lokalno pohranjen, uzaludno je slati CERT teret da bi ga primatelj onda jednostavno ignorirao. Dodatni tereti ili dijelovi tereta koji bi riješili ovaj problem nisu specificirani.

Najčešći tip CERT tereta je X.509 Certificate Signature. [5] predlaže da svaka implementacija koja podržava ovaj tip, podrži i Hash and URL of X.509 certificate iz slijedećih razloga:

- kod uređaja koji nemaju prostor za pohranjivanje certifikata,
- kod niske propusnosti veze,
- zbog smanjenja veličine IKE poruke.

Prva dva razloga uglavnom se pojavljuju kod malih mobilnih uređaja, dok je treći razlog iznimno važan zbog problema UDP fragmentacije. Prevelike UDP poruke i IP fragmenti otvaraju mogućnost za DoS napad, a uz to, neki NAT uređaji i vatrozidi blokiraju IP fragmente.

Hash and URL CERT teret sastoji se sažetka certifikata i HTTP URL-a. PKIX radna grupa zalagala se za uvođenje novog polja u X.509 certifikat u kojem bi bio spremljen HTTP URL, no prijedlog nije bio prihvaćen. Primatelj Hash and URL CERT tereta, dohvaća certifikat s navedene lokacije, izrađuje sažetak i uspoređuje ga s primljenim sažetkom.

Važnost Hash and URL tipa CERT tereta dolazi do izražaja kod razmjene velikog broja certifikata (zbog smanjenja IKEv2 poruke), bilo da se razmjenjuje skup certifikata (*certificate*

bundle) ili lanac certifikata (*certificate chain*). Lanac certifikata se sastoji od klijentskog certifikata, središnjih povezanih certifikata (preko javnog ključa) prema root CA certifikatu. Širi pojam od lanca certifikata je skup certifikata, koji može označavati i skup klijentskih certifikata. Primjerice, korisnik može posjedovati dva certifikata – X.509 Certificate – Signature i X.509 Certificate Attribute. U tom slučaju X.509 Certificate Attribute se koristi za razmjenu dodatnih autorizacijskih informacija. Drugi primjer razmjene skupa certifikata su X.509 certifikati koji imaju jednak javni ključ, razlikuju se podacima u poljima s identitetom (Subject polje ili subjectAltName ekstenzija). Primjer neophodnosti razmjene takvog skupa certifikata (s dodatnim identifikacijskim podacima) je u slučaju kada je IKEv2 sugovornik konfiguriran tako da provjerava jednakost identiteta primljenog u IKEv2 poruci s identitetom zapisanim u primljenom certifikata u CERT teretu. U oba slučaja (i kod lanca certifikata i skupa certifikata) u jednom CERT teretu se može nalaziti samo jedan certifikat, a za izračun AUTH tereta koristi se javni ključ iz prvog poslanog CERT tereta.

4.1.6. Identifikacijski teret

Identifikacijski teret (ID *payload*) u IKEv2 porukama sadrži identitet kojeg inicijator i *responder* umeću u IKE AUTH zahtjev odnosno odgovor. ID teret koristi se za:

- Ovjeru. Ovjera u IKEv2 je potvrda identiteta primljenog u ID teretu. Pri tome se ID payload koristi za izračun AUTH tereta.
- Pretraživanje PAD baze. PAD baza sadrži podatke o ovjeri sugovornika, a pojedini element baze dohvaća se na temelju primljenog identiteta.
- Za tzv. simbolički SPD dohvat (*symbolic SPD lookup*). To je proces odabira određenog SP-a na temelju identiteta (iz ID payloada), za razliku od "običnog SPD dohvata" koji predstavlja dohvat određenog SP-a na temelju IP adresa (iz TS payloada).
- Za dohvat certifikata ukoliko lokalno postoji baza s certifikatima (opcionalno).

ID teret ne mora biti popunjen nikakvim identifikacijskim podatcima iz okoline, npr. iz IP zaglavlja. Primjerice, neka *roadwarrior* klijent koristi svoju privatnu IP adresu kao identitet unutar ID tereta i ima certifikat u kojem je ta adresa navedena kao identifikacijski podataka u subjectAltName ekstenziji. Klijent se može predstavljati takvim identitetom i certifikatom i onda kada nije u svojoj privatnoj mreži. Jedini je zahtjev da niti jedan od preostalih IKEv2 sugovornika ne koristi isti takav ID.

IKEv2 definira nekoliko tipova ID payloada:

- ID_IPV4(6)_ADDR IPv4 ili IPv6 adresa,
- ID_FQDN. FQDN (Fully-qualified domain name string),
- ID_RFC822_ADDR. E-mail adresa (prema rfc822),
- ID_DER_ASN1_DN. DER (Distinguished Enconding Rules) zapis ASN.1 X.500 DN-a (Distinguished Name),
- ID_DER_ASN1_GN. DER zapis ASN.1 X.500 GN-a (General Name),
- ID_KEY_ID. Identifikator proizvođača.

Odluka o tome koji identifikacijski tip koristiti ovisi o scenariju. Iz sigurnosnog aspekta, niti jedan od tipova nema prednost. Na primjer, upotreba ID_RFC822_ADDR, ID_FQDN i ID_KEY_ID-a tipova prikladna je u slučaju velikog broja sugovornika s dinamičkim IP adresama. U slučaju malog broja sugovornika sa statičkim IP adresama prikladno je koristiti ID_IPV4(6)_ADDR identifikacijski tip. Također, običaj je upotrebljavati ID_DER_ASN1_DN tip isključivo kod RSA ovjere temeljene na nekom od oblika X.509 certifikata, budući da taj tip odgovara Subject polju iz certifikata. Primatelj IKE AUTH poruke onda može ili ne mora (to je lokalna odluka) obavljati provjeru jednakosti ta dva identiteta ([5] zahtjeva da takva karakteristika mora biti konfigurabilna). Slično, ID_KEY_ID nije običaj upotrebljavati u slučaju RSA ovjere, budući da ne postoji X.509 ekstenzija koja može sadržavati informacije iz takvog tereta što bi uzrokovalo problem interoperabilnosti kod provjere jednakosti identiteta iz ID tereta i certifikata. Također, ID_DER_ASN1_GN prema [5] nije preporučljiv za korištenje u IKEv2 protokolu. Za razliku od ID_DER_ASN1_DN tipa čiji je sadržaj striktno određen (Distinguished Name), ID_DER_ASN1_GN može spremiti mnogo različitih oblika identiteta uključujući i one koji se mogu spremiti u subjectAltName ekstenzije. Odlučeno je ipak da su ekstenzije, a ID_DER_ASN1_GN tip preliberalan po tom pitanje i stoga nije preporučljiv za korištenje.

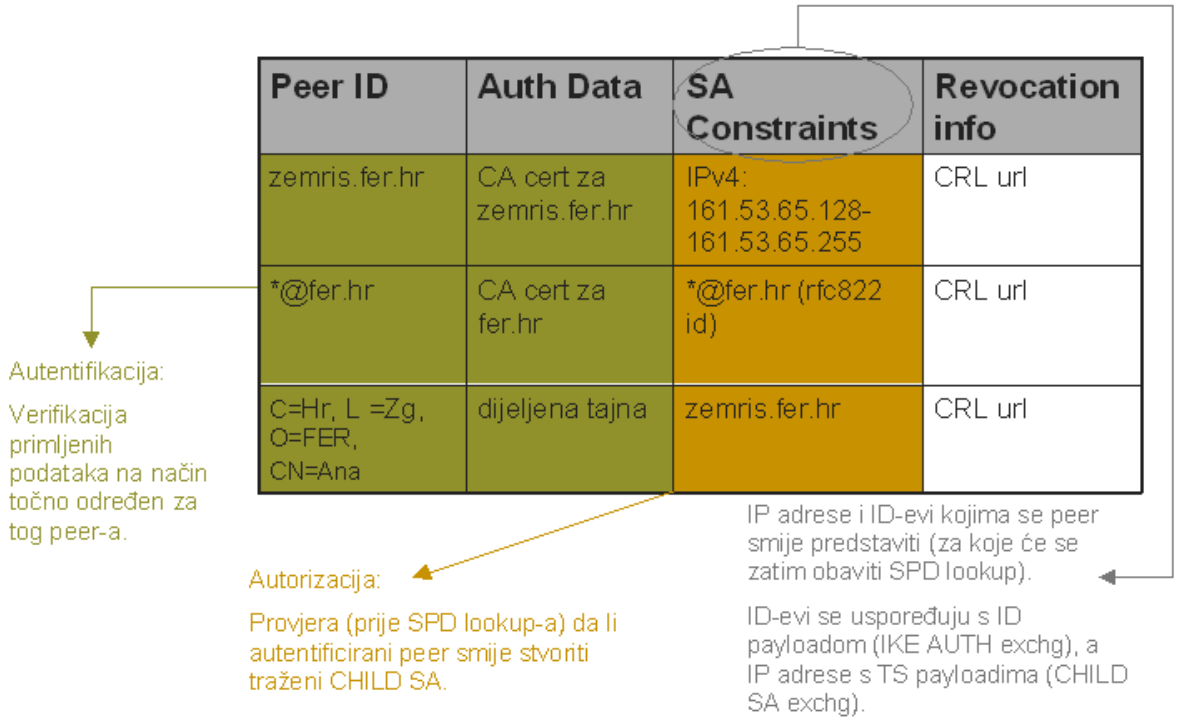
4.1.7. Autorizacijska baza

Dvije temeljne baze IPSec arhitekture su SPD i SAD baza, a implementiraju se u jezgri operacijskog sustava. Treća baza koja je sastavni dio IPSec arhitekture do izražaja dolazi tek kod kompliciranijih scenarija ovjere (posebice kod RSA ovjere temeljene na certifikatima), a dodatno sadrži i autorizacijske informacije za dodjelu sigurnosnih udruženja (SA-ova). Ta se baza naziva PAD baza (Peer Authorization Database) i čini povezanicu između IKEv2 protokola i SPD baze. Za razliku od SPD i SAD baze, PAD baza nije postoji u jezgri operacijskog sustava, a IPSec specifikacija opisuje funkcionalnosti koje ona mora pružiti ostavljajući odluku o načinu njene implementacije lokalnom.

PAD baza (Slika 5: Autorizacijska baza arhitekture IPSec) je podijeljena na dva dijela:

- ovjerni dio,
- autorizacijski dio.

Ovjerni dio definira koju metodu ovjere partner koristi te koji materijal za ovjeru treba koristiti kod verifikacije njegovog materijala za ovjeru. Autorizacijski dio sadrži informacije na kojima se temelji provjera prava dodjele CHILD SA-ova partneru s nekim određenim identitetom. Takav identitet ne mora nužno biti iz ID tereta, nego primjerice iz certifikata ili to može biti identitet ovjeren unutar EAP metode.



Slika 5: Autorizacijska baza arhitekture IPSec

Ovjerni dijelovi PAD baze su:

- Identitet partnera. Tipovi identiteta su jednaki tipovi identiteta u IKEv2 ID teretu.
- Ovjerne podatke. Dijeljena tajna, klijentski certifikat, CA certifikat.
- Informacije o opozivanju ovjernog materijala. Pokazivač na CRL ili OCSP server i pripadajući javni ključ.

Autorizacijski dijelovi PAD baze su:

- adrese ili identiteti kojima se partner smije predstaviti da bi mu se nekad toga omgućila dodjela zatraženog CHILD SA.

Uspostava sigurnosnog udruženja temeljena na PAD bazi odvija se kroz tri koraka:

1. Ovjera. Nakon primitka partnerovog identiteta i autentifikacijskog materijala, primatelj verificira ovjerni materijal kako je to opisano u elementu PAD baze koji odgovara partnerovom identitetu.
2. Autorizacija. U slučaju uspješne verifikacije primljenog ovjernog materijala, primatelj provjerava da li smije tom pošiljatelju dodijeliti zatraženi CHILD SA (tzv. CHILD SA *constraining*).
3. SPD dohvat. Nakon uspješne ovjere i autorizacije, obavlja se postupak dohvata SP-a. Ukoliko je SPD dohvat temelj na PAD bazi uz zahtjev obavezne jednakosti identiteta iz ID payloada i certifikata, SPD dohvat se naziva sigumi SPD dohvat (secure SPD lookup).

PAD baza dolazi do izražaja tek u kompliciranijim autentifikacijskim scenarijima. Zamislimo IKEv2 SGW koji ne obavlja ovjeru kakvu omogućuje PAD baza. Neka klijent posjeduje dva certifikata: CERT1 izdan od tvrtke T1 i CERT2 izdan od tvrtke T2. Iza SGW-a nalaze se privatna mreža tvrtke T1 i privatna mreža tvrtke T2, a SGW posjeduje i T1 CA certifikat (CA_CERT1) i T2 CA certifikat (CA_CERT2). Pretpostavka je da se klijent prema tvrtki T1 treba ovjeriti s CERT1 certifikatom, no ako SGW nema PAD element koji će ga informirati o tome da za pristup do T1 mora verificirati CERT1 s CA_CERT1, SGW nikako ne može spriječiti klijenta da se ne pokuša ovjeriti s certifikatom CERT2. Ukoliko primjerice CERT1 istekne (a CERT2 ostane valjan), SGW će propustiti klijenta do mreže T1 jer će sve primljene certifikate pokušati verificirati sa svim CA certifikatima koje posjeduje – uspjeti će verificirati certifikat CERT2 s CA_CERT2.

Ipak, PAD baza ne rješava sve probleme vezane uz izbjegavanje heuristike kod odabira ovjernog materijala koji se šalje partneru ili onog koji se koristi kod verifikacije primljenog ovjernog materijala. Do problema dolazi u situacijama kada identitet sugovornika još nije poznat, a podaci iz PAD elementa vezanog uz partnera su potrebni. Primjeri takvih situacija su:

- odabir CA certifikata za oblikovanje CERTREQ payloada,
- odabir certifikata za oblikovanje CERT payloada.

CERTREQ teret šalje se prije primitika IKE AUTH poruke s ID teretom. U trenutku stvaranja i slanja CERTREQ payloada (IKE SA INIT odgovor ili IKE AUTH) ne postoji način saznavanja partnerovog identiteta i oblikovanja CERTREQ-a na temelju njemu pripadajućih CA certifikata. Stoga je običaj u CERTREQ teretu poslati sažetke javnih ključeva iz svih valjanih certifikata.

Niti u sljedećem koraku (nakon primitka CERTREQ tereta), ne zna se identitet sugovornika. Zbog toga se certifikati za slanje odabiru na sljedeći način:

- šalju se svi certifikati koji odgovaraju primljenom CERTREQ teretu ako takvi postoje,
- ako ne postoje certifikati koji odgovaraju primljenom CERTREQ teretu, implementacije može prekinuti uspostavljanje sigurnosnog udruženja ili poslati sve certifikate koje posjeduje (što ostaje lokalna odluka).

5. Implementacija

U ovom je poglavlju opis implementacije IKEv2 ovjere temeljene na X.509 certifikatima, uz korištenje OpenSSL i libcurl knjižnica.

Izdvojimo stanja ikev2 automata koja zahtijevaju promijene u svrhu implementacije ovjere temeljene na certifikatima (detaljnije je opisan samo jedan smjer ovjere – ovjera inicijatora prema odgovaratelju, dok je za drugi smjer navedeno samo kada i koji tereti se oblikuju):

- IKE_SMR_INIT. Stanje odgovaratelja vezano uz IKE SA INIT razmjenu. U ovom stanju responder oblikuje CERTREQ teret na temelju svih CA certifikata koje posjeduje, a navedenih u sekciji general.
- IKE_SMI_AUTH. Stanje inicijatora vezano uz IKE AUTH razmjenu. Inicijator je primio CERTREQ u IKE SA INIT zahtjevu te oblikuje CERT, AUTH i ID terete. CERT terete oblikuje na temelju sažetaka javnih ključeva iz CERTREQ-a između certifikata definiranih konfiguracijskim parametrom u sekciji general. Nakon oblikovanja CERT tereta, čita privatni ključ certifikata iz prvog od CERT tereta i koristi ga za izračun AUTH tereta. ID teret oblikuje na temelju ručno unešenog identiteta u konfiguracijskom parametru my_identifier u sekciji peer ili na temelju identiteta pročitanog iz certifikata (definiranog u istom konfiguracijskom parametru). U svrhu vlastite ovjere prema responderu, inicijator oblikuje i šalje CERTREQ.
- IKE_SMR_AUTH. Stanje odgovaratelja vezano uz IKE AUTH razmjenu. Odgovaratelj je primio IKE AUTH zahtjev. Na temelju identiteta iz ID tereta odabire PAD element pridružen primljenom identitu (tj. odabire odgovarajuću peer sekciju konfiguracijske datoteke). Verificira certifikate iz primljenih CERT tereta te na temelju javnog ključa certifikata iz prvog CERT tereta provjerava valjanost primljenog AUTH tereta. Ukoliko je konfiguracijski parametar verify_id uključen, provjerava da li identitet primljen u ID teretu odgovara identitu iz nekog od primljenih certifikata (identitu iz Subject polja ili subjectAltName ekstenzije). Ukoliko je svaki od prethodnih koraka uspješno obavljen, responder odgovara s IKE AUTH porukom u koju umeće CERT terete oblikovane na temelju primljenog CERTREQ-a.
- IKE_SMI_INSTALLCSA. Stanje inicijatora nakon što su svi CERTREQ i CERT tereti razmjenjeni, a prije uspostave IKE SA i CHILD SA preostaje još samo verifikacija certifikata iz IKE AUTH odgovora.

Moguće je da jedan od sudionika koristi bilo koju drugu metodu ovjere. U tom je slučaju sugovornik mora posjedovati prikladan CA certifikat kako bi mogao verificirati primljene CERT terete i AUTH teret. Također, sudionik koji koristi RSA autentifikaciju mora posjedovati kriptografski materijal (dijeljenu tajnu u slučaju PSK ovjere) kako bi verificirao partnerove primljene terete. Također, postoje i druge varijacije ovog scenarija – primjerice, kada neki od sugovornika posjeduje lokalno pohranjene certifikate te ignorira primljene CERT terete.

5.1.Konfiguracijska datoteka

Za implementaciju funkcionalnosti PAD baze (propisane prema [7]) neophodna je prikladno oblikovana konfiguracijska datoteka (Ispis 21: Dio konfiguracijske datoteke ikev2.conf vezan uz RSA ovjeru temeljenu na X.509 certifikatima):

Ispis 21: Dio konfiguracijske datoteke ikev2.conf vezan uz RSA ovjeru temeljenu na X.509 certifikatima

```
general {
    ...
    ca type_x509 "ca_cert.pem";
    cert type_x509 "client_cert.pem" "client_privatekey.pem";
    ...
}
remote any
{
    nonce_size 32;
    proposal {
        encryption_algorithm 3des;
        auth_algorithm md5_96;
        pseudo_random_function md5;
        dh_group modp768;
    }
}
peer any {
    ...
    ca type_x509 "ca_cert.pem";
    auth_method rsa_sig;
    my_identifier fqdn moja.domena.com;
    ...

    sainfo local 192.168.247.129 remote 192.168.247.134 {
```

```
        auth_algorithm sha1_96;
        encryption_algorithm 3des;
    }
}
```

U primjeru su navedene sekcije konfiguracijske datoteke vezane uz autentifikaciju temeljenu na certifikatima:

- remote. Konfiguracijski parametri potrebni za odvijanje IKE SA INIT razmjene.
- peer. Konfiguracijski parametri potrebni za odvijanje IKE AUTH razmjene.
- general. Općeniti konfiguracijski parametri i svi oni konfiguracijski parametri koji se ne uklapaju u preostale sekcije. Primjerice, konfiguracijski parametar ca iz sekcije general služi za oblikovanje CERTREQ tereta. Logički pripada ovjernom dijelu sekcije specifične za određenog partnera (sekcija peer), no ti su podaci potrebni prije IKE AUTH razmjene, zbog čega taj konfiguracijski parametar ne može biti uvršten u sekciju peer.

Konfiguracijski parametri ca i cert unutar sekcije general imaju slijedeću ulogu:

- konfiguracijski parametar ca. Definira CA certifikate na temelju kojih se oblikuje CERTREQ teret. Budući da se postupak oblikovanja CERTREQ-a događa prije nego saznavanje partnerovog identiteta, sekcija general je jedino mjesto na kojem se može nalaziti ovaj konfiguracijski parametar.
- Konfiguracijski parametar cert. Definira klijentske certifikate koji se uzimaju u obzir kod oblikovanja CERT tereta na temelju primljenog CERTREQ-a. Postupak oblikovanja CERT payloada se također događa prije odabira PAD element vezanog za identitet partnera te se zbog toga nalazi u sekciji general. Konfiguracijski parametar se sastoji od dva dijela: datoteka s certifikatom te datoteka s privatnim ključem.

Sekcija remote sadrži skup algoritama: pseudo-slučajna funkcija (prf), algoritam ovjere, enkripcijski algoritam i DH grupa koje štite IKEv2 promet. Funkcija prf koristi se za izračunavanje AUTH tereta.

Sekcija peer sadrži konfiguracijske parametre karakteristične za određenog partnera. Između ostalih, tu se nalazi i podsekcija s autentifikacijskim podacima sa slijedećim konfiguracijskim parametrima:

- Konfiguracijski parametar ca. Definira CA certifikate koji se koriste u postupku verifikacije primljenog ovjernog materijala partnera (certifikata).
- Konfiguracijski parametar cert. Definira lokalno pohranjene certifikate partnera. Ukoliko takvi postoje, primljeni CERT teret se ignorira.

- Konfiguracijski parametar my_identifier. Definira sadržaj ID tereta koji se šalje partneru u IKE AUTH poruci. Korisniku je ostavljena mogućnost da identifikacijski podatak unosi sam ili da se identiet čita iz certifikata i njime popunjava ID teret (ključna riječ `from`) konfiguracijskim parametrom iz Ispis 22: Konfiguracijski parametar za popunjavanje tereta ID podacima iz certifikata:

Ispis 22: Konfiguracijski parametar za popunjavanje tereta ID podacima iz certifikata

```
my_identifier fqdn from "cert.pem";
my_identifier rfc822addr from "cert.pem";
my_identifier ipv4 from "cert.pem";
my_identifier der_asn1_dn from "cert.pem";
```

U prva tri (fqdn, rfc822addr, ipv4) reda, identitet se čita iz subjectAltName ekstenzije certifikata, a u zadnjem redu iz Subject polja certifikata.

Ukoliko se koristi Hash and URL tip X.509 certifikata (neovisno o tome u kojoj se sekciji nalazi), konfiguracijski parametar (ca ili cert) definira HTTP URL s kojeg se dohvaća certifikat i datoteku s certifikatom (kako bi se iz javnog ključa certifikata stvori SHA-1 sažetak koji se šalje u Hash and URL CERT teretu) kao što je prikazano u Ispis 23: Konfigurcijski parametri ca i cert za Hash and URL tip certifikata:

Ispis 23: Konfigurcijski parametri ca i cert za Hash and URL tip certifikata

```
ca type_x509 "ca_cert.pem" "http://moja.domena.com/cert.pem;

cert type_x509 "client_privatekey" "client_cert.pem"
       "http://ikev2certs.example.com/certificate.pem";
```

5.2.Glavne strukture

Postoje četiri osnovne strukture za pohranu podataka vezanih uz certifikate:

- struktura za pohranu podataka pročitanih iz primljenog CERT tereta,
- struktura za pohranu podataka pročitanih iz primljenog CERTREQ tereta,
- struktura za pohranu podataka iz konfiguracijske datoteke potrebnih za oblikovanje CERT tereta,
- struktura za pohranu podataka iz konfiguracijske datoteke potrebnih za oblikovanje CERTREQ tereta.

Tablica 5: Strukture za pohranu podataka iz konfiguracijske datoteke za oblikovanje CERT i CERTREQ tereta

```
struct ca_item {
   guint8 type;
   char *file;
   char *url;
}
```

```
struct cert_item {
   guint8 type;
   char *file;
   char *url;
   char *privkeyfile;
}
```

U tablici (Tablica 5: Strukture za pohranu podataka iz konfiguracijske datoteke za oblikovanje CERT i CERTREQ tereta) prikaza su strukture za pohranu podataka pročitanih iz konfiguracijske datoteke, a potrebnih za oblikovanje tereta CERT odnosno CERTREQ. U strukturi ca_item (tip X.509 Certificate – Signature ili Hash and URL of X.509 Certificate) nalazi se varijabla koja definira naziv datoteke certifikata (najčešće će to biti PEM format). Ukoliko je navedena varijabla postavljena na vrijednost NULL, CA certifikat ne postoji pohranjen lokalno te varijabla url definira lokaciju za dohvat CA certifikata. Slično vrijedi i za strukturu cert_item koja dodatno sadrži i varijablu koja pohranjuje naziv privatnog ključa koji pripada javnom ključu iz certifiata definiranog varijablama file ili url. Privatni ključ koristi se kod oblikovanja tereta AUTH.

Tablica 6: Strukture za pohranu podataka pročitanih iz primljenih CERT i CERTREQ tereta

```
struct cert {
     guint8 encoding;
     unsigned char *pub_key;
     guint16 pub_key_len;
     unsigned char *certificate;
     guint16 certificate_len;

};
```

```
struct certreq {
     guint8 encoding;
     unsigned char *hash_list;
     guint16 hash_list_len;
};
```

U tablici (Tablica 6: Strukture za pohranu podataka pročitanih iz primljenih CERT i CERTREQ tereta) nalaze se strukture za pohranu podataka pročitanih iz primljenih CERT i CERTREQ tereta. Struktura cert sadrži informaciju o tipu CERT tereta (encoding, X.509 Certificate – Signature ili Hash and URL of X.509 Certificate). U primljenom CERT teretu se nakon tipa CERT tereta nalazi DER kodiran X.509 certifikat. Iz takvog certifikata čita se javni ključ (potreban za verifikaciju AUTH tereta) i sprema u varijablu pub_key. Struktura certreq sadrži tip CERTREQ tereta (enconding, X.509 Certificate – Signature ili Hash and URL of X.509 Certificate) te listu SHA-1 sažetaka javnih ključeva CA certifikata sugovornika. U CERTREQ teretu, okteti liste hash_list nalaze se zapisani nakon tipa certifikata te su izravni kopirani u varijablu hash_list.

Sami tereti CERT i CERTREQ definirani su strukturom payload_cert (Tablica 7: Struktura payload_cert za izravnu pohranu podataka iz CERT i CERTREQ tereta):

Tablica 7: Struktura payload_cert za izravnu pohranu podataka iz CERT i CERTREQ tereta

```
                      1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ! Next Payload  !C!  RESERVED   !         Payload Length        !
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ! Cert Encoding !                                               !
 +-+-+-+-+-+-+-+-+                                               !
```

```
struct payload_cert {
   guint8 next;
   guint8 c:1;
   guint8 reserved:7;
   guint16 length;
   guint8 encoding;
}
```

```
~                       Certificate Data                        ~
!                                                               !
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
__attribute__((packed
));
```

5.3.Knjižnica OpenSSL

Oblikovanje i obrada CERT i CERTREQ tereta kao i izračun te verifikacija AUTH tereta, zasnivaju se na funkcijama knjižnice OpenSSL [9].

5.3.1. Pretvorba formata certifikata

Jedna od češće upotrebljavanih funkcija je ona za pretvorbu formata certifikata. Certifikat je u CERT teretu pohranjen u binarnom obliku (DER kodiran ASN.1 X.509 certifikat). Prije verifikacije nužno ga je pretvoriti iz tog oblika u interni X.509 oblik pri čemu se koristi neka of d2i_* (*der to internal*) funkcija. Obrnuta funkcija je sličnog naziva – i2d_*. U tablici () se nalazi primjer kodiranja (d2i) i dekodiranja (i2d) X.509 certifikata za OpenSSL 0.9.8:

```
int len;
unsigned char *buf;

buf = NULL;

len = i2d_X509(x, &buf);

if (len < 0)
       /* greska */
```

```
X509 *x;

unsigned char *buf, *p;

int len;

/* Postavi buf i len .. */

p = buf;

x = NULL;

if(!d2i_X509(&x, &p, len))
   /* greska */
```

5.3.2. Pretvorba identiteta tipa DN

Identitet u Distinguished Name obliku prema sugovorniku se šalje kao DER kodiran identitet. Ukoliko se identitet ne čita izravno iz certifikata tj. ukoliko ga upisuje korisnik, tada je takav znakovni niz potrebno pretvoriti u DER ASN.1 X.509 NAME oblik (X509_NAME). Postupak započinje stvaranjem X509_NAME strukture kojoj se dodaju pojedina Distinguished Name polja (Common Name, Location, Organization) odvojena zarezom. E-mail adresa nije standardni dio Subject polja i navodi se odijeljena kosom crtom (/). Preporuka je zanemariti taj dio Subject polja i pohraniti e-mail adresu kao X.509v3 ekstenziju. Opisanu pretvorbu obavlja funkcija u ispisu Ispis 24: Pretvorba znakovnog niza (Subject polja) u X509_NAME i DER ASN.1 format:

Ispis 24: Pretvorba znakovnog niza (Subject polja) u X509_NAME i DER ASN.1 format

```
X509_NAME *name;
char *der_asn1dn = NULL;
char *buf, *field, *bytes;
int i, der_len;
```

```
    gboolean field_pos;
    unsigned char *ptr = NULL;
    unsigned char **der = &ptr;


    buf = g_memdup(string, len);

    name = X509_NAME_new();
    if (name == NULL)
            /* Greska s X509_NAME */

    i = 0;
    field = NULL;
    bytes = NULL;
    field_pos = TRUE;

    /* Izdvajanje dijelova (nedostaju retci ...) */
    while (i < len) {
            if (field_pos && (field == NULL)) {

                    if ((buf[i] == ' ') || ((i == 0) && (buf[i] == '/')))
                            field = &buf[i + 1];
                    else
                            field = &buf[i];
            }
            /* ... */
            if (!field_pos) {
                    if ((buf[i] == ',') || (buf[i] == '/')) {
                            buf[i] = '\0';
                            field_pos = TRUE;
                            X509_NAME_add_entry_by_txt(name, field,
                                        MBSTRING_ASC, bytes, -1, -1, 0)) {


                            field = NULL;
                            bytes = NULL;
                    }
            }
            i++;
    }
    /* Pretvorba stvorene X509_NAME strukture u DER format */

    der_len = i2d_X509_NAME(name, der);
    ...
    X509_NAME_free(name);
```

5.3.3. Dohvaćanje javnog ključa

Primjer dohvaćanja javnog ključa iz X.509 certifikata u internom X509 formatu nalazi se u ispisu Ispis 25: Čitanje javnog ključa iz certifikata u X509 obliku i pretvorba u DER format:

Ispis 25: Čitanje javnog ključa iz certifikata u X509 obliku i pretvorba u DER format

```
X509 *x509;
EVP_PKEY *pubkey;


/* Citanje javnog kljuca iz certifikata */
pubkey = X509_get_pubkey(x509);
```

```
/* Pretvroba EVP_PKEY internog formata u binarni DER format */
i2d_PublicKey(pubkey, public_key);
```

5.3.4. Verifikacija X.509 certifikata

Verifikacija X.509 certifikata podrazumijeva provjeru digitalnog potpisa u certifikata na temelju javnog ključa iz CA certifikata (certifikacijskog centra koji ga je izdao) i CRL-a. Postupak započinje učitavanjem CA certifikata u SSL kontekst (SSL_CTX objekt), Ispis 26: Učitavanje CA certifikata u SSL kontekst:

Ispis 26: Učitavanje CA certifikata u SSL kontekst

```
int SSL_CTX_load_verify_locations(SSL_CTX *ctx, const char *CAfile, const
char *CApath);
```

Kao što je već spomenuto, CA certifikati se mogu nalaziti u zasebnim datotekama (CAfile) ili unutar tzv. Standardnog CA direktorija do kojeg se navodi staza (CApath). Jedan od ta dva argumenta može imati NULL vrijednost.

Slično se obavlja i učitavanje CRL-a u SSL_CTX kontekst (Ispis 27: Učitavanje CRL-a u SSL kontekst):

Ispis 27: Učitavanje CRL-a u SSL kontekst

```
int X509_load_crl_file(SSL_CTX *ctx, const char *file, int type);
```

Najčešće korišteni tip CRL-a je `X509_FILETYPE_PEM`.

Nakon učitavanja konteksta, a prije same verifikacije certifikata, za certifikat (x509 je X509 DER kodiran certifikat) se provjerava sadržaj pojedinih polja certifikata i obavlja ispis greške ili jednostavno ispis polja (npr. izdavač, vlasnik certifikata), Ispis 28: Dohvaćanje pojedinih polja certifikata:

Ispis 28: Dohvaćanje pojedinih polja certifikata

```
    char data[256];
    int depth, err;
    X509 *x509;

        x509 = X509_STORE_CTX_get_current_cert(store);
        depth = X509_STORE_CTX_get_error_depth(store);
        err = X509_STORE_CTX_get_error(store);

        X509_NAME_oneline(X509_get_issuer_name(x509), data, 256);
```

```
        LOG_DEBUG("issuer = %s\n", data);

        X509_NAME_oneline(X509_get_subject_name(x509), data, 256);
        LOG_DEBUG("subject = %s\n", data);
        LOG_DEBUG("err %i:%s\n", err,
                X509_verify_cert_error_string(err));
```

Glavna verifikacijska funkcija naziva se x509_cert_verify koja se poziva nakon što je kontekst u potpunost učitan i stoga je poziv vrlo jednostavan (Ispis 29: Poziv glavna funkcije za verifikaciju certifikata):

Ispis 29: Poziv glavna funkcije za verifikaciju certifikata

```
SSL_CTX *ctx;
X509_verify_cert(ctx);
```

5.3.5. RSA verifikacija tereta AUTH

Osnovu verifikacije AUTH tereta čine funkcije RSA_sign i RSA_verify, budući da se AUTH teret izračunava RSA potpisivanjem okteta IKE SA INIT poruke i kriptografskog materijala. Funkcija RSA_sign izračunava potpis (sigret duljine siglen) potpisuje poruku (m, duljine m_len) pomoću privatnog ključa (rsa) specificarnog u PKCS #1 v2.0 (Ispis 30: Stvaranje digitalnog potpisa):

Ispis 30: Stvaranje digitalnog potpisa

```
int RSA_sign(int type, unsigned char *m, unsigned int m_len, unsigned char
*sigret, unsigned int *siglen, RSA *rsa);
```

Funkcija RSA_verify verificira potpis (sigbuf, duljine siglen) tj. Provjerava da li odgovara izvornoj poruci (m, duljine m_len), Ispis 31: Provjera digitalnog potpisa:

Ispis 31: Provjera digitalnog potpisa

```
int RSA_verify(int type, unsigned char *m, unsigned int m_len,
unsigned char *sigbuf, unsigned int siglen, RSA *rsa);
```

5.3.6. Čitanje ekstenzija iz certifikata

Čitanje ekstenzija iz X.509 certifikata vrlo je važno zbog manipulacije sa subjectAltName ekstenzijom. Pročitana ekstenzija iz certifikata koristi se za popunjavanje identifikacijskog tereta. Primatelj certifikata čita ekstenziju kako bi provjerio jednakost identiteta iz primljenog certifikata i ID tereta (Ispis 28: Dohvaćanje X.509v3 ekstenzije):

Ispis 32: Dohvaćanje X.509v3 ekstenzije

```
char *extstr;
X509_EXTENSION *ext;
ext = X509_get_ext(cert, i);
extstr = OBJ_nid2sn(OBJ_obj2nid(X509_EXTENSION_get_object(ext)));
```

U primjeru je pokazano dohvaćanje jedne ekstenzije – njenog imena (string subjectAltName pohranjen u varijabli ext) te njene vrijednosti (pohranjene u varijabli extstr). U našem primjeru (klijentski certifikat koji se nalazi u dodatku) će u varijabli extstring biti pohranjen niz "moja.domena.com". Funkcija OBJ_obj2nid pretvara pročitanu ekstenziju u NID (*Number Identifier*) oblik kako bi je funkcija OBJ_nid2sn pretvorila iz NID oblika u ASN.1 zapis.

5.4.Knjižnica libcurl

Za dohvaćanje Hash and URL certifikata s definiranog URL-a koristi se libcurl knjižnica. Postupak dohvaćanja certifikata s zadane lokacije i pohrana u memoriju započinje postavljanjem konteksta (učitavanjem URL-a, lokacije za pohranu dohvaćenih podataka i dr.) pomoću funkcije curl_easy_setopt (Ispis 29: Postavljanje libcurl konteksta):

Ispis 33: Postavljanje libcurl konteksta

```
    CURL *curl_handle;
    CURLcode opcode;
    void **ptrptr;
    int retval = 0;

    struct dl_data downloaded;

    LOG_FUNC_START(1);

    downloaded.data = NULL;
    downloaded.size = 0;

    curl_handle = curl_easy_init();
    if (curl_handle == NULL) {
            LOG_ERROR("Error initializing curl handle");
            retval = -1;
            goto out;
   }

    curl_easy_setopt(curl_handle, CURLOPT_URL, url);

    /* define callback function  */
   curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, writecallback_cb);

   /* define pointer for downloaded data */
   curl_easy_setopt(curl_handle, CURLOPT_WRITEDATA, (void *) &downloaded);

   /* some servers don't like requests that are made without a user-agent
      field, so we provide one */
   curl_easy_setopt(curl_handle, CURLOPT_USERAGENT, "libcurl-agent/1.0");
```

Nakon postavljanja konteksta slijedi dohvaćanje podataka (funkcija curl_easy_perform) i čišćenje konteksta (curl_easy_cleanup), Ispis 30: Dohvaćanje podataka s udaljene lokacije i čišćenje libcurl konteksta:

Ispis 34: Dohvaćanje podataka s udaljene lokacije i čišćenje libcurl konteksta

```
opcode = curl_easy_perform(curl_handle);
curl_easy_cleanup(curl_handle);

if (opcode != 0) {      /* if operation failed */
        LOG_ERROR("Error performing libcurl operation");
        retval = -1;
        goto out;
}
```

6. Zaključak

Važnost RSA autentifikacije, temeljene na *X.509 Certificate – Signature* te *Hash and URL X.509 Certificate* tipovima certifikata, kao i prednost pred drugim metodama ovjere leži u praktičnosti i različitim karakteristikama koje pruža infrastruktura javnog ključa (PKI).

Postoji veliki broj scenarija u kojima IKEv2 zahtijeva RSA ovjeru kao nepohodnu metodu. U slučaju proširene ovjere (EAP) inicijatora što je uobičajen slučaj kod tzv. *roadwarrior* scenarija, sugovornik je obavezan koristiti RSA metodu ovjere.

U ovom je radu objašnjena RSA metoda ovjere u IKEv2 protokolu te njena implementacija u ikev2 *daemonu.* Također ovaj rad ističe i propuste u specifikaciji protokola te prijedloge rješenja. Propusti u specifikaciji kao i nedostatak specifikacije pojedinih dijelova protokola (posebice vezanih uz RSA metodu ovjere) trenutno dovode do problema interoperabilnosti. Moguće rješenje je nadopuna temeljne specifikacije IKEv2 protokola ili specifikacija ekstenzija protokola. U ovom je radu predložen prvi način rješavanja problema.

Ipak, drugo navedeno rješenje (nadopuna temeljne specifikacije) je jedino dugoročno rješenje. Postoji nekolicina IETF draftova koji predlažu izbacivanje RSA metode ovjere temeljene na certifikatima u značajnom broju scenarija, budući da je proširena metoda ovjere (EAP) daleko fleksibilnija. Ono što se navodi kao osnovna prednost RSA metode ovjere temeljene na certifikatima je jednostavna razmjena kriptografskog materijala pomoću PKI infrastrukture. No, da li je to zaista prednost? Uvođenje dodatne arhitekture kao što je PKI je skupo i otvara prostor za nove ranjivosti.

Sve veći zahtjevi na karakteristike metode ovjere, nedostatak dokumentacije, ali i sama primjena te korištenje RSA ovjere, nažalost, pokazali su da je jedini razlog njene česte primjene specifikacija koja je metodu (bespotrebno) proglasila obaveznom u slučaju proširene ovjere inicijatora komunikacije.

Literatura

[1] Polk T, Housley R. 2002. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. John Wiley, London.

[2] Chandra P, Messier M, Viega J. 2002. *Network security with OpenSSL*. O'Reilly.

[3] Housley R, Polk T, Ford W, Solo D. 2002. *Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile* (rfc3280).

[4] Telecommunication standardization sector of ITU. 1997. *X.509, Authentication framework*.

[5] Korver B. 2007. *The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX* (draft-ietf-pki4ipsec-ikecert-profile-12).

[6] Eronen P, Tschofenig H. 2007. *Extension for EAP Authentication in IKEv2* (draft-eronen-ipsec-ikev2-eap-auth-05).

[7] Kent S. 2005. *Security architecture for the Internet Protocol* (rfc4301).

[8] Kaufman C. 2005. *Internet Key Exchange (IKEv2) Protocol* (rfc4306).

[9] http://www.openssl.org. *Project OpenSSL*.

Dodatak

ASN.1 građevne jedinice certifikata

.**UTCTime.** UTCTime je standardni ASN.1 tip za zapis datuma i vremena, a upotrebljava se u polju u kojem je zapisan period valjanosti certifikata. Specificira godinu, mjesec, dan, sat, minutu i sekunde, svako s po dvije znamenke, gledajući s lijeva na desno. Na kraju se nalazi znak Z (Zulu, Greenwich Mean Time): YYMMDDHHMMSSZ.

Ukoliko je YY jednako ili veće od 50, godina se interpretira kao 19YY. Ako je YY manji od 50, interpretira se kao 19YY.

GeneralizedTime. GeneralizedTime ASN.1 tip je vrlo sličan UTCTime tipu, no za razliku od njega, za zapis godine koristi četiri znamenke: YYYYMMDDHHMMSSZ. Kao i UTCTime, i ovaj se tip upotrebljava u polju u kojem je zapisan period valjanosti certifikata. Iz tog razloga, UTCTime je moguće upotrijebiti u certifikatima s periodom valjanosti između 1950. i 2049. godine. GeneralizedTime je moguće upotrijebiti za prikaz datuma nakon 2050. godine.

Kao i UTCTime, i GeneralizedTime podrazumijeva preciznost do 1 sekunde, te oba tipa moraju nužno uključivati i sekundu (makar njihova vrijednost bila 0).

Distinguished Name. Distinguished Name je ASN.1 tip koji se upotrebljava u polju vlasnika i izdavača certifikata, a podržava hijerarhijski sustav imena. Potiče iz X.500 preporuke i osmišljena je kao rješenje za jednostavno oblikovanje globalno jedinstvenih imena.

Postoje dva osnovna DN tipa prikaza. Prvi se temelji na X.500 imeničkim atributima, a drugi na imenima iz DNS-a. Najpoznatiji X.500 imenički atributi su država (c=), organizacija (o=), organizacijska jedinica (ou=), lokalitet (l=) i *common name* (cn=). Na primjer:

c=Hr, o=FER, ou=zemris, l=Zagreb, cn=Ana Kukec

Na Internetu je mnogo praktičnije upotrebljavati DN temeljen na DNS imenima. U tu svrhu, IETF:P je definirao domensku komponentu (domain component, dc):

dc=hr, dc=fer, dc=zemris; cn=Ana Kukec

dc=hr, dc=fer, dc=zemris; uid=Ana Kukec

Za zapis identiteta osobe ili grupe prepopručeno je koristiti cn (common name) ili uid (user identifier). U gornjem primjeru identitet osobe je: Ana Kukec, iz domene zemris.fer.hr.

General Name. General Name, kao i Distinguished Name, služi za zapis identiteta, no on se ne temelji na hijerarhijskom imeničkom sustavu i služi za pohranu većeg broja različitih tipova imena. Unutar GeneralName polja moguće je specificati gotovo sva imena koja su od nekakve koristi na Internetu (npr. E-mail adresa, X.400, ime kućnog ljubimca, broj telefona).

CA certifikat, klijentski certifikat i CSR

CA certifikat:

```
Certificate:
    Data:
```

```
        Version: 3 (0x2)
        Serial Number:
            d4:78:f8:de:f3:c0:44:f7
        Signature Algorithm: md5WithRSAEncryption
        Issuer: CN=imunes-server, C=HR/emailAddress=rootCA@zemris, O=rootCA
        Validity
            Not Before: Mar 13 09:02:21 2007 GMT
            Not After : Jul 28 09:02:21 2034 GMT
        Subject: CN=imunes-server, C=HR/emailAddress=rootCA@zemris, O=rootCA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:bd:7a:e0:c3:6f:f3:ac:21:ce:0b:d8:af:31:43:
                    8c:6e:f2:b3:ce:fb:46:6c:32:ee:74:1b:61:9c:cb:
                    03:0d:10:47:4e:40:e6:51:c4:06:ae:8c:04:f1:a8:
                    a1:22:00:89:df:e9:1d:f0:cd:71:7d:84:24:cb:f6:
                    c9:d8:6a:48:96:b5:cb:2a:c4:09:30:9b:c5:ac:2f:
                    44:76:bb:fd:d9:2f:e6:99:e6:08:c9:5a:9f:45:35:
                    ab:92:95:c0:4a:cb:3e:5e:c4:56:c5:29:3e:13:fa:
                    1b:17:b0:12:07:3c:07:17:69:b3:2c:1b:c3:9a:5c:
                    4e:72:e2:11:21:c0:28:bf:69
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
    Signature Algorithm: md5WithRSAEncryption
        10:80:b5:2b:50:55:14:0c:a7:6b:ed:d5:1e:be:8c:f4:dd:29:
        4e:01:31:ae:a6:80:21:85:ca:7e:82:ea:08:e1:bd:61:b0:d0:
        df:57:97:df:5c:15:53:14:46:e5:bb:ee:18:23:34:4d:75:d8:
        a8:c4:75:77:4c:51:76:02:27:10:24:08:2d:f7:5c:64:8b:cd:
        51:0d:6a:09:8a:1d:d9:3a:d4:5e:d0:53:0b:a9:cf:91:54:94:
        5d:e3:6e:15:ad:4f:28:74:44:96:2b:68:b9:69:a8:05:de:76:
        5e:b6:81:a9:db:5f:ae:49:4e:27:68:d9:f1:4f:51:60:9f:a6:
        32:ff
```

Klijentski certifikat:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
```

```
        Signature Algorithm: md5WithRSAEncryption
        Issuer: CN=rootCA-common-name, C=HR/emailAddress=rootCA@zemris, O=rootCA
        Validity
            Not Before: Jan 24 21:51:35 2007 GMT
            Not After : Jan 24 21:51:35 2008 GMT
        Subject: CN=Ana, ST=Croatia, C=HR/emailAddress=anchie@esa.fer.hr, O=FER
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:ca:56:84:ed:b2:71:38:aa:07:1b:18:2c:70:11:
                    d2:4c:05:58:25:32:03:13:5a:04:de:19:96:6c:40:
                    ec:a7:92:3b:e1:7f:89:f7:b7:aa:3a:54:da:34:7e:
                    d8:54:c5:c8:90:fc:72:91:ee:1b:a0:84:14:26:9a:
                    c8:64:ad:32:22:2b:18:41:6b:2a:32:5a:9d:1e:fe:
                    09:07:26:5f:c4:db:27:fc:d7:4b:67:f1:c3:56:80:
                    ca:26:4a:ed:4e:e9:6b:94:94:2c:04:e7:42:c6:cb:
                    1b:89:0c:b2:3b:fc:ec:70:3d:d4:97:cf:a1:02:fa:
                    20:1c:d6:f5:e0:6b:d2:b6:c5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Alternative Name:
                DNS:moja.domena.com
    Signature Algorithm: md5WithRSAEncryption
        c2:7c:ec:f9:6b:c1:b2:01:a4:9a:42:30:44:79:a5:a3:e4:b3:
        af:93:01:3b:69:e4:64:00:a9:b2:6c:0c:ff:03:1e:1e:62:2b:
        80:89:91:ba:0d:a4:96:c4:c3:d7:17:5f:e8:50:08:8a:5c:f9:
        e3:2d:15:44:e8:03:21:81:84:39:55:be:8f:9b:05:1a:79:66:
        bc:21:da:cf:5d:6f:cd:81:e0:6b:83:58:e5:7e:0b:84:74:46:
        4a:88:05:e7:c0:c2:9b:97:ff:58:46:47:77:44:81:3d:b8:c2:
        ab:62:66:7b:21:5b:21:f4:31:bd:bb:23:a7:49:58:70:9f:d2:
        fa:75
```

Glavna funkcija za verifikaciju X.509 certifikata

```
/**
 * Perform verification of a single certificate with given context.
 *
 * @param cert_path     Properly set standard CA directory with CA
 *              certificates named i.e. 1234c307.0
```

```
 * @param ca_file       File with the CA certificate(s) in PEM format.
 * @param cert          Pointer to certificate data.
 * @param cert_len      Length of the certificate_data.
 * @param crl_file      File with the CRL.
 *
 * \return 0 if certificate verified succesfully, 0 otherwise
 */
int cert_verify(const char *ca_file, unsigned char *cert,
        guint16 cert_len, const char *crl_file)
{
    X509 *x509;
    X509_LOOKUP *lookup = NULL;
    X509_STORE *store = NULL;
    X509_STORE_CTX *ctx;
    int ret = 1;
    const char *ca_dir = NULL;
    struct timeb time_start, time_end;

    LOG_FUNC_START(1);

    if ((cert == NULL) || !cert_len)
        goto out;

    /**
     * Transform binary certificate data into X509 structure
     */
    x509 = d2i_X509(NULL, &cert, cert_len);
    if (!x509) {
        LOG_ERROR("Certificate error");
        goto out;
    }

    OpenSSL_add_all_algorithms();

    store = X509_STORE_new();
    X509_STORE_set_verify_cb_func(store, verify_callback);

    /**
     * Load the CA certificates and CRLs.
```

```
     */
    if (X509_STORE_load_locations(store, ca_file, ca_dir) != 1) {
        /* Log error: Error loading the CA file or directory. */
    }
    if (X509_STORE_set_default_paths(store) != 1) {
        /* Log error: Error loading the system-wide CA certificates. */
    }
    if (!(lookup = X509_STORE_add_lookup(store, X509_LOOKUP_file()))) {
        /* Log error: Error creating X509 lookup object. */
    }
    /**
     * CRL verification is possible only in OpenSSL v >=0.9.7
     */
#if (OPENSSL_VERSION_NUMBER > 0x00907000L)
    /* set the flags of the store so that CRLs are consulted */
    X509_STORE_set_flags(store, X509_V_FLAG_CRL_CHECK |
            X509_V_FLAG_CRL_CHECK_ALL);
#endif

    /* create a verification context and initialize it */
    if (!(ctx = X509_STORE_CTX_new())) {
        /* Log error: Error creating X509_STORE_CTX object") */
    }
#if (OPENSSL_VERSION_NUMBER > 0x00907000L)
    if (X509_STORE_CTX_init(ctx, store, x509, NULL) != 1) {
        /* Log error: Error initializing verification context. */
    }
#else
    X509_STORE_CTX_init(ctx, store, x509, NULL);
#endif

    /**
     * Verify the certificate. If certificated has been verified correctly
     * the ret is 0.
     */
    if (X509_verify_cert(ctx) != 1) {
        /* Log the event of unsuccessfull certificate verifing. */
        ret = 1;
        LOG_ERROR("Error verifying the certificate.");
```

```
        goto out;
    }

    X509_STORE_CTX_cleanup(ctx);

    LOG_DEBUG("Certificate verified successfully!");
    ret = 0;
    return ret;
}
```

Konfiguracijska datoteka za ikev2 (ikev2.conf)

Konfiguracijska datoteka ikev2 *daemona* (ikev2.conf):

```
#
# This is a configuration file for IKEv2 daemon and as such it is a
# combination of SAD, SPD and PAD. For details of those databases see
# RFC2401bis.
#

#
# This is a configuration file for IKEv2 daemon and as such it is a
# combination of SAD, SPD and PAD. For details of those databases see
# RFC2401bis.
#
general {
    rand_device "/dev/urandom";

    psk_file "psk.txt";

    # Number of half-opened connections above which we start to send
    # cookies
    dos_treshold 50;

    # Number of concurent threads for processing sessions
    sm_threads 5;

    # How do we behave, as initiator, responder or both? Default is
    # both. Not implemented for now...
```

```
    # mode responder|initiator|unspec;
    mode initiator;

    # Enable this if you want sanity check of configuration file after
    # parsing is over...
    #
    # Default is off
    #
    # Not implemented for now...
    #sanity_check off|warning|error;
    sanity_check off;

    # How many fully established IKE SAs from the same IP address will
    # IKE daemon allow before ignoring new requests...
    ikesa_max 50;

    # How many half opened IKE SAs from the same IP address before
    # IKE daemon will stop accepting new IKE SA INIT requests
    ikesa_max_halfopened 50;

    ca type_x509 "cacert.pem";
    cert type_x509 "cert.pem" "testkey.pem";
}
logging {

    # This configuration directive requests that all log info
    # from subsystem <subsystem> with level lower that or
    # equal to <loglevel> be written to some file.
    #
    # Subsystems are crypto, message, sm, payload, network, ...
    #
    # The following block can be used multiple times.
    file "logfile" {
        # The following configuration directive determines if
        # time is prepended to each log entry
        log timestamp, line, function, severity, subsystem;
        level trace;
        subsystem main, aes_xcbc, config, crypto, csa, message,
        payload, network, pfkey, proposals, session,
```

```
            netlib, sockaddr, sm, transform, ts, parser,
            radius, supplicant, timeout, cfg, cert;
    }

    # This configuration directive requests that all log info
    # from subsystem <subsystem> with level lower that or
    # equal to <loglevel> be written to syslog.
    #
    # This directive could only be used once!
    #syslog daemon {
    #   level warn;
    #   subsystem <subsytem>;
    #}

    # This configuration directive requests that all log info
    # from subsystem <subsystem> with level lower that or
    # equal to <loglevel> be written to stderr.
    #
    # The following block can be used multiple times.
    #stderr {
        # The following configuration directive determines if
        # time is prepended to each log entry
    #   log timestamp, line, function, severity, subsystem;
    #   level trace;
    #   subsystem main, aes_xcbc, config, crypto, csa, message,
    #       payload, network, pfkey, proposals, session,
    #       netlib, sockaddr, sm, transform, ts, parser,
    #       radius, supplicant, timeout, cfg, auth, cert;
    #}
}
radius_server id1 {
    nas_identifier pero.zdero;

    timeout 15 s;
    retries 3;

    auth_server {
        address 161.53.65.11;
        shared_secret test;
```

```
    }

    auth_server {
        address 192.168.0.1;
        shared_secret test1;
    }

    acct_server {
        address 161.53.65.12;
        shared_secret test;
    }

    acct_server {
        address 192.168.0.2;
        shared_secret test1;
    }

}

cfg {

    module_path "/usr/local/lib64/ikev2";

    provider dhcp1 {
        module "dhcp.so";
        timeout 32;
        retry 2;
    }

    provider file1 {
        module "file.so";
        file "file.dat";
        file1 file1.dat;
    }
}
remote any
{
    nonce_size 32;
```

```
    proposal {
        encryption_algorithm 3des;
        auth_algorithm md5_96;
        pseudo_random_function md5;
        dh_group modp768;
    }

    proposal {
        encryption_algorithm aes128_cbc;
        auth_algorithm md5_96;
        pseudo_random_function md5;
        dh_group modp1024;
    }
}
peer any {

    ca type_x509 "cacert.pem";

    #ca type_x509 "cacert.pem"
    #   crl "http://www.url.com/cacrl.crl";

    #cert type_x509 "rrkey1.pem" "rrcert1.pem";
    #cert type_x509 "rrkey1.pem" "rrcert2.pem"
    #       "http://ikev2certs.example.com/certificate.cer";

    #psk_file "./psk.txt";
    #wpa_conf "./wpa_supplicant1.conf";

    cfg {

        provider_id file1;

        script_up "./ifup.sh";
        script_down "./ifdown.sh";
    }

    auth_method pre_shared_key;

    peer_auth_method rsa_sig;
```

```
    authlimit {
        time 18 days;
        allocs 128;
        octets 1 MB;
    }

    rekeylimit {
        time 18 days;
        allocs 128;
        octets 1 MB;
    }

    ike_max_idle 1 hour;

    my_identifier rfc822_addr ana.kukec@fer.hr;

    sainfo local 192.168.247.129 remote 192.168.247.134 {
        auth_algorithm sha1_96;
        encryption_algorithm 3des;
    }
}
```