SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1449

SIGURNOST U BEŽIČNIM LOKALNIM MREŽAMA

Valentin Vidić

Zagreb, lipanj 2004.

Zahvaljujem se svom mentoru
prof. dr. sc. Leu Budinu
i
doc. dr. sc. Marinu Golubu na
stručnom vodstvu

SADRŽAJ

1. Uvod

Posljednjih godina svjedoci smo naglog širenja tržišta mobilnih usluga i uređaja. Najsvjetliji primjer zasigurno je tržište mobilne telefonije o čijoj rasprostranjenosti najbolje govori činjenica da se danas teško može pronaći osoba koja ne posjeduje mobilni telefon. Pojava ručnih računala (engl. *handheld*) i pad cijene prijenosnih računala obećava da bi se isto moglo dogoditi i na tržištu mobilne računalne opreme. Dok su mobilni telefoni donijeli rasprostranjenu i standardiziranu mrežnu infrastrukturu, bežična mrežna infrastruktura za računala pojavila se relativno nedavno. IEEE je 1999. godine objavio standard za bežične lokalne mreže pod oznakom 802.11. Standard opisuje protokol komunikacije i izgled okvira dok su detalji fizičkog sloja opisani u dodatcima standardu pod nazivom 802.11a, 802.11b itd. Problem koji standardom nije kvalitetno riješen upravo je pitanje sigurnosti.

Danas, pet godina kasnije, bežična mrežna oprema postala je pristupačna cijenom i rasprostranjena. No ipak, mnoge organizacije još se nisu odlučile na korištenje nove tehnologije. Razlog tomu su sigurnosni problemi koji bežične računalne mreže prate od njihove pojave. Bežični prijenos podataka inherentno je nesigurniji zbog prirode medija koji se koristi. Signal u klasičnim računalnim mrežama koristi žičani medij do kojega je pristup ograničen mjerama fizičke sigurnosti. Elektromagnetski val, međutim, širi se u svim smjerovima i izvan područja pokrivenog mjerama fizičke sigurnosti. Zbog toga je sigurnost u bežičnim mrežama potrebno ostvariti na razini mrežnog protokola. Dva osnovna problema koja je potrebno riješiti su: autentifikacija korisnika i prijenos podataka zaštićen od prisluškivanja i izmjena. Standard 802.11 sadrži prve pokušaje rješavanja tih problema temeljene na RC4 algoritmu.

Nedugo nakon izlaska 802.11 standarda kriptografska zajednica izvršila je niz analiza koje su otkrile različite nedostatke u korištenim protokolima. Između ostalog otkriveni su propusti u korištenom protokolu autentifikacije, ali sve do 2001. vjerovalo se da je RC4 kriptografski siguran. Te godine objavljen je članak[1] sa opisom propusta u RC4 algoritmu koji omogućava otkrivanje ključa kriptiranja iz kriptiranih podataka. Vrlo brzo su se pojavile i prve implementacije koje su pokazala da je ključ kriptiranja moguće otkriti sakupljanjem nekoliko milijuna okvira podataka. Kao posljedica tih događaja započinje rad na novim specifikacijama koje bi trebale riješiti uočene propuste. Sigurna autentifikacija korisnika postala je moguća pojavom IEEE 802.1X standarda. U njemu je opisan protokol autentifikacije korisnika temeljen na proširivom autentifikacijskom protokolu (engl. *EAP – Extensible Authentification Protocol*). Autentifikacija korisnika u početku se obavljala pomoću certifikata, a u posljednje vrijeme i pomoću lozinki.

Korištenjem proširivog autentifikacijskog protokola ublažen je i problem nesigurnosti RC4 algoritma. Naime, kao nusprodukt uspješne autentifikacije automatski se generiraju novi ključevi kriptiranja. Osim što je time svaki korisnik dobio različit ključ kriptiranja, postalo je moguće generirati nove ključeve periodičkim ponavljanjem autentifikacije. Novi ključ kriptiranja potrebno je generirati prije nego napadač skupi dovoljno okvira za izvođenje

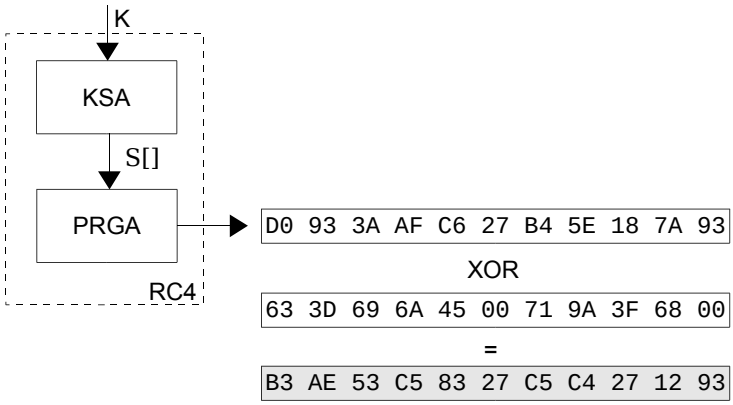
napada. Pitanje enkripcijskog algoritma trebalo bi biti trajno riješeno pojavom IEEE 802.11i specifikacije. Ona opisuje dva nova protokola kriptiranja: TKIP i CCMP. Prvi se i dalje temelji na RC4 algoritmu i treba poslužiti samo u prijelaznom razdoblju, dok bi CCMP temeljen na AES algoritmu trebao postati trajno rješenje.

Ovaj diplomski rad, u prvome dijelu, sadrži opise svih navedenih algoritama i protokola, počevši od RC4 algoritma, pa sve do CCMP protokola. Drugi dio diplomskog rada uključuje praktični rad vezan uz probijanje RC4 algoritma. Poglavlje 3. sadrži detaljni opis propusta u RC4 algoritmu koji omogućava otkrivanje ključa kriptiranja. Programska izvedba tog napada opisana je u poglavlju 4. Poglavlje 5. sadrži rezultate mjerenja u kojemu su uspoređene performanse sa postojećim programom iste namjene. Analiza dobivenih rezultata provedena je u 6. poglavlju.

2. Algoritmi i protokoli

2.1. Algoritam enkripcije RC4

RC4 algoritam bio je i do danas ostao temelj za ostvarivanje sigurnosti u bežičnim 802.11 lokalnim mrežama. Većina opreme dostupne na tržištu sklopovski podržava ovaj algoritam i zbog razloga kompatibilnosti on će se vjerojatno nastaviti koristiti i u bližoj budućnosti (iako se počinju pojavljivati proizvodi koji podržavaju i druge algoritme, prije svega AES). RC4 algoritam [2] razvio je Ron Rivest 1987. godine za RSA Data Security, Inc. Algoritam je bio tajan sve do 1994. godine kada je izvorni kôd algoritma anonimno objavljen na Internetu.

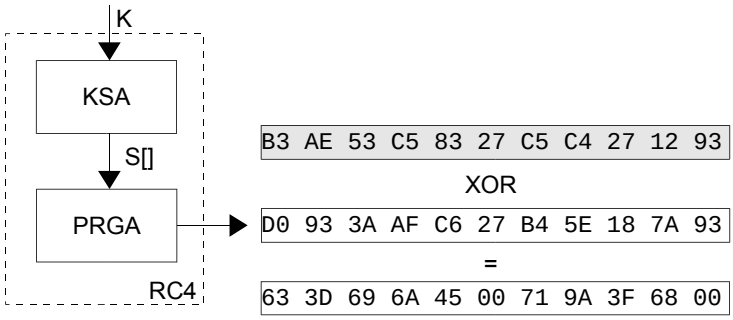


Slika 2.1. RC4 enkripcija

RC4 je simetrični znakovni (engl. *stream*) algoritam – svaka riječ kriptira i dekriptira se zasebno. Najčešće se koristi riječ dužine jedan bajt. Ključ koji se pri tome koristi može biti dužine do 256 bajtova. Algoritam generira niz pseudoslučajnih brojeva proizvoljne duljine (slika 2.1.). XOR miješanjem tog niza s podacima dobiva se kriptirani niz. Na prijemnoj strani generira se isti niz pseudoslučajnih brojeva (prijemna strana koristi isti ključ) i XOR miješa sa kriptiranim nizom. Zbog svojstva

$$(p \text{ XOR } z) \text{ XOR } z = p \qquad (1)$$

prijemna strana nakon toga dobiva izvorne podatke (slika 2.2.).



Slika 2.2. RC4 dekripcija

Preostaje još objasniti kako se generira niz pseudoslučajnih brojeva. Generiranje niza izvodi se u dva koraka[1]. Prvi korak (engl. *KSA – Key Scheduling Algorithm*) inicijalizira S-kutiju. U drugom koraku (engl. *PRGA – Pseudo Random Generation Algorithm*) korištenjem inicijalizirane S-kutije generira se niz pseudoslučajnih brojeva. Inicijalizacija S-kutije je parametrizirana ključem enkripcije prema slijedećem algoritmu:

```
KSA(K):
   Inicijalizacija:
      for i = 0 ... N-1
         S[i] = i
      j = 0

   Miješanje:
      for i = 0 ... N-1
         j = j + S[i] + K[i mod l]
         zamijeni(S[i], S[j])
```

U prvom dijelu KSA algoritma S-kutija (koji predstavlja permutaciju niza 0 ... N-1)[1] se postavlja u početno stanje – identitet - S-kutija preslikava svaki broj u samoga sebe. U drugom dijelu KSA algoritma korištenjem dva brojača (*i,j*) obavlja se miješanje. Brojač *i* se pomiče linearno, dok se *j* mijenja pseudoslučajno (zbog *S[i]*) i ovisno o ključu (zbog *K[i mod l]*). U svakom koraku zamjenjuju se elementi S-kutije na koje pokazuju brojači. Važno je napomenuti da se sve operacije zbrajanja obavljaju modulo *N*. Ovako inicijalizirana S-kutija koristi se za generiranje pseudoslučajnog niza u PRGA algoritmu:

```
PRGA(S):
   Inicijalizacija:
      i = 0
      j = 0

   Generiranje pseudoslučajnog niza:
      i = i + 1
      j = j + S[i]
      zamijeni(S[i], S[j])
      generiraj z = S[S[i] + S[j]]
```

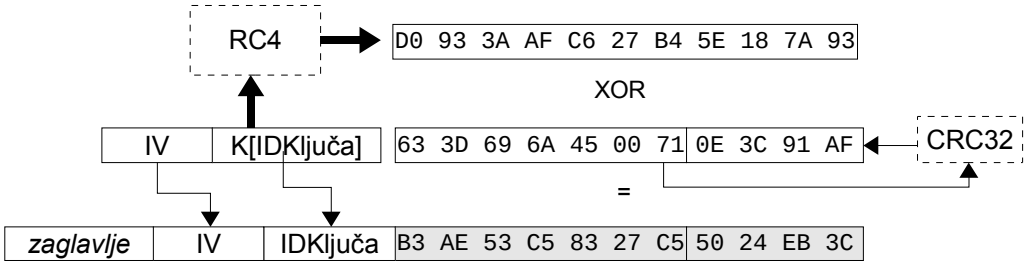
Prvi dio PRGA algoritma inicijalizira brojače na nulu. S-kutija je inicijalizirana u prethodnom koraku (*KSA*). Nakon toga brojači se povećavaju: *i* linearno, *j* pseudoslučajno. Vrijednosti u S-kutiji na koje pokazuju se zamjenjuju i generira se jedna riječ pseudoslučajnog niza (*z*). Tijekom generiranja pseudoslučajnog niza S-kutija se zbog zamjene elemenata i dalje polako mijenja. Brojači *i* i *j*, te S-kutija predstavljaju stanje algoritma. Različitih stanja ima $N!N^2$ (broj različitih permutacija S-kutije * broj različitih vrijednost brojača *i* * broj različitih vrijednosti brojača *j*). Za *n=8* (tj. *N=256*) postoji oko 2^{1700} različitih stanja. Zbog toga je otkrivanje stanja algoritma korištenjem grube sile neisplativo.

2.2. Protokol enkripcije WEP

801.11 standard[3] za bežične lokalne mreže definira WEP (engl. *WEP - Wired Equivalent Privacy*) kao algoritam za zaštitu mrežnog prometa od prisluškivanja. Ime algoritma sugerira da bi u bežičnoj mreži trebao pružiti

1 $N=2^n$, gdje je n broj bita u riječi (za n=8, N=256)

sigurnost ekvivalentnu sigurnosti u lokalnim mrežama. Sigurnost lokalnih mreža temelji se prije svega na fizičkoj sigurnosti prostora u kojem se mreža nalazi. Bežične lokalne mreže ne mogu se fizički zaštititi jer je širenje signala teško ograničiti. Iako bi se prema tome moglo zaključiti da WEP postiže definirani cilj i onemogućava slučajnom prolazniku korištenje i prisluškivanje mreže (slično kao zaključana vrata prostorije) jedno je sigurno: WEP ne predstavlja cjelovito i kvalitetno rješenje sigurnosti u bežičnim mrežama.

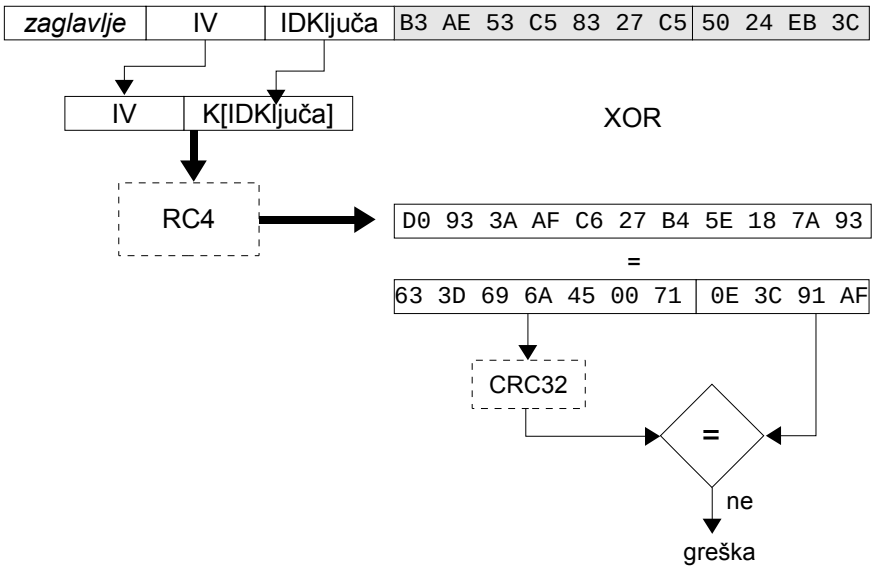


Slika 2.3. WEP enkripcija okvira

WEP kao osnovu koristi RC4 algoritam. Ključ kojim se inicijalizira RC4 algoritam sastoji se od dva dijela: inicijalizacijskog vektora (engl. *IV – Initialization Vector*) i tajnog ključa (slika 2.3.). Tajni ključ (često nazivan WEP ključ) poznat je svim ovlaštenim korisnicima mreže i jednak je za sve korisnike. Distribucija tajnog ključa nije definirana standardom i najčešće se obavlja "ručno"[2]. Inicijalizacijski vektor obično je različit za svaki okvir, dok je tajni ključ stalan. Posljedica toga je da za okvire sa različitim IV-om RC4 generira različite nizove pseudoslučajnih brojeva. IV je dužine 3 bajta te se nakon nekog vremena nužno mora dogoditi ponavljanje. Da bi prijemna strana mogla dekriptirati sadržaj okvira, IV se šalje nekriptiran u zaglavlju okvira. Radi zaštite okvira od izmjena, za sadržaj okvira računa se CRC32 suma (engl. *ICV – Integrity Check Value*) i kriptira zajedno sa podacima. Prijemna strana nakon dekriptiranja računa ICV za sadržaj okvira i uspoređuje ga sa ICV-om dobivenim nakon dekripcije okvira (slika 2.4.).

Zaglavlje okvira sadrži i identifikator korištenog tajnog ključa. Identifikator je dužine 2 bita pa je moguće istovremeno koristiti 4 različita tajna ključa. Ukoliko se u mreži koristi više ključeva bitno je da na svim uređajima isti tajni ključevi budu pridruženi istim identifikatorima. Ako, na primjer, uređaj A kriptira okvir tajnim ključem s identifikatorom 2, tada uređaj B mora imati isti ključ pridružen identifikatoru 2 da bi mogao ispravno dekriptirati sadržaj okvira. Prema tome, identifikatori tajnog ključa predstavljaju indekse polja tajnih ključeva. Standard 802.11 opisuje i korištenje različitih WEP ključeva za različite MAC adrese. Ta mogućnost obično nije dostupna preko uobičajenog Web sučelja pristupne točke (eventualno samo preko SNMP-a), pa se rijetko koristi. WEP ključevi mogu biti dugi 40 bita (dodavanjem IV-a dobiva se 64 bitni RC4 ključ) ili 104 bita (128 bitni RC4 ključ).

2 Upisivanjem vrijednosti ključa prilikom konfiguriranja postavki mrežne kartice



Slika 2.4. WEP dekripcija okvira

2.3. Autentifikacija

802.11 standard upisuje dva načina autentifikacije:

- otvoreni sustav (engl. *Open system*)
- korištenjem dijeljene tajne (engl. *Shared key*)

2.3.1. Otvoreni sustav

Otvorenom sustavu može pristupiti svatko, pa je autentifikacija samo formalna. Identitet korisnika sustava se ne provjerava već je svima dopušteno korištenje sustava. Autentifikacija se u otvorenom sustavu izvodi u dva koraka. U prvom koraku stanica koja se autentificira (klijent) generira okvir sadržaja prikazanog u tablici 2.1.

Tablica 2.1. Otvoreni sustav: prvi okvir

Element zaglavlja	***Vrijednost***
Tip okvira	Upravljački
Podtip okvira	Autentifikacija
Tip autentifikacije	Otvoreni sustav
Identitet stanice	MAC adresa klijenta
Redni broj okvira u transakciji autentifikacije	1

Stanica koja obavlja autentifikaciju na to odgovara okvirom sadržaja prikazanog u tablici 2.2. Autentifikacija može završiti uspjehom ili neuspjehom. U slučaju uspjeha polje rezultata drugog okvira sadrži vrijednost 0. U slučaju neuspjeha polje rezultata sadrži vrijednost koja opisuje razlog neuspjeha. Autentifikacija završava neuspjehom ako stanica ne podržava algoritam autentifikacije (vrijednost 13), ako je istekao

vremenski period za autentifikaciju (vrijednost 16), ako je na pristupnu točku (engl. *AP – Access Point*) već spojen maksimalan broj klijenata (vrijednost 17) itd. Dodatak A. sadrži primjer okvira autentifikacije u otvorenom sustavu.

Tablica 2.2. Otvoreni sustav: drugi okvir

Element zaglavlja	***Vrijednost***
Tip okvira	Upravljački
Podtip okvira	Autentifikacija
Tip autentifikacije	Otvoreni sustav
Redni broj okvira u transakciji autentifikacije	2
Rezultat	0

2.3.2. Autentifikacija dijeljenom tajnom

Autentifikacija dijeljenom tajnom pretpostavlja da samo autorizirani korisnici poznaju dijeljenu tajnu. Dijeljena tajna je WEP ključ koji se koristi za enkripciju okvira s podatcima. Autentifikacija pomoću dijeljene tajne izvodi se u četiri koraka. Stanica koja pristupa mreži u prvom okviru šalje informacije prikazane u tablici 2.3.

Tablica 2.3. Dijeljena tajna: prvi okvir

Element zaglavlja	***Vrijednost***
Tip okvira	Upravljački
Podtip okvira	Autentifikacija
Tip autentifikacije	Dijeljena tajna
Identitet stanice	MAC adresa klijenta
Redni broj okvira u transakciji autentifikacije	1

Pristupna točka odgovara okvirom prikazanim u tablici 2.4.

Tablica 2.4. Dijeljena tajna: drugi okvir

Element zaglavlja	***Vrijednost***
Tip okvira	Upravljački
Podtip okvira	Autentifikacija
Tip autentifikacije	Dijeljena tajna
Redni broj okvira u transakciji autentifikacije	2
Rezultat	0
Dodatno polje	128 bajtova slučajno generiranih podataka

Ukoliko polje rezultata sadrži kôd uspjeha (vrijednost 0) okvir sadrži dodatno polje (engl. *Challenge text*) duljine 128 bajtova. Ovo polje predstavlja niz slučajno generiranih brojeva (može se koristiti pseudoslučani niz generiran RC4 algoritmom). U slučaju da polje rezultata sadrži vrijednost različitu od 0, autentifikacija završava neuspjehom. U tom slučaju vrijednost polja rezultata opisuje razlog neuspjeha (isto kao kod autentifikacije u otvorenom sustavu).

Ukoliko autentifikacija u prethodnom koraku nije završila neuspjehom, stanica koja se autentificira odgovara WEP kriptiranim okvirom čiji je sadržaj prikazanim u tablici 2.5.

Tablica 2.5. Dijeljena tajna: treći okvir

Element zaglavlja	***Vrijednost***
Tip okvira	Upravljački
Podtip okvira	Autentifikacija
Tip autentifikacije	Dijeljena tajna
Redni broj okvira u transakciji autentifikacije	3
Dodatno polje	128 bajtova podataka iz okvira 2.

Okvir se kriptira pomoću WEP ključa (dijeljena tajna). Stanica koja obavlja autentifikaciju nakon primitka gornjeg okvira dekriptira tijelo okvira. Ukoliko je dekripcija uspješna (ICV je ispravan) i niz bajtova dobiven dekripcijom identičan nizu poslanom u okviru s rednim brojem 2, stanica koja pristupa mreži zna dijeljenu tajnu i treba joj dozvoliti pristup. U suprotnom, autentifikacija je neuspješna i stanici se ne dozvoljava pristup. Posljednji okvir šalje stanica koja obavlja autentifikaciju (tablica 2.6.) Opis vrijednosti koje može poprimiti polje rezultata jednak je kao za okvir s rednim brojem 2. Dodatak A. sadrži primjer autentifikacije dijeljenom tajnom.

Tablica 2.6. Dijeljena tajna: četvrti okvir

Element zaglavlja	***Vrijednost***
Tip okvira	Upravljački
Podtip okvira	Autentifikacija
Tip autentifikacije	Dijeljena tajna
Redni broj okvira u transakciji autentifikacije	4
Rezultat	0

Na žalost, autentifikacija dijeljenom tajnom ne ograničava pristup mreži samo na korisnike koji poznaju dijeljenu tajnu. Prisluškivanjem autentifikacije ovlaštenog korisnika napadač može otkriti dovoljno informacija da i sam pristupi mreži. Napadač može odrediti korišteni RC4 pseudoslučajni niz XOR miješanjem nekriptiranog (slučajni niz iz drugog okvira) i kriptiranog sadržaja trećeg okvira. Korištenjem pseudoslučajnog niza zatim može pristupiti mreži

na isti način kao i ovlašteni korisnik. Pristupna točka će mu u drugom koraku poslati novi slučajni niz brojeva, ali napadač može ispravno kriptirati treći okvir XOR miješanjem slučajnog niza brojeva sa otkrivenim RC4 pseudoslučajnim nizom. Nakon uspješne autentifikacije napadač može korištenjem RC4 pseudoslučajnog niza generirati male podatkovne okvire u mrežu (ograničen je veličinom otkrivenog RC4 pseudoslučajnog niza – oko 128 bajtova). Primanje okvira nije moguće jer napadač ne zna WEP ključ i mala je vjerojatnost da će primljeni okviri koristiti upravo onaj inicijalizacijski vektor za koji je otkriven pripadni RC4 pseudoslučajni niz. Iz ovog opisa može se zaključiti da niti jedna od početno definiranih metoda autentifikacije ne omogućava sigurnu autentifikaciju korisnika.

2.4. Povezivanje

Nakon procesa autentifikacije slijedi proces povezivanja (engl. *association*). Smisao povezivanja je registriranje položaja stanice. Ukoliko bežičnu mrežu čini više pristupnih točaka koje signalom pokrivaju različito područje, stanica se povezuje na pristupnu točku u čijem se području signala nalazi. Pristupne točke međusobno izmjenjuju informacije o spojenim stanicama. Ukoliko je potrebno poslati okvir određenoj stanici, zbog povezivanja, točno se zna kojoj pristupnoj točki dotični okvir treba proslijediti. Stanica može biti autentificirana na više pristupnih točaka, ali može biti povezana na samo jednu. Protokol povezivanja sastoji se od svega dva okvira: zahtjeva i odgovora (engl. *Association request/response*).

2.5. Autentifikacija MAC adresa

Iako standardom nije definirana, mnogi proizvođači nude mogućnost autentificiranja klijenata prema MAC adresi. Pristupna točka može sadržavati popis MAC adresa kojima je dozvoljen/zabranjen pristup ili za to koristiti autentifikacijski (obično RADIUS) server. MAC autentifikacija se može provoditi nakon primitka prvog okvira u procesu autentifikacije[3]. Ukoliko se MAC adresa ne nalazi u skupu dozvoljenih adresa, polje rezultata u drugom okviru procesa autentifikacije sadržavati će kôd neuspjeha. Cisco pristupne točke koriste drugačiju metodu. One dozvoljavaju autentifikaciju i asocijaciju svima, ali nakon tog propuštaju samo promet sa ovlaštenih MAC adresa. Na žalost, MAC autentifikacija nije efikasna protiv strpljivog napadača. Napadač može prisluškivati promet u mreži i otkriti kojim MAC adresama je dozvoljen pristup. Nakon što neki od autoriziranih korisnika prestane koristiti mrežu, napadač može promijeniti MAC adresu svoje kartice na adresu autoriziranog korisnika i pristupiti mreži. Promjena MAC adrese na Linux-u nije puno kompliciranija od postavljanja IP adrese sučelja:

```
# ifconfig wlan0 hw ether 00:09:69:12:5E:71
```

2.6. Proširivi autentifikacijski protokol

Sigurnija autentifikacija u bežičnim mrežama prvi puta je postala moguća izlaskom 802.1X specifikacije[4]. 802.1X standard opisuje korištenje proširivog autentifikacijskog protokola (engl. *EAP - Extensible Authentication*

3 Hostapd (AP softver za Linux) koristi tu metodu

Protocol) u IEEE lokalnim mrežama, pa tako i u 802.11 mrežama. EAP je proširivi protokol autentifikacije standardiziran od IETF-a[5], u početku namijenjen za autentifikaciju korisnika preko modemskih PPP veza. Osnovna ideja kod razvoja EAP-a bilo je izdvajanje autentifikacije u zaseban protokol, tako da dodavanje novih metoda autentifikacije više ne zahtjeva izmjene PPP standarda. Posljedica toga je EAP koji omogućava korištenje različitih metoda autentifikacije preko različitih medija. Tako ga je moguće, osim na PPP vezama i 802.11 mrežama, koristiti i za autentifikaciju na sveprisutnim 802.3 lokalnim mrežama. O EAP-u u bežičnim mrežama može se govoriti sa više aspekata:

- opis EAP paketa i protokola autentifikacije,
- enkapsulacija EAP paketa u 802.11 okvire,
- različite metode autentifikacije temeljene na EAP-u.

2.6.1. Protokol autentifikacije

U autentifikaciji sudjeluju dva subjekta:

- *klijent*, koji želi pristupiti mreži i čiji identitet treba provjeriti, te
- *autentifikator*, koji provjerava identitet i autorizaciju klijenta.

Postoje četiri tipa EAP paketa[5]:

- zahtjev (engl. *request*),
- odgovor (engl. *response*),
- uspjeh (engl. *success*) i
- neuspjeh (engl. *failure*).

Paketi tipa zahtjev i dogovor imaju podtip koji određuje koja metoda autentifikacije se koristi. Sadržaj paket interpretira se u ovisnosti o korištenoj metodi. Autentifikacija se izvodi tako da klijent i autentifikator razmjenjuju niz paketa:

- pakete tipa zahtjev šalje autentifikator,
- pakete tipa odgovor šalje klijent.

Autentifikacija završava kada autentifikator pošalje:

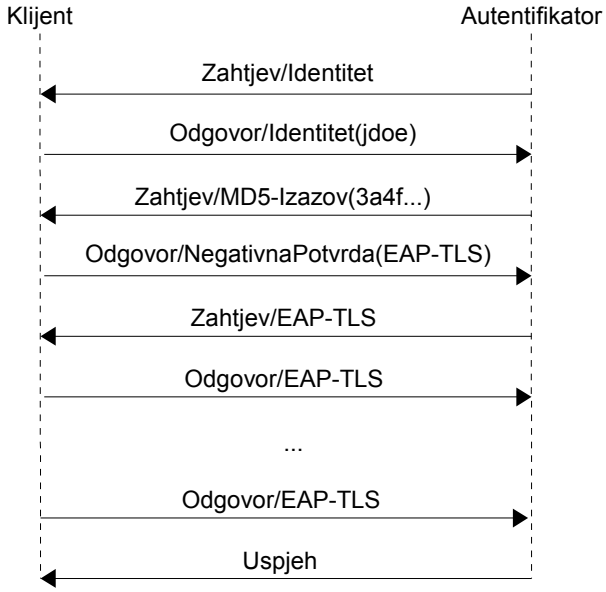
- paket tipa uspjeh, ako se klijent uspješno autentificirao, odnosno
- paket tipa neuspjeh, u suprotnom.

U nastavku je opisan primjer autentifikacije[6] korištenjem EAP-a (slika 2.5.)

1. Autentifikacija započinje paketom tipa *Zahtjev/Identitet*. Podtipom *Identitet* autentifikator zahtjeva od klijenta da se identificira.
2. Klijent odgovara paketom tipa *Odgovor/Identitet* koji sadrži identifikator klijenta (obično je to korisničko ime).
3. Autentifikator zatim odlučuje kojom će metodom autentificirati klijenta, te šalje zahtjev pripadnog podtipa. Tako će za MD5 autentifikaciju poslati

paket tipa *Zahtjev/MD5-Izazov*. Sadržaj paketa ovisi o odabranoj metodi autentifikacije.

4. Ukoliko se ne slaže s odabranom metodom autentifikacije, klijent odgovara paketom tipa *Odgovor/NegativnaPotvrda*. Podtip paketa *NegativnaPotvrda* (engl. *NAK*) omogućava klijentu da odbije ponuđenu metodu autentifikacije i sugerira neku drugu.
5. Autentifikator mijenja metodu autentifikacije (pri čemu može, ali ne mora uzeti u obzir sugestiju klijenta) i šalje prvi paket tipa *Zahtjev* za novo odabranu metodu. Koraci 4. i 5. ponavljaju se sve dok se ne dogovori klijentu i autentifikatoru prihvatljiva metoda autentifikacije.
6. Klijent i autentifikator nakon toga izmjenjuju određeni broj (ovisno o odabranoj metodi autentifikacije) paketa tipa *Zahtjev* i *Odgovor*. Podtip i sadržaj paketa ovise o odabranoj metodi.
7. Nakon primanja posljednjeg paketa tipa *Odgovor*, autentifikator može autentificirati klijenta. Ukoliko je klijent uspješno autentificiran autentifikator šalje paket tipa *Uspjeh*. U suprotnom šalje paket tipa *Neuspjeh*.



Slika 2.5. EAP Autentifikacija

2.6.2. Enkapsulacija u okvire

Standard 802.1X[4] definira na koji se način EAP koristi u IEEE lokalnim mrežama, a time i u bežičnim 802.11 lokalnim mrežama. U 802.1X standardu opisana je enkapsulacija EAP paketa u okvire nazvana EAPoL (engl. *EAP over LAN*) ili ponekad EAPoW (engl. *EAP over Wireless*). Definirano je više vrsta okvira, od kojih su najvažniji:

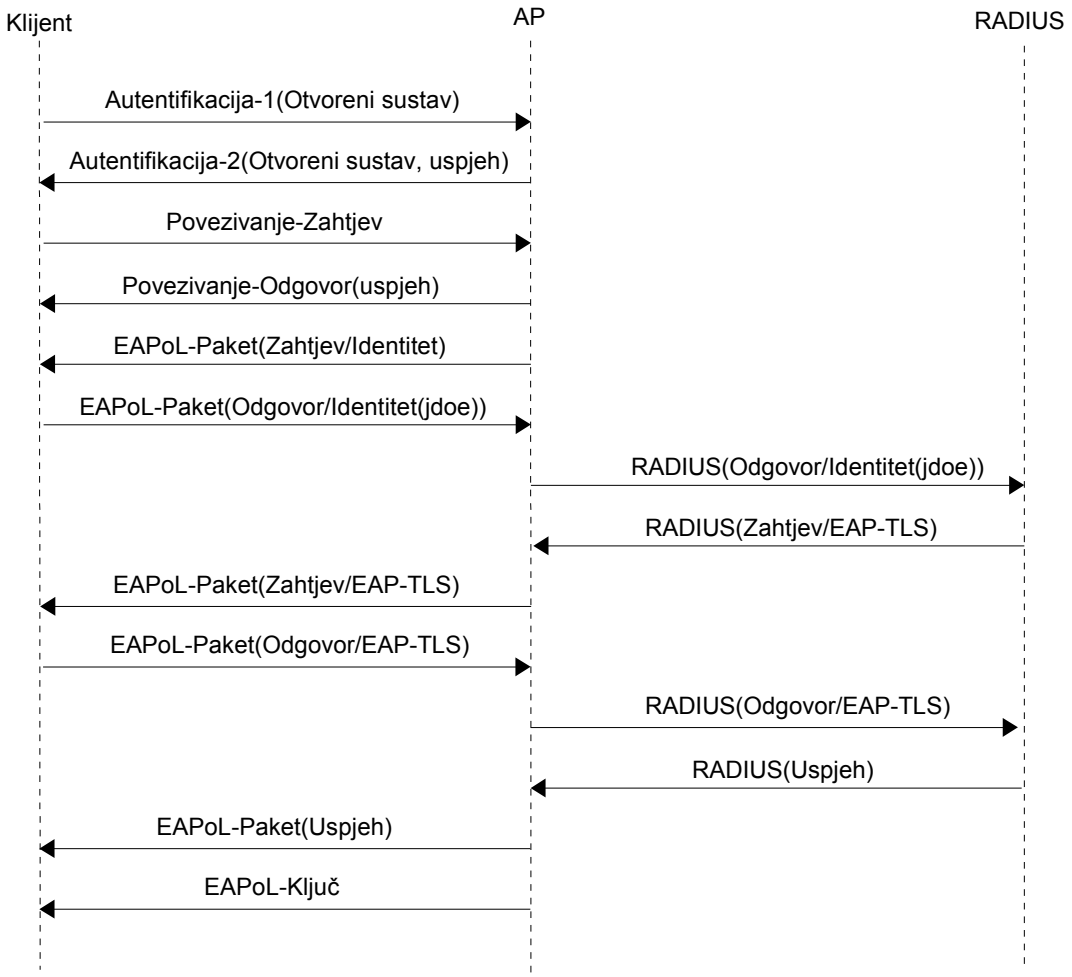
- EAPoL-Start, kojim klijent može zahtijevati pokretanje autentifikacije,
- EAPoL-Paket, koji enkapsulira EAP paket,

- EAPoL-Ključ, koji služi za prijenos ključa kriptiranja, i
- EAPoL-Kraj (engl. *EAPoL-Logoff*), kojim se klijent odjavljuje.

U autentifikaciji na 801.11 mreži sudjeluju tri subjekta:

- klijent, koji želi pristupiti mreži,
- pristupna točka, preko koje klijent pristupa mreži i
- autentifikacijski server, koji autentificira korisnika (redovito je to RADIUS server).

EAP paketi na putu između korisnika i pristupne točke enkapsulirani su u EAPoL okvirima, dok su na putu između pristupne točke i RADIUS servera enkapsulirani u pakete RADIUS protokola (atribut EAP-Message). Pristupna točka služi samo kao posrednik: ona EAP pakete iz EAPoL okvira prosljeđuje autentifikacijskom serveru u RADIUS paketima. EAP autentifikacija (slika 2.5.) provodi se zapravo između klijenta i RADIUS servera. Sve dok se uspješno ne autentificira, pristupna točka filtrira sav promet klijenta (osim EAPoL okvira koje prosljeđuje RADIUS serveru). Slika 2.6. pokazuje cjelokupni proces autentifikacije.



Slika 2.6. EAP autentifikacija na bežičnoj mreži

2.6.3. Metode autentifikacije

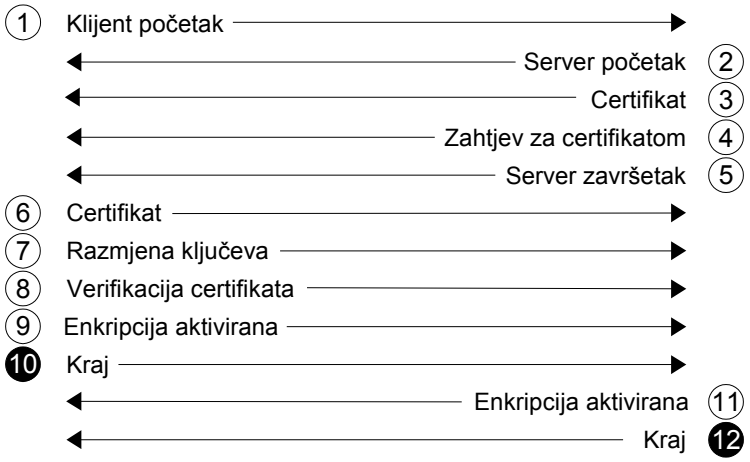
Iako je moguće koristiti bilo koju EAP metodu autentifikacije, bežične okruženje postavlja dodatne uvjete [7] od kojih su najvažnija sljedeća dva:

1. Metoda autentifikacije mora, osim autentifikacije korisnika, omogućiti i korisniku da autentificira mrežu na koju se spaja. Napadač može lako postaviti pristupnu točku sa istim identifikatorom (engl. *SSID – Service Set ID*) kao postojeća mreža. Ukoliko se korisnik spoji na napadačevu pristupnu točku (npr. zbog jačeg signala) i ne autentificira mrežu na koju se spaja, napadač to može iskoristiti za otkivanje podataka kojima se korisnik autentificira (lozinke i sl.). Napadač može prosljeđivati korisnički promet na pravu mrežu i pri tome ima potpunu kontrolu nad tim prometom (može ga koristiti za sve uobičajene mrežne napade) dok korisnik ne primjećuje gotovo ništa.
2. Metoda autentifikacije mora omogućiti sigurnu distribuciju WEP ključeva klijentu. Nakon uspješno završene autentifikacije klijent mora imati dovoljno informacija da odredi koji WEP ključ treba koristiti za enkripciju podataka. Posljedica toga je da ključ više nije potrebno klijentu dostaviti nekim drugim putem (npr. "ručno") prije korištenja mreže. Za svakog klijenta generira se različit ključ, a postaje moguće i automatizirano periodički mijenjati ključ ponavljanjem autentifikacije.

U nastavku će biti opisane najčešće korištene metode koje zadovoljavaju ove uvjete.

2.6.3.1. Autentifikacija certifikatom

Prva EAP metoda autentifikacije koja je zadovoljila oba kriterija bio je EAP-TLS[8]. Metoda se temelji na dokazanom TLS protokolu[9] koji se koristi za zaštitu osjetljivog prometa na Internetu. Subjekti se u TLS-u autentificiraju pomoću certifikata i dogovaraju ključ kriptiranja. Uspostava TLS sjednice prikazana je na slici 2.7. Bijelo obojeni redni brojevi predstavljaju poruke koje se izmjenjuju nekriptirane, dok je na poruke sa crno obojenim rednim brojem primijenjena enkripcija.

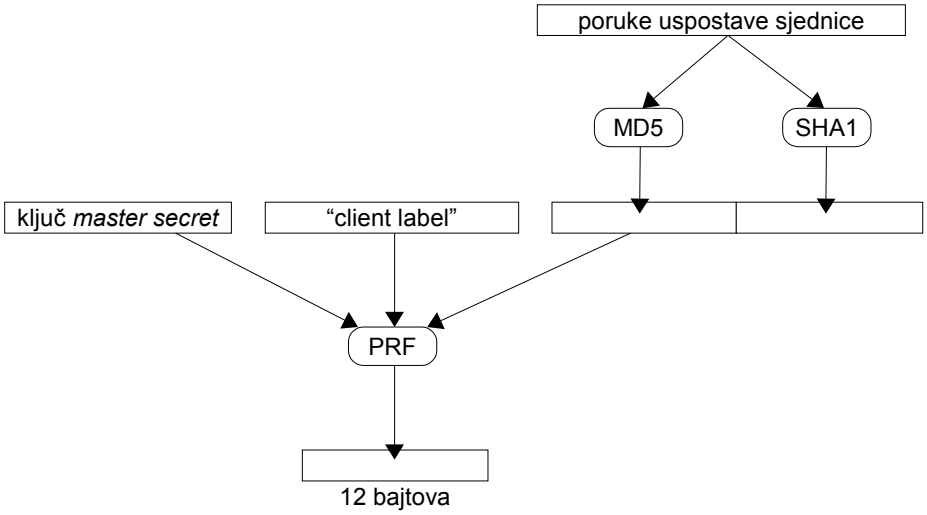


Slika 2.7. Uspostava TLS sjednice

Slika prikazuje tip i smjer poruka koje se izmjenjuju. TLS poruke prikazane na slici 2.7. jednake su onima koje se koriste prilikom uspostave TLS veze na Internetu. Razlika je jedino u tome što se za prijenos TLS poruka na Internetu koristi TCP, dok se u EAP-TLS-u za to koriste EAP paketi. U nastavku su opisani elementi pojedinih poruka bitni za EAP-TLS autentifikaciju:

1. Klijent započinje komunikaciju slanjem poruke *Klijent početak* (engl. *ClientHello*). Ta poruka, između ostaloga, sadrži na siguran način generiran slučajan broj (engl. *ClientRandom*), te listu podržanih algoritama za razmjenu ključeva, enkripciju i zaštitu integriteta (tj. sažetaka).
2. Server odgovara porukom *Server početak* (engl. *ServerHello*) koja sadrži na siguran način generiran slučajan broj (engl. *ServerRandom*), te identifikatore odabranih algoritama. Server odabire jedan algoritam za razmjenu ključeva, jedan algoritam enkripcije i jednu funkciju sažetka.
3. Server zatim šalje svoj certifikat, certifikat CA (engl. *Certification Authority*)koji je izdao certifikat servera i tako sve do (i uključivo) certifikata korijenskog CA.
4. Porukom *Zahtjev za certifikatom* (engl. *CertificateRequest*) server zahtjeva od klijenta da pošalje svoj certifikat. Pri tome navodi prihvatljive tipove certifikata s obzirom na korišteni algoritam digitalnog potpisa (npr. RSA), te listu imena prihvatljivih CA.
5. Poruka *Server završetak* (engl. *ServerHelloDone*) označava kraj slijeda poruka koje generira server.
6. Klijent odgovara porukom u kojoj šalje svoj certifikat.
7. Klijent generira i šalje ključ *premaster secret* kriptiran javnim ključem servera. Javni ključ servera sadržan je u certifikatu iz koraka 3. Ključ *premaster secret* je slučajan broj koji klijent generira na siguran način. Iz tog ključa klijent i server računaju ključ *master secret*.
8. Poruka *Verifikacija certifikata* omogućava serveru da provjeri posjeduje li klijent tajni ključ pridružen certifikatu iz koraka 6. Poruka sadrži digitalni potpis (tj. sažetak kriptiran privatnim ključem klijenta) svih prethodno razmijenjenih poruka.
9. Porukom *Enkripcija aktivirana* (engl. *ChangeChipperSpec*) aktivira se enkripcija u smjeru od klijenta prema serveru.
10. Posljednja poruka koju klijent šalje služi kao test kriptiranog kanala od klijenta prema serveru. Sadržaj poruke kriptiran je dogovorenim algoritmom i ključem. Sadržaj poruke klijent računa kao što je prikazano na slici 2.8. Server računa sadržaj poruke na isti način i uspoređuje ga s podacima koje dobiva dekriptiranjem. Ukoliko su jednaki, kriptirani kanal od klijenta prema serveru je uspješno uspostavljen.
11. Porukom *Enkripcija aktivirana* (engl. *ChangeChipperSpec*) server aktivira enkripciju u smjeru od servera prema klijentu.

12. Poruka *Kraj* (engl. *Finished*) služi kao test kriptiranog kanala u smjeru od servera prema klijentu (analogno koraku 10.). Sadržaj poruke generira se na isti način kao na slici 2.8., ali se koristi niz znakova “server label” (umjesto “client label”). Ukoliko klijent uspješno dekriptira primljenu poruku kriptirani kanal od servera prema klijenta uspješno je uspostavljen.



Slika 2.8. Generiranje sadržaja poruke *Kraj*

Autentifikacija se u EAP-TLS-u obavlja pomoću certifikata. Server klijentu u koraku 3. šalje svoj certifikat, te lanac certifikata sve do korijenskog CA. Nakon primitka navedenih informacija klijent treba provjeriti da li je certifikat servera valjan. Provjera valjanosti uključuje provjeru valjanosti svih certifikata u lancu do korijenskog CA. Pri tome treba koristiti certifikat korijenskog CA dobiven nekim drugim putem jer napadač (koji se predstavlja kao autentifikacijski server) može generirati lažni certifikat korijenskog CA i svih ostalih certifikata u lancu. Osim toga, certifikat korijenskog CA mora biti jedan od korijenskih CA kojima klijent vjeruje[4]. Digitalni potpisi svih certifikata u lancu (uključivo i certifikat servera) moraju biti ispravni. Osim digitalnog potpisa potrebno je provjeriti i da li su certifikati vremenski valjani (trenutno vrijeme mora se nalaziti u periodu valjanosti svih certifikata).

Klijent obično ima i mogućnost definirati ime servera kojemu vjeruje. Tada se definirano ime uspoređuje s onim sadržanim u certifikatu (CN podelementu Subject elementa certifikata). Odabirom povjerljivog korijenskog CA i imenom povjerljivog servera klijent može pobliže definirati, pa čak i jedinstveno odrediti certifikat servera kojemu će vjerovati. Iako je poželjno provjeriti da li se certifikat servera nalazi u listi opozvanih certifikata (engl. *CRL – Certificate revocation list*) to obično nije moguće jer se klijent tek spaja na mrežu. Eventualno je moguće koristiti listu koja se distribuira nekim drugim putem (bolje) ili provjeriti listu opozvanih certifikata nakon spajanja (lošije). Ukoliko je opisanim nizom provjera klijent utvrdio valjanost i prihvatljivost certifikata servera preostaje mu još utvrditi posjeduje li server privatni ključ pridružen certifikatu (tj. javnom ključu sadržanom u certifikatu).

4 Autentifikacijski softver na klijentskoj strani obično ima mogućnost odabira CA kojemu se vjeruje – samo certifikati servera koje je izdao odabrani CA biti će prihvaćeni.

Klijent zaključuje da server posjeduje privatni ključ ukoliko može dekriptirati podatke kriptirane javnim ključem servera. Tako u poruci *Razmjena ključeva* (korak 7.) klijent šalje serveru ključ *premaster secret* kriptiran javnim ključem servera. Ukoliko server posjeduje privatni ključ, on će ispravno dekriptirati poruku i dobiti ključ *premaster secret*. Iz njega će prvo odrediti ključ *master secret*, a zatim i ključ enkripcije za smjer od servera prema klijentu. Klijent će na osnovnu poruke *Kraj* (12. korak) odrediti posjeduje li server privatni ključ. Ukoliko uspješno dekriptira poruku, klijent zaključuje da server poznaje privatni ključ.

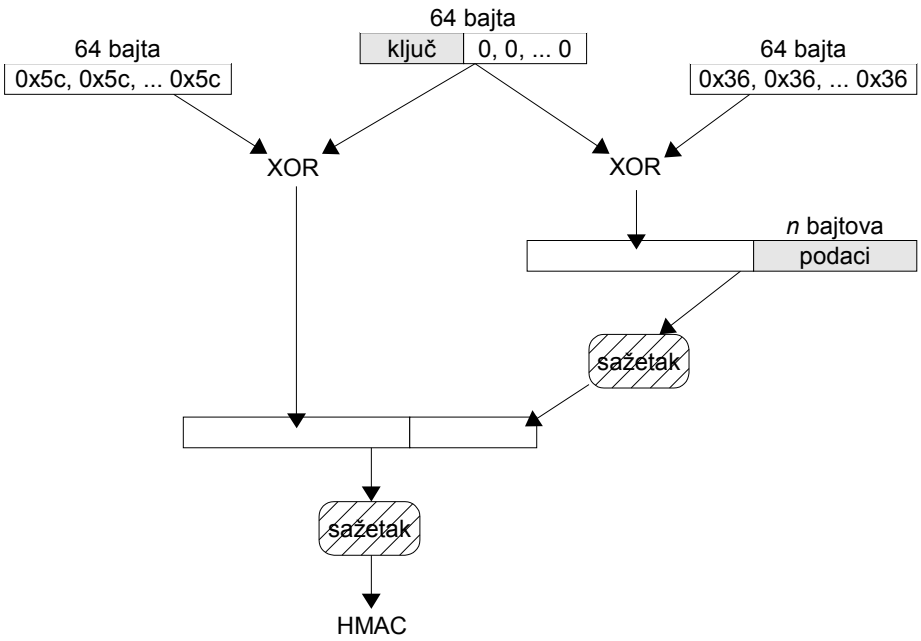
Server autentificira klijenta na sličan način. Prvo se provjerava valjanost certifikata klijenta. To uključuje provjeru ispravnosti digitalnog potpisa, te provjeru vremenske valjanosti certifikata. Server posjeduje listu povjerljivih CA, te prihvaća samo one klijentske certifikate koje je izdao neki od povjerljivih CA. Korištenjem liste opozvanih certifikata provjerava se da li je certifikat opozvan. Ukoliko se certifikat pokaže kao valjan, potrebno je još provjeriti posjeduje li klijent privatni ključ. Za tu provjeru koristi se poruka *Verifikacija certifikata* (korak 7.) Navedena poruka sadrži digitalni potpis svih prethodno razmijenjenih TLS poruka. Server generira sažetak svih prethodno razmijenjenih poruka, te ga uspoređuje sa sažetkom dobivenim dekriptiranjem digitalnog potpisa. Sažetci će biti jednaki ukoliko je klijent kriptirao sažetak svojim privatnim ključem.

Prethodni odjeljci opisali su korištenje TLS-a za autentifikaciju. Preostaje još opisati kako se pomoću TLS-a dobivaju ključevi WEP enkripcije. Tijekom uspostave TLS sjednice klijent i server dogovaraju ključ *premaster secret*. Klijent generira ključ *premaster secret* i šalje ga serveru kriptiranog javnim ključem servera (poruka *Razmjena ključeva*). Iz ključa *premaster secret* korištenjem PRF-a (engl. *Pseudorandom function*) klijent i server određuju ključ *master secret* duljine 48 bajtova. PRF prima tri parametra: ključ, niz znakova i sjeme, a na izlazu generira niz proizvoljne duljine. Kod generiranja ključa *master secret* iz ključa *premaster secret* (slika 2.12.) kao parametri PRF-a koristi se:

- ključ *premaster secret*,
- niz znakova “master secret” i
- slučajni brojevi koje su server i klijent razmijenili kod uspostave TLS sjednice (poruke *Klijent početak* i *Server početak*).

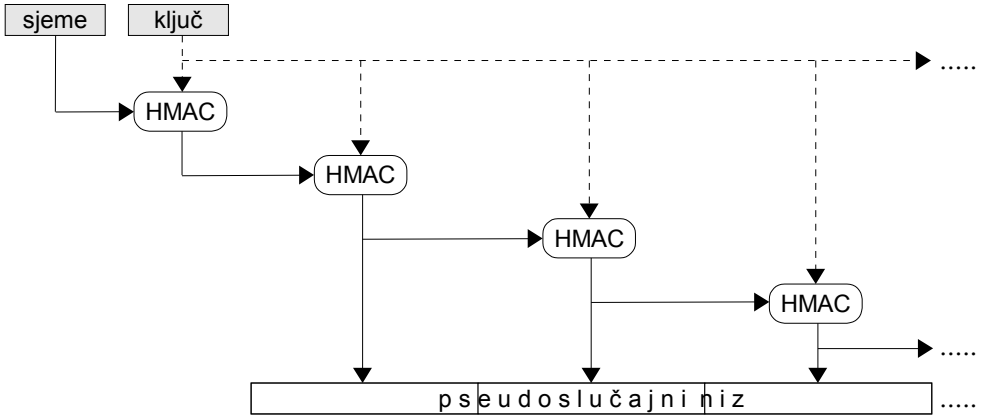
PRF se koristi kada je iz niza konačne duljine potrebno generirati pseudoslučajni niz proizvoljne duljine. PRF se temelji na MAC (engl. Message Authentication Code) funkcijama. MAC funkcije računaju sažetak poruke parametriziran tajnim ključem. Parametrizacija ključem sprječava napadača da promjeni sadržaj poruke i generira novi sažetak. MAC sažetak može generirati (i provjeriti) samo onaj tko poznaje ključ. Prema tome, MAC funkcije, osim provjere integriteta, omogućavaju i provjeru autentičnosti poruke. MAC ostvaren pomoću funkcija sažetka (slika 2.9.) naziva se HMAC [10]. Vrijednost HMAC funkcije dobiva se računanjem sažetka u dvije razine. Kao ulaz sažetka u prvoj razini koristi se niz koji sadrži tajni ključ i podatke. Za ulaz sažetka u drugoj razini koristi se izlaz prve razine i tajni ključ. Dobiveni rezultat ovisi i o korištenom tajnom ključu i o podatcima. Minimalna

promjena u bilo kojemu od tih parametara potpuno mijenja izlaz HMAC funkcije. Duljina izlaza iz HMAC funkcije ovisi o korištenoj funkciji sažetka jer je izlaz posljednjeg bloka sažetka ujedno i izlaz HMAC funkcije.



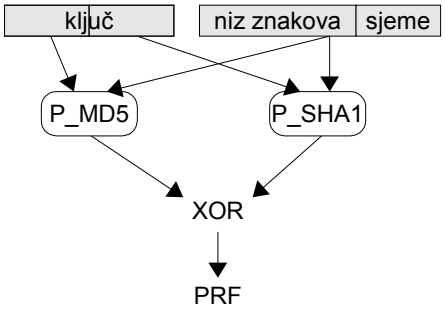
Slika 2.9. Računanje HMAC funkcije

HMAC funkcija koristi se u izgradnji P_hash funkcija[11]. P_hash funkcija generira pseudoslučajni niz proizvoljne duljine, a kao parametre prima sjeme i tajni ključ (slika 2.10.) HMAC se računa onoliko puta koliko je potrebno da se dobije izlaz potrebne duljine.



Slika 2.10. Računanje P_hash funkcije

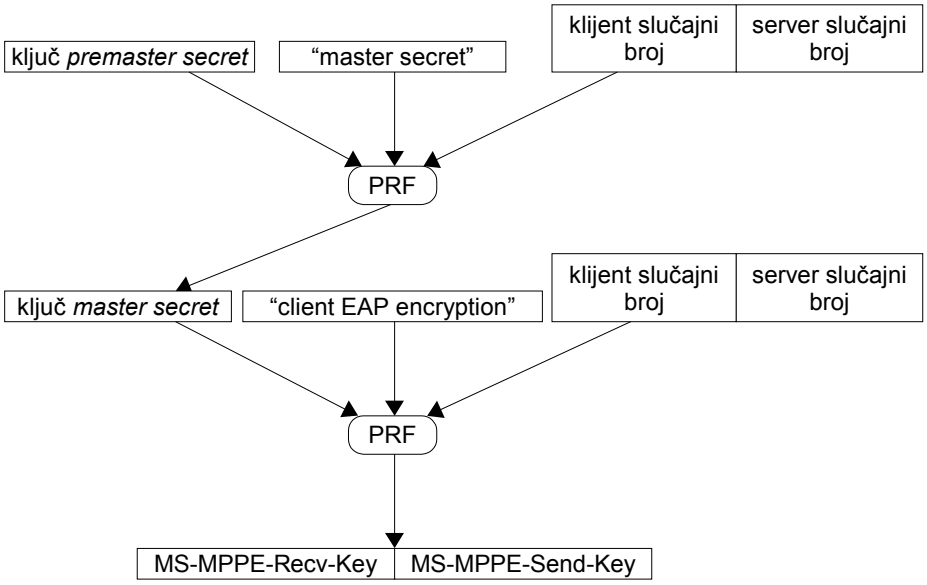
Izlaz PRF-a dobiva se XOR kombiniranjem izlaza iz dvije P_hash funkcije: P_MD5 i P_SHA1 (slika 2.11.) Dvije P_hash funkcije (svaka koristi polovicu ključa) koriste se za slučaj da se tijekom vremena pokaže da je jedna od funkcija sažetka nesigurna. Tada preostala sigurna funkcija sažetka i dalje čuva tajnost pseudoslučajnog izlaza.



Slika 2.11. Računanje PRF funkcije

Nakon što odredi ključ *master secret*, RADIUS server korištenjem PRF-a generira 64 bajta podataka (slika 2.12.) Kao parametri koriste se:

- ključ *master secret*,
- niz znakova “client EAP encryption”, te
- slučajni brojevi koje su klijent i server razmijenili kod uspostave TLS sjednice.



Slika 2.12. Generiranje WEP ključa

Izračunati niz šalje se pristupnoj točki u RADIUS paketu i to:

- prvih 32 bajta kao vrijednost atributa *MS-MPPE-Recv-Key*,
- drugih 32 bajta kao vrijednost atributa *MS-MPPE-Send-Key*.

Klijent također posjeduje ključ *master secret* i na isti način generira dva nova ključa dužine 32 bajta. Generirani ključevi koriste se za razmjenu WEP ključa između pristupne točke i klijenta. Nakon primitka paketa od RADIUS servera, pristupna točka prosljeđuje klijentu EAP paket tipa *Uspjeh* (također sadržan u primljenom RADIUS paketu). Pristupna točka zatim generira WEP ključ koji će koristiti za komunikaciju s dotičnim klijentom. Generirani WEP ključ se dostavlja klijentu korištenjem EAPoL-Ključ okvira (tablica 2.7.)

Tablica 2.7. Sadržaj okvira EAPoL-Ključ

Element	*Dužina u bajtovima*
Inicijalizacijski vektor	16
Indeks ključa	1
Dužina WEP ključa	2
Kriptirani WEP ključ	13
MAC sažetak okvira	16

WEP ključ se kriptira RC4 algoritmom, a ključ enkripcije čine zajedno:

- inicijalizacijski vektor i
- vrijednost iz *MS-MPPE-Recv-Key* atributa.

Inicijalizacijski vektor i kriptirani WEP ključ spremaju se u pripadna polja u okviru. Za okvir se računa HMAC-MD5[5] sažetak. Kao MAC ključ koristi se vrijednost iz *MS-MPPE-Send-Key* atributa RADIUS paketa.

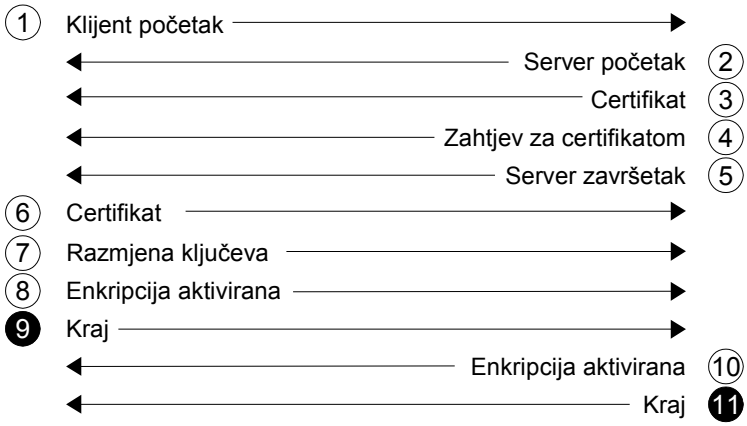
Klijent nakon primitka EAPoL-Ključ okvira provjerava MAC sažetak. Ukoliko primljeni i izračunati sažetak nisu jednaki, okvir je izmijenjen u prijenosu te se odbacuje. Ako je primljeni sažetak ispravan WEP ključ se dekriptira. Klijent sprema WEP ključ pod indeksom navedenim u EAPoL-Ključ okviru. Ukoliko je najviši bit indeksa postavljen ključ će se koristiti za enkripciju svih podatkovnih okvira koje klijent šalje, te za dekripciju okvira namijenjenih samo njemu (engl. *unicast*). Ako najviši bit nije postavljen, ključ će se koristi za dekripciju podatkovnih okvira namijenjenih svim stanicama (engl. *broadcast*). Primjer takvog prometa su ARP zahtjevi - njih je potrebno proslijediti svim stanicama. Kada ključ za *broadcast* promet nebi postojao, pristupna točka bi okvir morala poslati svakome klijentu posebno. Korištenjem *broadcast* ključa, kojeg poznaju svi klijenti, moguće je okvir poslati svim klijentima istovremeno (korištenjem jednog okivra). Ključ za *broadcast* promet obično se periodički mijenja i pristupna točka tada svim klijentima šalje novi ključ pomoću EAPoL-Ključ okvira.

2.6.3.2. Autentifikacija lozinkom

EAP-TLS zadovoljava uvjete za korištenje u bežičnom okruženju i dostupan je na svim platformama, kako na strani klijenata tako i na serverima. No EAP-TLS ima i jedan bitan nedostatak - on zahtjeva PKI (engl. P*ublic Key Infrastructure*). U implementaciji se pokazalo da je razvoj PKI-a za mnoge organizacije prevelika prepreka. Uspostava i održavanje PKI-a složen je proces, pa je EAP-TLS jedino prihvaćen u organizacijama koje su već imale razvijen PKI ili su bile spremne uložiti sredstva u njegov razvoj. Najteži dio tog procesa predstavlja generiranje i održavanje certifikata na korisničkoj strani. Zbog toga različiti proizvođači kreću u razvoj rješenja koje neće zahtijevati razvijeni PKI. Rezultati tog rada su dvije često korištene EAP metode: PEAP[12] i EAP-TTLS[13]. Ove dvije metode temelje se na dobrim stranama EAP-TLS-a, ali pri tome pokušavaju izbjeći zamku PKI-a. PEAP i EAP-TTLS često se spominju zajedno jer im je funkcionalnost vrlo slična, ali

5 HMAC sažetak temeljen na MD5 sažetku

su razvijeni od različitih proizvođača. PEAP i EAP-TTLS predstavljaju drugu generaciju EAP metoda temeljenih na TLS protokolu.

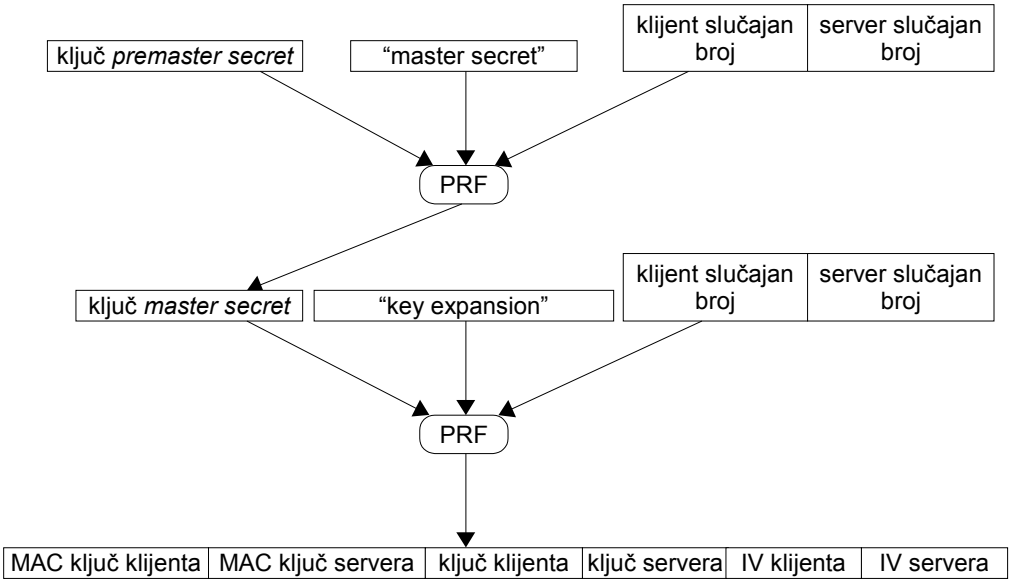


Slika 2.13. Uspostava TLS sjednice u PEAP-u i EAP-TTLS-u

Osnovna ideja je sljedeća: autentifikacija se provodi u dvije faze. U prvoj fazi (slika 2.13.) uspostavlja se TLS sjednica i autentificira se server. Nakon toga slijedi druga faza u kojoj se preko kriptirane TLS veze autentificira klijent. Ovaj način rada u mnogome podsjeća na korištenje TLS-a na Webu. Dodatna prednost ove metode je da se za autentifikaciju korisnika u drugoj fazi mogu koristiti postojeće metode autentifikacije. Osim što nije potrebno implementirati PKI na korisničkoj strani, može se iskoristiti postojeća autentifikacijska infrastruktura. Moguće je koristiti i manje sigurne metode autentifikacije korisnika (npr. lozinke) jer autentifikacijski podaci u drugoj fazi putuju zaštićeni TLS kriptiranim kanalom. Autentifikacija servera i dalje se obavlja pomoću certifikata, ali je broj certifikata koje treba održavati daleko manji nego kod EAP-TLS-a. Razlike između PEAP-a i EAP-TTLS-a su u drugoj fazi – metode koriste različite formate zapisa autentifikacijskih podataka, no ostvarena funkcionalnost je gotovo identična. Zbog toga je malo vjerojatno je da će obje metode opstati na dugi rok. Koja će metoda preživjeti neće ovisiti samo o tehničkim prednostima. Ulogu će igrati i drugi faktori kao što su podrška proizvođača, dostupnost i cijena programske podrške, kako na strani klijenta tako i na serverima.

Korisno je usporediti uspostavu TLS tunela kod EAP-TLS-a (slika 2.7.) i PEAP-a (slika 2.13.), odnosno EAP-TTLS-a. Poruke koje šalje server u oba su slučaja jednake, razlika je u odgovoru klijenta. U oba slučaja server zahtjeva od klijenta certifikat (poruka *Zahtjev za certifikatom*). No iako obje slike sadrže poruku *Certifikat* koji šalje klijent, ta poruka je u slučaju PEAP-a prazna (tj. sadrži certifikat duljine nula). Na taj način klijent obavještava servera da ne posjeduje certifikat ili se ne želi autentificirati certifikatom. Pošto klijent nije poslao certifikat, izostavlja i poruku *Verifikacija certifikata*. Prema tome, PEAP i EAP-TTLS omogućuju autentifikaciju klijenta certifikatom, ali se ta mogućnost rijetko koristi. Nakon uspostavljenog TLS tunela započinje druga faza u kojoj klijent i server razmjenjuju niz kriptiranih TLS poruka tipa *Aplikacijski podaci* (engl. *ApplicationData*). Sadržaj tih poruka različit je za navedene metode i biti će opisan za svaku posebno. Kriptiranje u TLS tunelu izvodi se prema algoritmima dogovorenim u prve

dvije TLS poruke: *Klijent početak* i *Server početak*. Korišteni ključevi enkripcije (za svaki smjer različiti) izvode se iz ključa *master secret* kao što je prikazano na slici 2.14. MAC ključevi koriste se za računanje MAC sažetaka poruka. Inicijalizacijski vektori (IV) koriste se kada algoritam enkripcije radi u CBC načinu rada[2].



Slika 2.14. Generiranje TLS ključeva enkripcije

PEAP (engl. *Protected EAP*) nastao je kao rezultat zajedničkog rada triju velikih kompanija: Microsoft, Cisca i RSA. Kao što ime metode sugerira, u drugoj fazi se provodi zaštićena (pomoću TLS enkripcije) autentifikacija klijenta korištenjem EAP-a. PEAP dakle enkapsulira drugu EAP metodu unutar TLS tunela. Sam način enkapsulacije ovisi o verziji protokola. Trenutno postoje tri verzije protokola: 0, 1 i 2. Verzija 0[14] je najčešće korištena (jer je standardno uključena u Windowse XP). Verzija 1[15] je također široko implementirana ali se rjeđe koristi, dok je verzija 2[12] trenutno dostupna samo kao specifikacija.

Verzije se međusobno razlikuju prije svega načinom enkapsulacije EAP paketa. U verziji 0 paketi se enkapsuliraju bez dijela zaglavlja čija se vrijednost može zaključiti iz zaglavlja vanjske EAP metode. Također se koristi nova metoda s tipom 33 nazvana EAP-Proširenja (engl. *EAP-Extensions*) čija je osnovna funkcija enkapsulacija EAP paketa. U verziji 0 EAP-Proširenje se koristi samo za prijenos EAP-Uspjeh i EAP-Neuspjeh poruka unutar tunela. U verziji 1 EAP paketi unutar TLS tunela se enkapsuliraju sa cijelim zaglavljem. U verziji 2 za enkapsulaciju se koristi EAP-Proširenja koji se sada naziva EAP-TLV (engl. *Type Length Value*). Na taj način zapravo se dobiva dvostruka enkapsulacija: PEAP unutar TLS tunela prenosi EAP-TLV pakete, dok EAP-TLV paketi sadrže pakete EAP metode koje se koristi za autentifikaciju klijenta. EAP-TLV također uključuje zaštitu od *MitM* (engl. *Man in the Middle*) napada na sličan način kao što se u TLS-u koriste poruke *Kraj*. Generiranje ključeva također se razlikuje među verzijama. Verzija 0 generira ključeve na isti način kao EAP-TLS (slika 2.12.) Verzija 1 razlikuje se samo utoliko što se koristi niz

znakova “client PEAP encryption” (umjesto “client EAP encryption”). Verzija 2 koristi složeniji način generiranja ključeva koji kombinira EAP-TLS-om generirani ključ i ključeve generirane autentifikacijom u drugoj fazi. Iako PEAP ima mogućnost tuneliranja bilo koje (ili više metoda u seriji) EAP metode (uključivo i samoga sebe), redovito se za autentifikaciju u drugoj fazi koristi samo MS-CHAPv2.

EAP-TTLS (engl. *Tunneled TLS*) podržavaju dva manja proizvođača: Funk i Meetinghouse. Sve navedeno za PEAP vrijedi i za EAP-TTLS. Razlika je samo u načinu enkapsulacije podataka u tunelu i načinu generiranja ključeva. Podatci se u tunelu prenose enkapsulirani u parove atribut-vrijednost (engl. *AVP – Attribute Value Pair*) definirane Diametar protokolom. Diametar je nasljednik RADIUS protokola, pa Diametar AVP-ovi uključuju i RADIUS AVP-ove. Takva enkapsulacija omogućava komunikaciju sa autentifikacijskim serverom u njemu prirodnom formatu. Tako je preko postojećih AVP-ova moguće tunelirati sve najčešće korištene protokole autentifikacije (PAP, CHAP, MS-CHAP i MS-CHAPv2). EAP metode moguće je tunelirati preko atributa EAP-Poruka (engl. *EAP-Message*). Generiranje ključeva slično je kao kod EAP-TLS-a, sa razlikom da se koristi niz znakova “ttls keying material” (umjesto “client EAP encryption”).

Preostale EAP metode zbog različitih razloga nisu šire prihvaćene za korištenje u bežičnim mrežama. Mnoge od njih ne zadovoljavaju uvjete za korištenje u bežičnom okruženju (npr. EAP-MD5). Neke metode zahtijevaju posebnu opremu pa su skuplje za implementaciju i manje raširene (npr. EAP-SIM). Druge su specifične za određenog proizvođača, nisu javno specificirane, nemaju široko dostupnu programsku podršku ili su u njima pronađeni sigurnosni propusti (npr. LEAP).

2.7. Sigurnosno proširenje standarda

Korištenjem EAP-a riješeno je pitanje sigurne autentifikacije u bežičnim mrežama. Preostao je još problem osiguranja tajnosti i integriteta podataka. WEP je standardom defininirani algoritam koji je trebao riješiti te probleme. No nedugo nakon izlaska standarda pokazalo se da WEP kvalitetno ne rješava niti problem privatnosti niti integriteta podataka. WEP je eventualno moguće koristiti za zaštitu tajnosti podataka tako da se periodički mijenja ključ kriptiranja. Time se onemogućava napadača da sakupi dovoljno kriptiranih okvira iz kojih bi mogao otkriti ključ kriptiranja. Periodička izmjena WEP ključa najčešće se koristi u kombinaciji sa prije opisanim EAP metodama. Pošto se prilikom svake autentifikacije generira novi WEP ključ, forsiranjem periodičkog ponavljanja autentifikacije generiraju se svježi WEP ključevi. Dugoročno rješenje tih problema trebao bi donijeti standard 802.11i [16]. To je zapravo dodatak 802.11 standardu koji opisuje dva nova protokola enkripcije podataka: TKIP i CCMP.

2.7.1. Protokol enkripcije CCMP

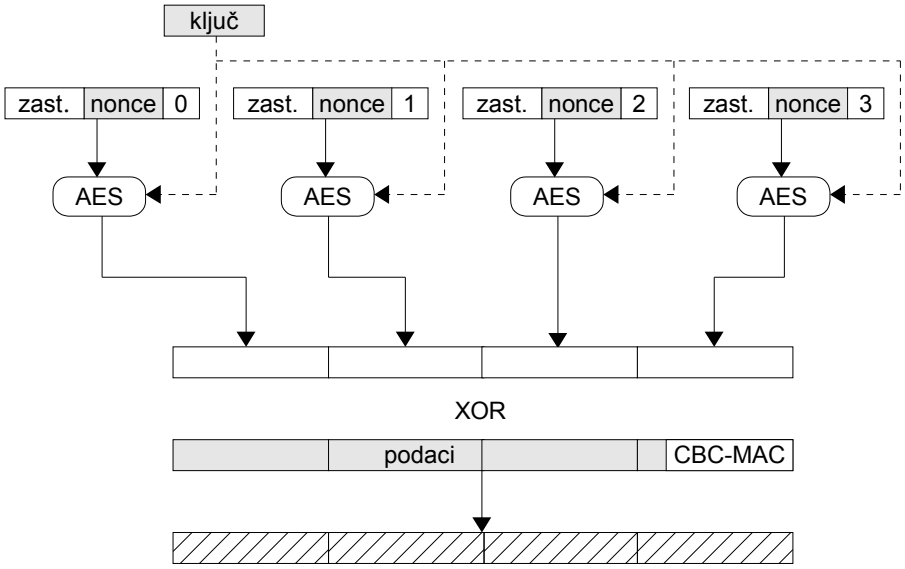
CCMP (engl. *Counter mode/CBC-MAC Protocol*) bi trebao biti dugoročno osigurati potrebnu sigurnost u bežičnim mrežama. Temelji se na AES algoritmu, a osim tajnosti osigurava integritet i autentičnost podatkovnih okvira. Kao što ime sugerira, za kriptiranje se koristi AES blok algoritam u

Counter načinu rada, dok se integritet i autentičnost osigurava CBC-MAC načinom rada AES algoritma. Zajedno ta dva načina čine CCM (engl. *Counter mode/CBC-MAC*) način rada[17]. Pri tome se koriste 128 bitni ključevi i 128 bitna veličina bloka.

2.7.1.1. Algoritam enkripcije

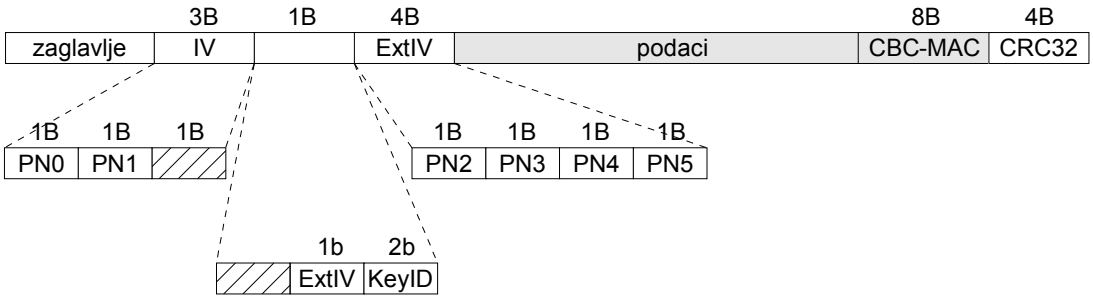
CCMP protokol koristi AES enkripciju brojačem (engl. *Counter mode*) [2]. U ovom načinu rada blok algoritam koristi se za generiranje niza bitova. Dobiveni niz bitova miješa se XOR operacijom sa podacima da bi se dobio kriptirani niz. Niz bitova dobiva se kriptiranjem blokova koji sadrže:

- zastavice,
- jedinstvenu vrijednost (engl. *Nonce*) i
- brojač (engl. *counter*).



Slika 2.15. Enkripcija brojačem u CCMP-u

Jedinstvenu vrijednost čini MAC adresa izvora okvira i redni broj okvira (engl. *PN - Packet Number*). Ista jedinstvena vrijednost koristi se i kod generiranja CBC-MAC-a. MAC adresa koristi se zato da isti redni brojevi okvira u različitim smjerovima (od i prema pristupnoj točki) daju različite jedinstvene vrijednosti, a time i različiti kriptirani niz. Redni broj pridružuje se okvirima u uzlaznom redoslijedu pa služe za detekciju ubacivanja starih okvira u mrežu. Prijemnik pamti redni broj posljednjeg primljenog okvira i prihvaća samo okvire sa rednim brojem većim od posljednjeg viđenog. Obično se nakon uspostave veze vrijednost posljednjeg viđenog okvira postavlja na 0, a okvirima se pridružuju redni brojevi od 1 na dalje. Redni broj okvira duljine je 6 bajtova i smješta se nakon zaglavlja okvira (slika 2.16.), a prije kriptiranih podataka. Dva bajta rednog broj smještaju se na mjesto gdje se kod WEP-a nalazi inicijalizacijski vektor, dok se ostala 4 smještaju iza identifikatora ključa u polju proširenog inicijalizacijskog vektora (engl. *ExtIV - Extended IV*). Početna vrijednost brojača je nula i povećava se za svaki blok koji se kriptira (slika 2.15.) Kriptira se sadržaj okvira kojemu je dodan CBC-MAC sažetak.



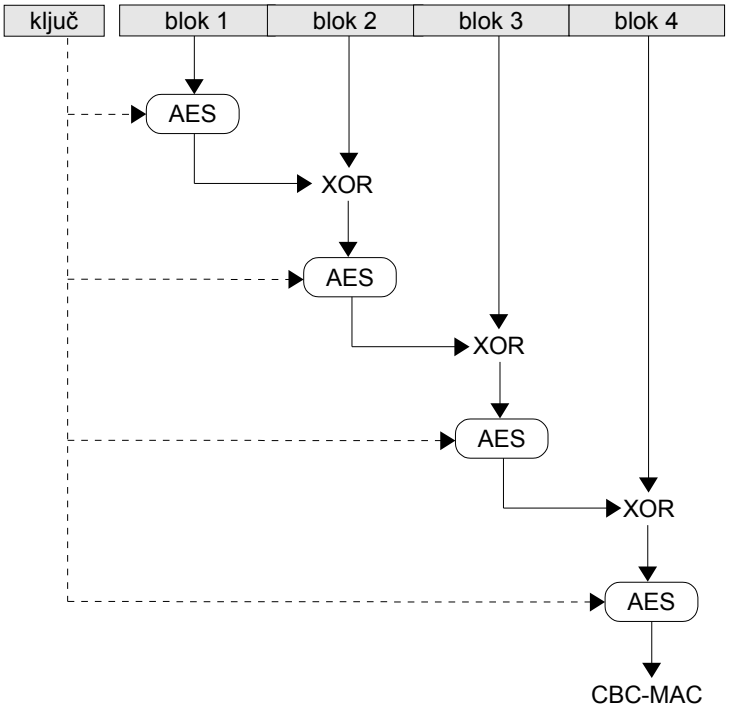
Slika 2.16. CCMP kriptirani okvir

2.7.1.2. Zaštita integriteta i autentifikacija okvira

CBC-MAC temelji se na CBC (engl. *Chiper Block Chaining*) načinu rada [2] algoritma enkripcije (slika 2.17.) U tom načinu rada blokovi podataka se prije kriptiranja XOR miješaju sa kriptiranim podacima prošlog bloka. CBC-MAC jednak je posljednjem bloku podataka dobivenih CBC kriptiranjem. Zbog XOR miješanja, vrijednost posljednjeg kriptiranog bloka ovisi o svim blokovima podataka. Zbog korištenja algoritma enkripcije vrijednost posljednjeg kriptiranog bloka ovisi i o ključu. Prema tome, neispravan MAC sažetak znači ili izmjene u podacima (povreda integriteta), ili korištenje pogrešnog ključa enkripcije (povreda autentičnosti). Slika 2.17. prikazuje računanje CBC-MAC-a za podatke podijeljene u četiri bloka. Izlaz posljednjeg AES bloka predstavlja CBC-MAC. Ukoliko se koristi MAC manje dužine, potreban broj bita uzima se s početka izlaznog bloka, dok se ostali bitovi odbacuju. Pošto CCMP koristi CBC-MAC od 64 bita i AES sa veličinom bloka 128 bita, posljednjih 64 bita se odbacuje. Blokovi koji se autentificiraju u CBC-MAC-u sadrže:

- zaglavlje,
- dužina dodatnih podataka,
- dodatne podatke (engl. *AAD - Additional Authentication Data*) i
- sadržaj okvira.

Zaglavlje opisuje niz podataka koji se autentificira. Zaglavlje sadrži zastavice, jedinstvenu vrijednost (MAC adresa izvora i redni broj okvira) te dužinu podataka. Zastavice, između ostalog, određuju autentificiraju li se neki dodatni podaci. Ukoliko je dotična zastavica postavljena, nakon zaglavlja slijedi dužina dodatnih podataka. Dodatne podatke koji se autentificiraju zapravo čini veći dio zaglavlja okvira. Time se sprječavaju razni napadi temeljeni na izmjeni zaglavlja okvira (npr. izvorišne MAC adrese). Ovako formirani niz podataka dijeli se u blokove dužine 128 bita (jer se koristi AES sa blokom od 128 bita) i predstavlja ulaz u CBC-MAC algoritam.



Slika 2.17. Računanje CBC-MAC-a

2.7.2. Protokol enkripcije TKIP

Iako se CCMP smatra trajnim rješenjem sigurnosti u bežičnim mrežama, njegova implementacija vjerojatno će zahtijevati zamjenu postojeće opreme. Zbog toga 802.11i standard sadrži i opis opcionalnog TKIP protokola (engl. *TKIP - Temporal Key Integrity Protocol*), koji bi se trebao moći implementirati i na postojećoj opremi samo izmjenom *firmware*-a. TKIP i dalje u pozadini koristi WEP algoritam, pa je moguće koristiti postojeće sklopovske implementacije tog algoritma. Viša razina sigurnosti postiže se generiranjem različitog WEP ključa[6] za svaki okvir, te dodavanjem sažetka otpornijeg na izmjene nego što je to WEP kriptirani CRC32.

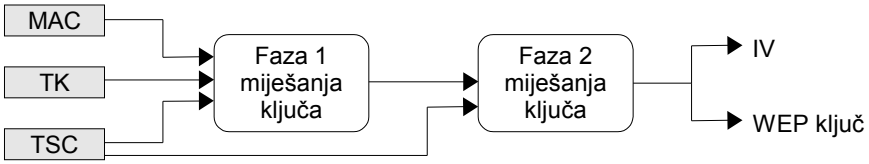
2.7.2.1. Algoritam enkripcije

Za svaki okvir podataka generira se novi RC4 ključ. Podaci koji se koriste za generiranje su:

- MAC adresa predajnika,
- ključ (engl. *TK - Temporal Key*) i
- redni broj okvira (engl. *TSC - TKIP Sequence Counter*).

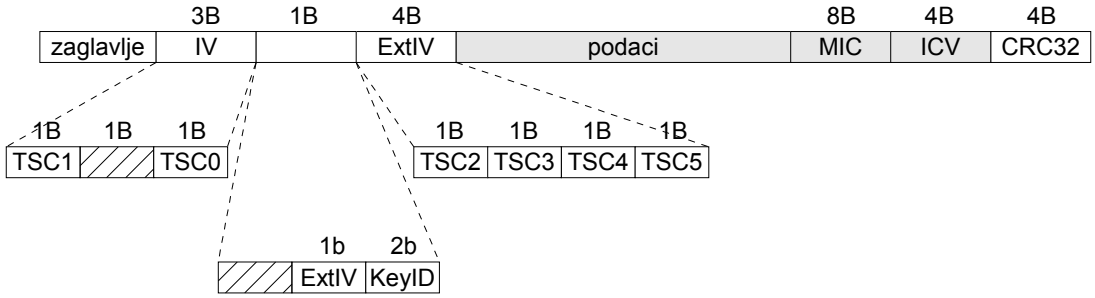
Generiranje RC4 ključa (engl. *key mixing*) se provodi u dvije faze (slika 2.18.) Prva faza koristi samo 32 viša bita rednog broja okvira tako da je rezultat te faze moguće ponovno koristiti za velik broj okvira. Druga faza koristi nižih 16 bita rednog broja i na svom izlazu daje IV i WEP ključ kojim treba kriptirati okvir. Standard definira S-kutiju koja se koristi u obje faze, te

6 Pošto je i IV različit za svaki okvir, TKIP zapravo generira različit RC4 ključ za svaki okvir.



Slika 2.18. TKIP: generiranje WEP ključa

procedure generiranja temeljene na jednostavnim operacijama nad bitovima i zbrajanju. Na taj način kriptiranje se bitno ne usporava u odnosu na klasičnu WEP enkripciju.



Slika 2.19. TKIP kriptirani okvir

Redni broj okvira prenosi se unutar zaglavlju okvira u poljima inicijalizacijskog vektora i proširenog inicijalizacijskog vektora (slika 2.19.) Zaštita od umetanja starih paketa (engl. replay) temelji se upravo na praćenju rednih brojeva okvira. Prijemnik pamti redni broj posljednjeg uspješno primljenog okvira. Kada primi sljedeći okvir odbacuje ga ukoliko mu redni broj nije veći od rednog broja posljednjeg uspješno primljenog okvira. MAC adresa koristi se da bi se u različitim smjerovima (prema i od pristupnoj točki) dobili različiti RC4 ključevi kada je i ključ i redni broj okvira jednak.

2.7.2.2. Zaštita integriteta i autentifikacija okvira

TKIP koristi Michael MIC (engl. *MIC - Message Integrity Code*) za zaštitu integriteta i autentičnosti okvira. Osim polja podataka, štite se i MAC adrese izvora i odredišta. Autentičnost se provjerava korištenjem MIC ključa koji je poznat samo klijentu i pristupnoj točki. Za razliku od CCMP-a gdje se umjesto ICV-a[7] koristi CBC-MAC, kod TKIP-a se koriste i MIC i ICV (slika 2.19.) ICV je ostavljen zbog kompatibilnosti s postojećim sklopovljem. MIC se koristi za sprječavanje mnogih napada koji su sa ICV-om mogući (npr. izmjene bitova podataka, zaglavlja okvira). Osim detekcije napada pomoću MIC-a, standard definira i protumjere. Protumjerama se nastoje ublažiti posljedice napada. Protumjere se iniciraju ukoliko između detekcije dviju greški u MIC sažetku prošlo manje od minute. Kada se detektira takva situacija pokreću se sljedeće akcije:

- korisniku se na neki način (dijalogom ili preko log datoteke) dojavljuje da je napad u tijeku,
- pristupnoj točki se pomoću posebnog EAPoL okvira dojavljuje greška,

7 ICV je WEP kriptirani CRC32

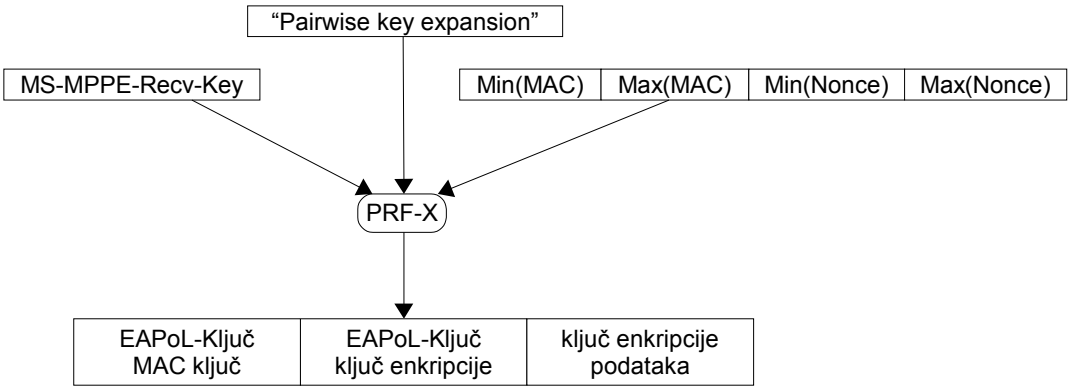
- zaustavlja se sva komunikacija do isteka minute od posljednje greške.

Na taj način se broj grešaka ograničava na maksimalno dvije u minuti. Protumjere imaju za cilj ograničiti broj izmijenjenih paketa koje će napadač imati priliku ubaciti u mrežu. Pristupna točka koji detektira ovakvu situaciju treba ponovno pokrenuti EAP autentifikaciju i time generirati nove ključeve enkripcije.

Slika 2.19. prikazuje tri zaštite integriteta okvira. Najslabija zaštita je CRC32 (ili FCS – engl. Frame Check Sequence) koji prvenstveno služi detekciji grešaka prilikom prijenosa. Nakon dekripcije tijela okvira, prvo se provjerava ICV, a tek ako ta provjera uspije, MIC. Prema tome greške koje ICV može detektirati neće pokrenuti protumjere. Zbog toga je detekcija greške u MIC sažetku vrlo vjerojatno posljedica napada (jer ICV nije detektirao grešku), pa su i oštre protumjere opravdane.

2.7.3. Razmjena ključeva

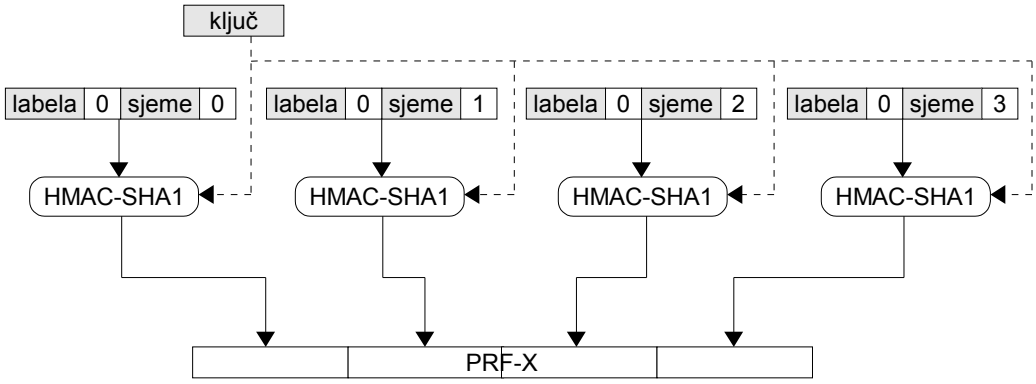
802.11i standard donosi promjene i na području razmjene ključeva. I dalje se kao osnova za generiranje ključeva koriste EAP metode, no 802.11i drugačije koristi EAP generirane ključeve. U 802.1X standardu pristupna točka je koristila EAP generirani ključ za enkripciju WEP ključa u EAPoL-Ključ okviru. U 802.11i standardu za razmjenu ključeva koriste se izmijenjeni EAPoL-Ključ okviri u kojima se ključ nikada ne prenosi eksplicitno. Na osnovu podataka izmijenjenih u EAPoL-Ključ okvirima pristupna točka i klijent generiraju iste ključeve. Generiranje ključeva prikazano je na slici 2.20.



Slika 2.20. Generiranje ključeva prema 802.11i standardu

MS-MPPE-Recv-Key je dijeljena tajna koju klijent i autentifikacijski server nezavisno generiraju nakon uspješno završene EAP autentifikacije (slika 2.12.) Autentifikacijski server izračunatu vrijednost prosljeđuje pristupnoj točki preko *MS-MPPE-Recv-Key* RADIUS atributa. Za računanje se, osim opisanog ključa i niza znakova, koristi i sjeme. Sjeme čine MAC adrese i jedinstveni slučajni brojevi koje klijent i pristupna razmjenjuju u protokolu od četiri poruke (opisan u nastavku). Zanimljivo je da sadržaj sjemena ovisi o vrijednostima MAC adrese i jedinstvenog slučajnog broja. Tako sjeme započinje manjom MAC adresom (promatra se kao binarni broj), nakon koje slijedi veća, bez obzira da li je to MAC adresa klijenta ili pristupne

točke. Isto vrijedi i za jedinstvene slučajne brojeve. Korištenje MAC adresa ima za posljedicu da se za različite klijente generira različiti ključ čak i ako je ključ *MS-MPPE-Recv-Key* jednak. Jedinstveni slučajni broj (engl. *Nonce*) ima ulogu generiranja različitog ključa čak i kad se koristi isti početni ključ između istih stanica (MAC adrese su jednake). Funkcija PRF-X (slika 2.21.) za generiranje ključeva slična je PRF funkciji iz TLS-a.



Slika 2.21. Računanje PRF-X funkcije

Umjesto računanja MAC sažetka u više razina, PRF-X koristi brojač - posljednji element ulaznog niza HMAC-SHA1 funkcije. Razlika je i u korištenoj funkciji sažetka – PRF-X koristi samo SHA1 sažetak. Računanje sažetka ponavlja se s uzlazno rastućim brojačem onoliko puta koliko je potrebno da bi se dobio niz željene dužine. Tako se niz od 512 bita dobiva računanjem 4 sažetka i odbacivanjem viška bitova. Ta se funkcija naziva PRF-512. Analogno se definiraju ostale PRF-X funkcije.

Niz izračunat PRF funkcijom dijeli se na tri dijela. Prvi dio čini ključ koji se koristi za MAC zaštitu EAPoL-Ključ okvira. Drugi dio koristi se za enkripciju pojedinih elemenata EAPoL-Ključ okvira. Ostatak niza koristi se kao ključ za kriptiranje podataka. Dužina i način korištenja tog ključa ovisi o korištenom algoritmu enkripcije. U slučaju CCMP-a 128 bita ostatka niza koristi se kao ključ. TKIP koristi 256 bita koji se dijele u tri dijela: prvih 128 bita koriste se kao ključ enkripcije (TK), sljedećih 64 bita koriste se kao MIC ključ u smjeru prema klijentu, dok se preostalih 64 bita koristi kao MIC ključ za okvire koje klijent generira.

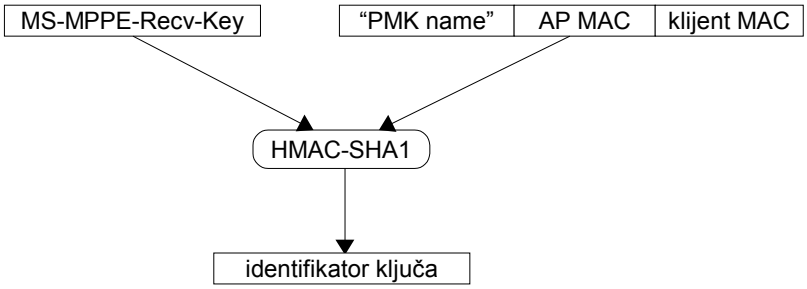
Protokol razmjene ključeva čini razmjena četiri EAPoL-Ključ okvira (engl. *4-Way Handshake*):

1. Pristupna točka generira i u prvoj poruci klijentu šalje jedinstveni slučajni broj (engl. *Nonce*). Okvir također sadrži identifikator ključa *MS-MPPE-Recv-Key*. Pomoću tog identifikatora klijent može provjeriti da li posjeduje isti ključ kao i pristupna točka. Identifikator ključa računa se kao što je prikazano na slici 2.22. Klijent može ponoviti postupak izračuna jer poznaje sve potrebne podatke. Ukoliko je primljeni identifikator jednak izračunatom, pristupna točka i klijent posjeduju jednaki ključ.
2. Klijent generira jedinstven slučajan broj i šalje ga pristupnoj točki. Taj okvir također sadrži i listu podržanih algoritama na strani klijenta. Pošto klijent sada ima oba jedinstvena slučajna broja on računa ključeve korištenjem

PRF-X funkcije (slika 2.20.) Izračunati EAPoL-Ključ MAC ključ koristi se za zaštitu okvira HMAC sažetkom.

3. Nakon primitka jedinstvenog slučajnog broja klijenta i pristupna točka može izračunati ključeve, te zaštititi treći okvir HMAC sažetkom. Treći okvir sadrži listu algoritama koje pristupna točka podržava. Ovaj okvir također sadrži redni broj posljednjeg grupnog okvira i kriptirani grupni ključ (koristi se za kriptiranje grupnih okvira[8] od pristupne točke prema klijentima). Grupni ključ kriptiran je EAPoL ključem enkripcije. Ovaj okvir ponovno sadrži jedinstveni slučajni broj servera jer prvi okvir nije bio zaštićen HMAC-om.
4. Posljednji okvir služi samo kao potvrda pristupnoj točki da je klijent ispravno primio prethodni okvir.

Pristupna točka i klijent razmjenjuju liste podržanih algoritama u HMAC zaštićenim okvirima da bi provjerili da li su prilikom povezivanja odabrani najbolji obostrano podržani algoritmi. Time se štite od napada u kojem napadač mijenja sadržaj okvir prilikom povezivanja i uzrokuje odabir slabijeg ili ranjivog algoritma. Nakon posljednjeg okvira klijent i pristupna točka mogu početi koristiti dogovorene ključeve enkripcije. Ukoliko pristupna točka promijeni grupni ključ, novu vrijednost može dostaviti klijentima protokolom za razmjenu grupnog ključa (engl. *Group Key Handshake*). Dotični protokol koristi dva EAPoL-Ključ okvira: prvi sadrži novi redni broj grupnih okvira i kriptirani novi grupni ključ; drugi okvir služi kao potvrda da je klijent primio novi grupni ključ.



Slika 2.22. Računanje identifikatora ključa

802.11i standard predviđa i korištenje statičkih ključeva (engl. *PSK - Preshared Key*). Statički ključevi zapravo predstavljaju 256 bitnu vrijednost *MS-MPPE-Recv-Key* atributa koja se inače generira uspješnom EAP autentifikacijom. Pošto statički ključevi ne koriste EAP autentifikaciju nije potrebno imati autentifikacijski server, ali je i ostvarena sigurnost manja. Ova mogućnost prvenstveno je namijenjena kućnim korisnicima. Osim unošenja 256 bita u heksadecimalnom obliku, predviđa se i korištenje lozinki. Iz lozinke i identifikatora mreže (engl. Service Set ID – SSID) poznatim se algoritmom dobiva 256 bitni ključ. Kao i uvijek, treba biti pažljiv u odabiru lozinki. Vrijednost identifikatora ključa ovisi o lozinki, identifikatoru mreže i MAC adresama pristupne točke i klijenta. Napadač može iz mrežnih paketa pročitati identifikator mreže, MAC adrese pristupne točke i klijenta, te sam

8 Grupni okviri (engl. *multicast*) nisu namjenjeni pojedinačnoj već svim stanicama (adresa odredišta je FF:FF:FF:FF:FF)

identifikator. Nakon toga može za niz slabih lozinki generirati identifikatore i usporediti ih s identifikatorom viđenim na mreži. Ukoliko se dogodi podudaranje, napadač je pronašao korištenu lozinku. Napadaču je posao otežan utoliko što ne može unaprijed izračunati parove lozinka – identifikator ključa jer identifikator ključa ovisi i o drugim parametrima.

U novije vrijeme često se spominje i sigurnosni standard pod nazivom WPA (engl. *Wireless Protected Access*). WPA specifikaciju donijeli su udruženi proizvođači bežične opreme zbog stalnih odgađanja izlaska 802.11i standarda. WPA specifikacija nije ništa drugo nego jedan od ranijih prijedloga (engl. *draft*) 802.11i standarda. Pošto WPA specifikacija nije slobodno dostupna, nije ju bilo moguće na ovom mjestu opisati, ali za pretpostaviti je da se vrlo malo razlikuje od upravo opisane 802.11i specifikacije.

3. Postupak otkrivanja WEP ključa

U opisu postupka otkrivanja WEP ključa, zbog ograničenog prostora, za dužinu riječi podatka *n* uzima se 3 bita (iz toga slijedi broj različitih vrijednosti riječi *N*=8). Prikazano je otkrivanje samo prve riječi ključa (koji je dug 3 riječi). Inicijalizacijski vektor dug je 3 riječi. Ovo pojednostavljenje ne oduzima na općenitosti opisa, te je proširenje na 8 bitnu riječ (tj. bajt) i ključeve dužine 5 (odnosno 13) riječi prirodno.

Kada predajnik treba poslati okvir kriptiran RC4 algoritmom on generira niz pseudoslučajnih riječi ovisnih o ključu (poglavlje 2.). Pretpostavimo da WEP ključ čine tri riječi: 2, 7, 5. Te su riječi poznate prijemnoj i predajnoj strani, ali ne i napadaču. Pretpostavimo da predajna strana koristi sljedeći incijalizacijski vektor: 3, 7, 2. Postupak inicijalizacije S-kutije KSA algoritmom prikazan je u tablici 3.1. U prvom koraku vrijednost brojača *i* je 0, dok se za brojač *j* dobiva vrijednost 3. Prvi i četvrti element S-kutije zatim se zamjenjuju. U drugom koraku brojač *i* se povećava za jedan, dok se za *j* ponovno dobiva vrijednost 3. Slijedi zamjena drugog i četvrtog elementa S-kutije. Izvođenje ostalih koraka provodi se na analogan način. Posljednji redak tablice prikazuje ključem inicijaliziranu S-kutiju.

Tablica 3.1. Predajnik: izvođenje KSA algoritma

K (RC4 ključ)									
IV			***WEP ključ***						
3	***7***	***2***	***2***	***7***	***5***				
S-kutija								***i***	***j=j+S[i]+K[i]***
0	1	2	3	4	5	6	7	0	0+0+3=3
3	1	2	0	4	5	6	7	1	3+1+7=11=3
3	0	2	1	4	5	6	7	2	3+2+2=7
3	0	7	1	4	5	6	2	3	7+1+2=2
3	0	1	7	4	5	6	2	4	2+4+7=13=5
3	0	1	7	5	4	6	2	5	5+4+5=14=6
3	0	1	7	5	6	4	2	6	6+4+3=13=5
3	0	1	7	5	4	6	2	7	5+2+7=6
3	0	1	7	5	4	2	6		

Generiranje prve riječi pseudoslučajnog niza (*z*) izvodi se po PRGA algoritmu koji koristi inicijaliziranu S-kutiju:

```
PRGA(S):
   Inicijalizacija:
      i = 0
      j = 0
   Generiranje prve riječi pseudoslučajnog niza:
      i = i + 1 = 0 + 1 = 1
      j = j + S[i] = 0 + S[1] = S[1] = 0
      zamijeni(S[i], S[j]) = zamijeni(S[1], S[0])
      generiraj z = S[S[i] + S[j]] = S[S[1] + S[0]] =
                  = S[3+0] = S[3] = 7
```

Prva generirana riječ *z* biti će uvijek jednaka:

$$z = S[S[i] + S[j]] = S[S[1] + S[S[1]]] \quad (1)$$

jer je *i* u prvom koraku uvijek 1, dok je *j* u prvom koraku jednak 0+S[i] = S[1]. U ovom primjeru dobiva se *z*=7. Ta se vrijednost XOR operacijom miješa sa prvom riječi podataka i daje prvu kriptiranu riječ. Na prijemnoj strani (koja zna tajni WEP ključ) na jednak način se inicijalizira S-kutija i generira ista prva riječ pseudoslučajnog niza. Ako je prva riječ podataka 4, prva kriptirana riječ biti će 4 XOR 7 = 3. Na prijemnoj strani primljena kriptirana riječ 3 miješa se s prvim bajtom pseudoslučajnog niza 7 i dobiva prva riječ podataka: 3 XOR 7 = 4. Napadač koji promatra ovu komunikaciju iz uhvaćenog okvira može pročitati sljedeće: korišteni inicijalizacijski vektor je (3, 7, 2), prva kriptirana riječ je 3. Pretpostavimo[1] da napadač poznaje prvu riječ podataka (4). Napadač tada može odrediti prvu riječ pseudoslučajnog niza *z* = 3 XOR 4 = 7. Napadač zatim izvodi KSA algoritam kao što je prikazano u tablici 3.2. Napadač može izvesti prva tri koraka KSA algoritma jer prve tri riječi RC4 ključa čini inicijalizacijski vektor.

Tablica 3.2. Napadač: izvođenje KSA algoritma

K									
IV			***WEP ključ***						
3	***7***	***2***	***?***	***?***	***?***				
S-kutija								***i***	***j=j+S[i]+K[i]***
0	1	2	3	4	5	6	7	0	0+0+3=3
3	1	2	0	4	5	6	7	1	3+1+7=11=3
3	0	2	1	4	5	6	7	2	3+2+2=7
3	0	7	1	4	5	6	2	3	7+1+K[3]=j_4

U četvrtom koraku ne može izračunati vrijednost $j=j_4$ jer ne poznaje prvu riječ WEP ključa (*K[3]*). No napadač može pretpostaviti sljedeće: neka je u četvrtom koraku predajnik izračunao $j=j_4$. Tada će u četvrtom koraku element $S[i_4]=S[3]$ biti zamijenjen sa nepoznatim elementom $S[j_4]$. Drugim riječima, vrijednost *1* iz S-kutije zamijeniti će mjesto s nekom nepoznatom vrijednošću. Napadač dalje pretpostavlja da ukoliko do kraja KSA algoritma ne budu zamijenjeni elementi na pozicijama 0, 1 i 3, prva riječ pseudoslučajnog niza *z* biti će upravo nepoznata vrijednost jer se ona nakon zamjene u četvrtom koraku nalazi na poziciji *S[3]*:

$$z = S[S[1] + S[S[1]]] = S[S[1] + S[0]] = S[3 + 0] = S[3]$$

Napadač poznaje prvu riječ pseudoslučajnog niza (*z*=7), pa zna da je nepoznata vrijednost upravo 7. Preostaje mu pretražiti S-kutiju prije zamjene u četvrtom koraku i pronaći poziciju j_4 na kojoj se nalazi vrijednost 7. Napadač tako određuje da je $j_4 = 2$. Nakon toga u formuli za j_4 ostaje samo jedna nepoznanica – prva riječ WEP ključa *K[3]*. Iz jednadžbe 7 + 1 + *K[3]*=2 napadač određuje da je *K[3]* = 2. Napadač je upravo odredio prvu riječ tajnog

1 Ova pretpostavka ima smisla i u stvarnosti jer je prva riječ podataka redovito upravo prvi bajt LLC zaglavlja koje na IP mrežama uvijek ima vrijednost 0xAA heksadecimalno.

ključa. Uz poznavanje prve riječi tajnog ključa moguće je izvesti jedan korak više KSA algoritma i na analogan način odrediti drugu i sve ostale riječi WEP ključa.

Pri tome je bitno odabrati pogodan inicijalizacijski vektor koji će u S-kutiji u elemente *S[S[1]]* i *S[1]* postaviti takve vrijednosti da prva vrijednost pseudoslučajnog niza bude *z = S[S[1] + S[S[1]]] = S[B]* gdje je *B* indeks riječi u RC4 ključu koju tražimo. Za S-kutiju za koju vrijedi *S[S[1] + S[S[1]]] = S[B]* kažemo da se nalazi u razriješenom (engl. *resolved*) stanju. Inicijalizacijske vektore koji postavljaju S-kutiju u razriješeno stanje nazivamo ranjivima. U prethodnom primjeru prva riječ WEP ključa nalazila se na četvrtoj poziciji u RC4 ključu pa je *B*=3. Za slijedeću riječ WEP ključa B bi morao biti 4 itd. Implementacije ovog napada obično koriste samo poznate razrede ranjivih inicijalizacijskih vektora. Pristup koji je korišten u izradi programa opisanog u poglavlju 4. je da se koriste svi inicijalizacijski vektori. Za svaki inicijalizacijski vektor provodi se KSA algoritam koliko je to moguće (ovisno koja se riječ ključa traži) i zatim se pogleda dali je S-kutija u razriješenom stanju. Ako je razriješeno stanje postignuto, određuje se vrijednost riječi ključa. U suprotnom, kada razriješeno stanje nije postignuto, riječ ključa se ne može otkriti i inicijalizacijski vektor ostaje neiskorišten. Kod viših riječi ključa za postizanje razriješenog stanja koriste se prema tome i niže, prethodno otkrivene, riječi WEP ključa.

Opisani postupak otkrivanja WEP ključa pretpostavio je da u nastavku (nakon četvrtog koraka) KSA algoritma neće doći do zamjene bilo koje od vrijednosti na pozicijama 0, 1 i 3. Ako se zamjena ne dogodi napadač će otkriti vrijednost jedne riječi ključa. No ukoliko se zamjena dogodi napadač će izračunati pogrešnu vrijednost ključa kao što prikazuje sljedeći primjer (tablica 3.3.) U odnosu na prethodni primjer inicijalizacijski vektor razlikuje se samo u posljednjoj riječi.

Tablica 3.3. Predajnik: izvođenje KSA algoritma

K									
IV			***WEP ključ***						
3	***7***	***5***	***2***	***7***	***5***				
S-kutija								***i***	***j=j+S[i]+K[i]***
0	1	2	3	4	5	6	7	0	0+0+3=3
3	1	2	0	4	5	6	7	1	3+1+7=11=3
3	0	2	1	4	5	6	7	2	3+2+5=10=2
3	0	2	1	4	5	6	7	3	2+1+2=5
3	0	2	5	4	1	6	7	4	5+4+7=16=0
4	0	2	5	3	1	6	7	5	0+1+5=6
4	0	2	5	3	6	1	7	6	6+1+3=10=2
4	0	1	5	3	6	2	7	7	2+7+7=16=0
7	0	1	5	3	6	2	4		

Vrijednost na poziciji 0 zamijenjena je nakon četvrtog koraka dva puta. Prva riječ pseudoslučajnog niza biti će:

$$z = S[S[1] + S[S[1]]] = S[0+7] = S[7] = 4.$$

Da zamjene elementa na poziciji *0* nije došlo prva vrijednost pseudoslučajnog niza bila bi *z*=5. Napadač bi iz toga mogao zaključiti da je j_4=5, te iz jednadžbe 2+1+*K[3]* = 5 ispravno zaključiti da je prva riječ ključa jednaka *2*. No pošto se zamjena dogodila i prva riječ niza je *z*=4, napadač zaključuje da je j_4=4 (tablica 3.4.), te iz jednadžbe 2+1+*K[3]* = 4 dobiva 1 za vrijednost prve riječi WEP ključa što je pogrešan rezultat. Napadač ne može razlikovati ova dva slučaja jer on ne zna što se dešava u algoritmu nakon četvrtog koraka. Može se zaključiti da inicijalizacijski vektori koji postavljaju S-kutiju u razriješeno stanje mogu, ali ne moraju otkriti napadaču vrijednost promatrane riječi WEP ključa.

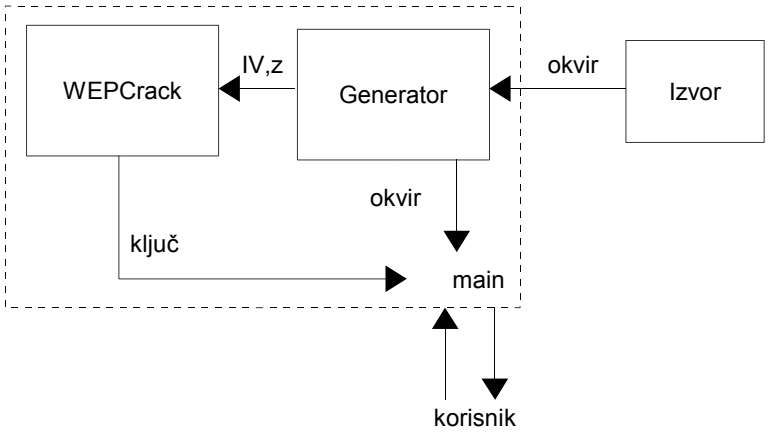
Tablica 3.4. Napadač: izvođenje KSA algoritma

K									
IV			***WEP ključ***						
3	***7***	***5***	***?***	***?***	***?***				
S-kutija								***i***	***j=j+S[i]+K[i]***
0	1	2	3	4	5	6	7	0	0+0+3=3
3	1	2	0	4	5	6	7	1	3+1+7=11=3
3	0	2	1	4	5	6	7	2	3+2+5=10=2
3	0	2	1	4	5	6	7	3	2+1+K[3]=j_4

Izvorni tekst [1] koji opisuje ovaj napad navodi vjerojatnost od oko 5% da do zamjene promatranih pozicija u S-kutiji neće doći te će napadač izračunati ispravnu vrijednost jedne riječi ključa. U ostalih 95% slučajeva do zamjene će doći i izračunata vrijednost riječi ključa može se promatrati kao slučajna varijabla sa uniformnom razdiobom (sve vrijednosti su podjednako vjerojatne). Uz 8 bitnu riječ, od 100 inicijalizacijskih vektora koji postavljaju S-kutiju u razriješeno stanje, oko 5 inicijalizacijskih vektora dati će ispravnu vrijednost promatrane riječi ključa, dok će preostalih 95 inicijalizacijskih vektora najvjerojatnije dati 95 različitih vrijednosti. No i dalje će se ispravna vrijednost riječi ključa pojavljivati češće (5 puta) nego ostale moguće vrijednosti (1 puta). Dakle, iako pojedini slabi inicijalizacijski vektori neće uvijek dati ispravnu vrijednost za određenu riječ ključa, vrijednosti koje se najčešće pojavljuju vjerojatno predstavljaju pravu vrijednost tražene riječi WEP ključa.

4. Opis programske implementacije

4.1. Uvod



Slika 4.1. Osnovna struktura programa *WepCrack+*

Program *WepCrack+*[1] opisan u ovom poglavlju služi za otkrivanje WEP ključa postupkom iz poglavlja 3. Osnovna struktura programa prikazana je na slici 4.1. Središnji dio programa čini modul pod nazivom *WEPCrack*. On sadrži većinu programskog kôda vezanog uz otkrivanje WEP ključa. Ulaz u *WEPCrack* modul čine parovi (inicijalizacijski vektor, prva riječ pseudoslučajnog niza). Pomoću tih parova *WEPCrack* modul pokušava pronaći korišteni WEP ključ. Izlazne vrijednosti *WEPCrack* modula su moguće vrijednosti WEP ključa. Inicijalizacijski vektori koji koje koristi *WEPCrack* modul nastaju u modulu nazvanom *Generator*. Modul *Generator* predstavlja apstraktno sučelje prema različitim mogućim generatorima okvira. Korišteni generatori okvira mogu se podijeliti u dvije grupe:

- generator stvarnih okvira,
- programski generator okvira.

Generator stvarnih okvira predstavlja sučelje prema izvoru okvira. Izvor okvira može biti:

- mrežna kartica u nadzornom (engl. *monitor*) načinu rada,
- datoteka u kojoj su sačuvani okviri.

Mrežna kartica koja se nalazi u nadzornom načinu rada omogućava promatranje svog mrežnog prometa. Kartica pri tome ne može slati okvire, ali prima okvire koje druge stanice izmjenjuju u komunikaciji. Na ovaj način moguće je u realnom vremenu pregledavati okvire i pronaći korišteni ključ. Ukoliko to zbog nekog razloga nije prihvatljivo, okvire je moguće spremiti u datoteku i naknadno (engl. *offline*) odrediti korišteni ključ, a zatim ih dekriptirati. Okviri se obično spremaju korištenjem *tcpdump* programa u datoteku poznatog formata.

1 Ime programa je odabrano tako da se razlikuje od postojeće WepCrack *perl* skripte iste namjene, te da se naznače poboljšanja u algoritmu.

Programski generator okvira ne daje na svom izlazu stvarne okvire već programski generirane (korištenjem sustavskog generatora slučajnih brojeva). Ovaj tip generatora koristi se prije svega za testiranje i demonstraciju rada programa kada ostali tipovi generatora nisu dostupni.

Modul *Generator* može na svom izlazu davati ili cijele okvire ili iz okvira pročitan par (IV, prva riječ pseudoslučajnog niza). Cijele okvire koristi glavni program za ispitivanje da li je pronađeni ključ ispravan, dok parove (IV, prva riječ pseudoslučajnog niza) koristi *WEPCrack* modul za otkrivanje WEP ključa.

Glavni program (*main*) povezuje opisane module. Na osnovu parametara primljenih od korisnika, glavni program pokreće *WEPCrack* modul, inicijalizira odgovarajući tip *Generatora* i povezuje ga sa pokrenutim *WEPCrack* modulom. Glavni program zatim provjerava potencijalne WEP ključeve tako da pomoću njih pokuša dekriptirati okvir. Ukoliko je dekriptiranje uspješno, ključ je pronađen, njegova vrijednost se prikazuje korisniku i program završava. U nastavku poglavlja detaljnije će biti opisani pojedini moduli.

4.2. Otkrivanje ključa enkripcije

Otkrivanje ključa izvodi se u modulu *WEPCrack*. Modul *WEPCrack* definiran je u datotekama *WEPCrack.cc* i *WEPCrack.h*. Čine ga dva razreda: *Monitor* i *CrackThread*. Razred *CrackThread* služi za otkrivanje vrijednosti određene riječi WEP ključa. Razred *Monitor* upravlja otkrivanjem WEP ključa tako da stvara objekte razreda *CrackThread*, nadzire njihov rad i uništava ih. Paralelno izvođenje objekata ovih razreda postignuto je korištenjem dretvi. *WEPCrack* modul pokreće se stvaranjem objekta razreda *Monitor*. Konstruktor klase *Monitor* (slika 4.2.) prima parametre izvođenja od glavnog programa. Značenje pojedinih parametara prikazano je u tablici 4.1.

Prva tri parametra određuju rad WEP algoritma. Preostali parametri određuju rad nadzornog algoritma. Nadzorni algoritam započinje tako da stvori jedan objekt razreda *CrackThread*. Taj objekt u posebnoj dretvi pokušava odrediti vrijednost prve riječi WEP ključa. Nadzorni algoritam periodički provjerava stanje stvorenog objekta. Kada primijeti da se neka od mogućih vrijednosti za prvu riječ ključa pojavljuje češće od ostalih, nadzorni algoritam može pokrenuti novu dretvu. Nova dretva određivat će vrijednost druge riječi ključa uz pretpostavku da je prva riječ ključa poznata i jednaka upravo najčešće pojavljivanoj vrijednosti. Pokretanje nove dretve određeno je prije svega parametrom *forkAt*. Ta vrijednost određuje koliko se minimalno puta određena vrijednost prve riječi ključa mora pojaviti da bi bila kandidat za pokretanje nove dretve. Ako je, na primjer, vrijednost parametra *forkAt* postavljena na 3, nova dretva se neće moći pokrenuti za neku vrijednost prve riječi ključa sve dok se ta vrijednost ne pojavi barem tri puta.

Tablica 4.1. Parametri konstruktora razreda Monitor

Naziv parametra	***Opis***
I	Dužina inicijalizacijskog vektora u riječima
I	Dužina WEP ključa u riječima
N	Broj različitih vrijednosti za pojedinu riječ ključa $(2^n)^2$
topN	Za koliko najvjerojatnijih vrijednosti riječi ključa se pokreću nove dretve
depth	Za koliko početnih riječi ključa se pokreće topN dretvi
forkAt	Minimalni broj pogodaka potrebnih za pokretanje nove dretve
topNhi	Za koliko najvjerojatnijih vrijednosti viših riječi ključa se pokreću nove dretve
loadlimit	Broj dretvi koje čekaju u redu aktivnih dretvi nakon kojeg treba zaustaviti pokretanje novih dretvi
debug_level	Razina ispisa za razred *Monitor*
crack_debug_level	Razina ispisa za razred *CrackThread*

Nadzorni algoritam može pokrenuti i više od jedne dretve ako se više različitih vrijednosti za prvu riječ ključa često pojavljuje. Ukoliko se, na primjer, dvije različite vrijednosti (npr. 0xd3 i 0x42) pojavljuju više od tri puta (i *forkAt=3*), nadzorni algoritam može pokrenuti dvije nove dretve. Nove dretve računati će vrijednost druge riječi ključa. Jedna dretva pretpostaviti će vrijednost prve riječi ključa 0xd3, dok će druga pretpostaviti vrijednost 0x42. Maksimalni broj dretvi koje nadzorni algoritam može pokrenuti za najčešće vrijednosti prve riječi ključa određen je parametrom *topN*. Ukoliko je, na primjer, *topN* postavljen na 3, tada će biti pokrenute najviše tri dretve za tri najvjerojatnije vrijednosti prve riječi ključa. Pokretanje dretvi za više najvjerojatnijih vrijednosti riječi ključa može se primijeniti i na ostale riječi ključa. Do koje će riječi ključa ovo pravilo vrijediti određeno je parametrom *depth*. Ukoliko je, na primjer, *depth=3* tada će se pravilo pokretana novih dretvi za *topN* najvjerojatnijih vrijednosti ključa primjenjivati za prvu, drugu i treću riječ ključa. Za preostale riječi pokretat će se *topNhi* dretvi i to za *topNhi* najvjerojatnijih vrijednosti promatrane riječi. Na pokretanje novih dretvi utječe i vrijednost parametra *loadlimit*. Taj parametar treba služiti da bi se ograničio broj ukupno pokrenutih dretvi. Prije pokretanja nove dretve pozivom funkcije *getloadavg* dobiva se informacija o prosječnom broju dretvi koje su se nalazile u redu aktivnih dretvi u protekloj minuti. Ako je dobivena vrijednost manja od vrijednosti parametra *loadlimit*, nova dretva se pokreće. U suprotnom, pokretanje nove dretve se odgađa dok se broj aktivnih dretvi ne smanji. Parametri *debug_level* i *crack_debug_level* utječu na količinu informacija koju će razredi *Monitor* i *CrackThread* ispisivati tijekom izvođenja. U nastavku poglavlja detaljnije će biti opisani razredi *Monitor* i *CrackThread*.

2 *n* = broj bita u riječi

4.2.1. Nadzor nad otkrivanjem ključa

```
class Monitor {
  private:
     int I;
     int l;
     int N;
     list<WeakIV> IVs;

     int topN;
     int depth;
     int forkLimit;
     int topNhi;
     int loadlimit;
     list<vector<byte> > keys;

     CrackThread* root;
     multimap<CrackThread*, CrackThread*> threads;
     pthread_t monitor_thread;
     pthread_mutex_t keys_lock;

     DebugPrint dp;
     int crackDL;

     void monitor();
     void check_thread(CrackThread* thread, int level);
     void print_threads_recursive(CrackThread* thread, int level);
     void erase_thread_recursive(CrackThread*, CrackThread*);

  public:
     Monitor(int I, int l, int N, int topN, int depthM, int forkAt,
             int topNhi, int loadlimit,
             int debug_level, int crack_debug_level);
     ~Monitor();

     void addIV(WeakIV iv) { IVs.push_back(iv); }
     bool getNextKey(vector<byte> &key);

     friend void* MonitorThreadStart(void* obj);
};
```

Slika 4.2. Deklaracija razreda Monitor

Slika 4.2. prikazuje deklaraciju razreda *Monitor*. Deklaracija sadrži prije opisani konstruktor te niz privatnih članskih varijabli koje čuvaju vrijednosti parametara konstruktora. Razred sadrži i dvije liste: listu inicijalizacijskih vektora (*IVs*) i listu pronađenih (mogućih) WEP ključeva (*keys*). Za ostvarenje strukture podataka liste koristi se implementacija iz STL-a[3] temeljena na predlošcima. Lista *IVs* sadrži strukture *WeakIV* (slika 4.3.)

```
struct WeakIV {
   vector<byte> IV;
   byte Z;
};
```

Slika 4.3. Deklaracija strukture WeakIV

Struktura *WeakIV* sadrži informacije koje su potrebne za otkrivanje jedne moguće vrijednosti određene riječi ključa: inicijalizacijski vektor (*IV*) i vrijednost prve riječi pseudoslučajnog RC4 niza (*Z*). Strukture *WeakIV* predstavljaju ulaz u modul *WEPCrack*, tj. razred *Monitor*. Lista *IVs* čuva sve

3 STL (Standard Templates Library) je standardna bibilioteka C++ predložaka koja sadrži implementacije mnogih često korištenih struktura podataka i algoritama

vrijednosti struktura koje su razredu poslane. Čuvanje svih *WeakIV* struktura potrebno je jer novostvorene *CrackThread* dretve koriste listu od početka. Dodavanje strukture u listu moguće je pozivanjem *addIV* metode. Metoda dodaje novu *WeakIV* strukturu na kraj liste. Kao rezultat rada *WEPCrack* modula dobiva se niz mogućih vrijednosti ključa. Te vrijednosti nalaze se u listi *keys*. Elementi liste su vektori bajtova. Vektor je struktura podataka definirana u STL-u[18] koja se ponaša kao polje promjenjive veličine. Vrijednosti ključeva moguće je izvan klase dohvatiti metodom *getNextKey*. Ta metoda vadi prvi element iz liste *keys* i vraća ga preko parametra *key*. Pristup listi *keys* iz više dretvi radi dodavanja ili čitanja ključeva moguć je zbog korištenja međusobnog isključivanja. Međusobno isključivanje ostvareno je korištenjem članske varijable *keys_lock* i POSIX funkcija *pthread_mutex_lock* i *pthread_mutex_unlock*. Razred *DebugPrint* implementira ispis u više razina, dok se u varijabli *crackDL* čuva vrijednost razine ispisa za *CrackThread* objekte. Ista im se prosljeđuje prilikom stvaranja.

Preostale metode i članske varijable implementiraju prije opisani nadzorni algoritam. Izvršavanje tog algoritma započinje stvaranjem objekta razreda *Monitor*, tj. pozivom konstruktora. Konstruktor (slika 4.4.), osim inicijaliziranja članskih varijabli, stvara *CrackThread* dretvu za prvu riječ ključa, te dretvu koja će izvršavati nadzorni algoritam. Dretva nadzornog algoritma započinje izvršavanje funkcijom *MonitorThreadStart*. *MonitorThreadStart* funkciji kao parametar prosljeđuje se pokazivač na *Monitor* objekt. Preko tog pokazivača poziva se *monitor* metoda objekta koja implementira nadzorni algoritam. Metoda *monitor* u beskonačnoj petlji poziva ispis i provjeru stanja pokrenutih *CrackThread* dretvi.

```
Monitor::Monitor(int I, int l, int N, int topN, int depth,
        int forkAt) {

   ...

   root = new CrackThread(I,l,N,0,vector<byte>(0),IVs);

   ::pthread_create(&monitor_thread, NULL,
      MonitorThreadStart, this);
};

void* MonitorThreadStart(void* obj) {
   Monitor* mon = (Monitor*) obj;

   mon->monitor();
}

void Monitor::monitor() {
   while (1) {
      print_threads_recursive(root, 0);
      check_thread(root, 0);

      sleep(5);
   }
}
```

Slika 4.4. Pokretanje nadzornog algoritma

Kada je nadzorni algoritam pokrenut aktivna je samo jedna *CrackThread* dretva pokrenuta iz konstruktora. Pokazivač na nju spremljen je

u *root* članskoj varijabli. Članska varijabla *monitor_thread* čuva identifikator dretve koja izvršava nadzorni algoritam. Taj identifikator koristit će se u destruktoru za zaustavljanje pripadne dretve. Nadzorni algoritam pokrenute *CrackThread* dretve organizira u stablo. Korijen stabla čini dretva koja određuje prvu riječ ključa – pokazivač na nju nalazi se u članskoj varijabli *root*. Nakon pokretanja dretvi koje određuju drugu riječ ključa, iste se dodaju u stablo kao djeca korijenskog elementa. Analogno, dretve koje određuju vrijednost treće riječi ključa dodaju se kao djeca dretvi koje određuju vrijednost druge riječi ključa itd. Opisano stablo implementirano je pomoću *multimap* strukture podataka u članskoj varijabli *threads*. *Multimap* struktura podataka obavlja pridruživanje 1:N, tj. jednoj dretvi pridružuje više drugih dretvi. Tako su korijenskoj dretvi (*root*) pridružene sve dretve koje određuju vrijednost druge riječi ključa – sva neposredna djeca iz stabla. Analogno vrijedi za ostale čvorove stabla, tj. dretve.

Nadzorni algoritam prolazi kroz ovako ostvareno stablo i nadzire rad pojedinih dretvi. Tako metoda *print_thread_recursive* započinje sa korijenskim elementom i rekurzivno ispisuje stanje svih dretvi sve do listova stabla. Prolazak kroz stablo obavlja se po dubini. Metoda *check_thread* također prolazi kroz stablo rekurzivnim obilaskom po dubini, ali pri tome obavlja pregled stanja svake pojedine dretve. Pregled stanja dretve podrazumijeva izračunavanje statistike, odnosno određivanje najvjerojatnijih vrijednosti za riječ ključa koju ta dretva određuje. Nakon toga se provjerava da li pokrenute dretve/djeca odgovaraju statistici i parametrima algoritma (*forkAt*, *topN*, *depth*, *topNhi*, *loadlimit*). Ukoliko je potrebno stvaraju se nove, odnosno uništavaju postojeće dretve. Uništavanje dretvi je potrebno ako se statistika promijenila od posljednje provjere. Ukoliko, na primjer, parametri dopuštaju pokretanje samo jedne dretve djeteta, i to za najvjerojatniju vrijednost promatrane riječi ključa, te se najvjerojatnija vrijednost promijenila od posljednje provjere, potrebno je uništiti postojeću dretvu dijete i stvoriti novu koja će pretpostaviti novu najvjerojatniju vrijednost određene riječi ključa. Prilikom uništavanja dretvi potrebno je uništiti i svu njihovu djecu – za to služi metoda *erase_thread_recursive*.

Destruktor razreda *Monitor* obavlja uništenje svih pokrenutih dretvi. Prvo se uništava dretva nadzornog algoritma, a zatim sve dretve/djeca sadržana u stablu *threads*. Posljednji se uništava korijen stabla dretvi sadržan u članskoj varijabli *root*. Na taj način uništenje objekta razreda *Monitor* povlači i uništenje svih dretvi koje je taj objekt stvorio.

4.2.2. Otkrivanje pojedinih riječi ključa

Funkcija razreda *CrackThread* je otkrivanje određene riječi ključa. Pri tome se pretpostavlja da su poznate riječi ključa niže od one koja se traži. Deklaracija razreda prikazana je na slici 4.5.

Osim uobičajenih parametara RC4 algoritma (*I*, *l* i *N*), razred sadrži i člansku varijablu *B*. Ona čuva redni broj riječi ključa koju taj objekt traži. Tako je za dretvu koja određuje prvu riječ ključa B=0. Za dretve koje određuju drugu riječ ključa B=1 itd. U članskoj varijabli *key* spremljene su vrijednosti riječi ključa nižih od *B*. Element *IVs* je referenca na listu inicijalizacijskih vektora koji se koriste za određivanje ključa. Pošto se koristi referenca, svi

```
class CrackThread {
   private:
      const int I;
      const int l;
      const int N;
      const int B;
      vector<int> stats;
      const vector<byte> key;
      const list<WeakIV>& IVs;
      pthread_t thread;

      int hits;
      int hitsFMS;
      int processed;
      DebugPrint dp;

      void crack();
      int crackByte(byte* K, byte Z);

   public:
      CrackThread(int I, int l, int N, int B, const vector<byte>& key,
               const list<WeakIV>& ivs);
      ~CrackThread();

      int getB() { return B; }
      int getHits() { return hits; }
      int getProcessed() { return processed; }
      const vector<int>& getStats() { return stats; }
      const vector<byte>& getKey() { return key; }

      friend void* CrackThreadStart(void* obj);
};
```

Slika 4.5. Deklaracija razreda CrackThread

objekti *CrackThread* razreda dijele istu listu, čime se štedi memorija. Ključnom riječi *const* pristup listi ograničava se samo na čitanje. Izmjene nad listom radi samo razred *Monitor* i to isključivo dodavanjem elemenata na kraj liste. Prilikom određivanja tražene riječi ključa učestalost pojavljivanja pojedinih vrijednosti sprema se u vektor *stats*. Vektor je dužine *N* jer čuva učestalost pojavljivanja za vrijednosti od 0 do N-1. Izvan razreda moguće ga je dohvatiti metodom *getStats*. Tijekom izvođenja algoritma za otkrivanje riječi ključa, u varijabli *processed* sakuplja se statistika o broju obrađenih okvira. Varijable *hits* i *hitsFMS* sadrže broj inicijalizacijskih vektora koji su dali korisnu informaciju o promatranoj riječi ključa ako se koriste svi odnosno *AirSnort*-u poznati inicijalizacijski vektori.

Izvođenje algoritma za pronalaženje riječi ključa obavlja se u zasebnoj dretvi. Dretva se pokreće iz konstruktora (slično kao kod razreda *Monitor*) preko funkcije *CrackThreadStart*. Ta funkcija naposljetku izvršava metodu *crack* koja sadrži sam algoritam. Da bi se stvorenu dretvu moglo zaustaviti iz destruktora, identifikator dretve sprema se u članskoj varijabli *thread*. Metoda *crack* prolazi kroz listu *IVs* i za svaki element liste poziva metodu *crackByte*. Ukoliko je lista *IVs* prazna ili su pregledali svi trenutno sadržani elementi, izvršavanje dretve se privremeno zaustavlja pozivom funkcije *sleep*. Metoda *crackByte* prikazana je na slici 4.6.

```
int CrackThread::crackByte(byte* K, byte Z) {
   int i, j, k, KB;
   int S[N];

   for(i=0; i<N; i++) // init S-box
      S[i]=i;
   j=0;

   for(i=0; i<I+B; i++) { // do KSA until byte B
      j = j + S[i] + K[i%(I+l)];
      j %= N;
      swap(S[i], S[j]);
   }

   if ( S[1]>=i || (S[1] + S[S[1]])%N != i ) // S[1] got messed up
      return -1;

   k=find(S, S+N, Z)-S;
   assert(k<N);

   KB = k - (j + S[i]);
   while(KB<0)
      KB+=N;

   return KB;
}
```

Slika 4.6. Metoda crackByte

Metoda *crackByte* kao parametre prima poznati dio RC4 ključa (*K*) koji se sastoji od inicijalizacijskog vektora i već otkrivenih riječi WEP ključa (iz članske varijable *key*). Drugi parametar predstavlja prvi bajt pseudoslučajnog niza (*Z*) – sadržan je u svakom elementu liste *IVs*. Metoda započinje inicijalizacijom S-kutije i izvođenjem RC4 KSA algoritma sve dok se ne stigne do nepoznate riječi WEP ključa (sa rednim brojem *B*). Ukoliko je nakon toga vrijednost S[1] elementa pogodna, nastavlja se s određivanjem moguće vrijednosti tražene riječi ključa. Prvo se korištenjem STL funkcije *find* pronalazi pozicija *k* elementa *Z* u S-kutiji. Slijedi izračunavanje vrijednosti tražene riječi ključa *KB* pomoću formule:

$$k = j + S[i] + KB$$

odnosno

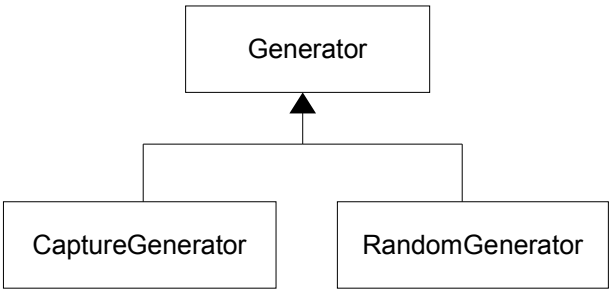
$$KB = k – j – S[i]$$

U navedenim formulama *k* predstavlja vrijednost brojača *j* u sljedećem koraku, dok je *KB* vrijednost riječi ključa na poziciji *B*. Dobivena vrijednost riječi ključa zatim se svodi na područje vrijednosti od 0 do N-1.

4.3. Generiranje okvira

Modul *Generator* čine tri razreda: *Generator*, *CaptureGenerator* i *RandomGenerator* definirana u datotekama *Generator.h* i *Generator.cc*. Razred *Generator* je osnovni dok su preostala dva iz njega izvedeni razredi (slika 4.7.) Svrha razreda *Generator* je definiranje sučelja koje će izvedeni razredi na različit način implementirati. Sučelje (slika 4.8.) čine svega dvije metode: *getWeakIV* i *getPacket* koje generiraju inicijalizacijski vektor

odnosno paket. Metoda *extractIV* implementirana je u osnovnom razredu i služi za određivanje inicijalizacijskog vektora iz okvira.



Slika 4.7. Hijerarhija razreda u modulu Generator

Metoda *getWeakIV* služi za generiranje parova (inicijalizacijski vektor, prva riječ RC4 pseudoslučajnog niza) opisanim *WeakIV* tipom podataka. Metodi *getWeakIV* se prosljeđuje referenca na *WeakIV* strukturu koju metoda zatim popunjava. Povratna vrijednost metode je tipa *bool* za slučaj kada metoda zbog nekog razloga ne može generirati podatke. Tada se kao povratnu vrijednost vraća konstanta *false*. Metoda *getPacket* razlikuje se utoliko što koristi STL tip podataka *vector* i vraća cijeli okvir podataka. Ova mogućnost se koristi kada je potrebno ispitati da li je neki od pronađenih ključeva kriptiranja ispravan. Tada se metodom *getPacket* dohvaća cijeli okvir i pokušava ga se dekriptirati pronađenim ključem.

```
class Generator {
   public:
      virtual bool getWeakIV(WeakIV&) = 0;
      virtual bool getPacket(vector<byte>&) = 0;
      bool extractIV(const vector<byte>&, WeakIV&);
};
```

Slika 4.8. Deklaracija razreda Generator

4.3.1. Dohvat okvira s mrežne kartice

Razred *CaptureGenerator* sučelje *Generator* implementira korištenjem *libpcap* biblioteke (engl. *packet capture*). Biblioteka ima mogućnost snimanja okvira u realnom vremenu (sa mrežne kartice) ili čitanja prethodno snimljenih okvira iz datoteke. Na *libpcap* biblioteci temeljeni su i najpoznatiji slobodni programi za nadzor mrežnog prometa: *tcpdump* i *ethereal*.

```
class CaptureGenerator: public Generator {
   private:
      pcap_t* handle;

      bool checkMAC(const char* mac);
   public:
      CaptureGenerator(const char* where, const char* mac,
         int keyid, bool live);
      ~CaptureGenerator();

      bool getWeakIV(WeakIV&);
      bool getPacket(vector<byte>&);

};
```

Slika 4.9. Deklaracija razreda CaptureGenerator

Parametar *live* konstruktora (slika 4.9.) određuje da li će se okviri dohvaćati izravno s mrežne kartice (vrijednost parametra *true*) ili čitati iz datoteke (vrijednost parametra *false*). Kada se okviri dohvaćaju s mrežne kartice tada parametar *where* sadrži naziv mrežnog sučelja pridruženog mrežnoj kartici. Ako se okviri čitaju iz datoteke tada parametar *where* sadrži naziv datoteke. Preostala dva parametra predstavljaju ograničenja na dohvaćene okvire. Parametar *mac* ograničava dohvat samo na okvire čiji je izvor ili odredište definirano navedeno MAC adresom. Parametar *keyid* ograničava dohvat samo na okvire kriptirane ključem čiji je identifikator jednak vrijednosti parametra. Konstruktor na osnovu parametara inicijalizira dohvat okvira pozivom odgovarajuće funkcije *libpcap* biblioteke. Ručica koja će se nadalje koristiti za dohvat okvira sprema se u člansku varijablu *handle*. Osim inicijalizacije biblioteke, konstruktor postavlja i filter na primljene okvire. Filter se formira tako da biblioteka vraća samo podatkovne okvire kriptirane ključem definiranog identifikatora i čija je adresa izvora ili odredišta jednaka navedenoj MAC adresi. Ukoliko se u bilo kojem od navedenih koraka dogodi pogreška ili je vrijednost nekog od parametara neispravna, konstruktor generira iznimku. Primjeri mogućih tipova iznimaka su razredi *BadCaptureSource*, *BadMACAddress* itd. definirani u datoteci *Generator.h*.

Metoda *getWeakIV* ostvarena je pozivom funkcije *pcap_next* iz *libpcap* biblioteke. Navedena funkcija vraća sljedeći okvir koji zadovoljava postavljene uvjete filtriranja. Metoda *getWeakIV* iz okvira čita zanimljive podatke: inicijalizacijski vektor i prvi bajt kriptiranih podataka. Inicijalizacijski vektor se neizmijenjen sprema u *WeakIV* strukturu, dok se prvi bajt podataka XOR miješa sa pretpostavljenim prvim bajtom podataka 0xaa i time dobiva prvi bajt pseudoslučajnog RC4 niza (element *Z WeakIV* strukture). Vrijednost 0xaa uvijek započinje LLC zaglavlje kada se koristi IP mrežni protokol. Implementacija metode *getPacket* razlikuje se utoliko što cijeli dohvaćeni okvir kopira u strukturu *vector*. STL struktura *vector* pogodna je za vraćanje sadržaja okvira iz metode jer može primiti okvire, tj. nizove bajtova, različite dužine. Destruktor razreda je jednostavan jer mu je jedina funkcija završetak pristupa biblioteci.

4.3.2. Programsko generiranje okvira

Razred *RandomGenerator* za generiranje okvira koristi sustavski generator slučajnih brojeva. Ovakva implementacija *Generator* sučelja korisna je u situacijama kada nije moguće koristiti prije opisani *CaptureGenerator* (mrežna kartica i datoteka s okvirima nije dostupna) ili za testiranje. Kako bi *RandomGenerator* mogao generirati okvire potrebno je u konstruktoru definirati ključ kriptiranja (slika 4.10.) Ključ je moguće navesti eksplicitno (konstruktor koji prima vektor bajtova) ili definirati samo dužinu ključa (konstruktor koji prima cijeli broj). Ukoliko je definirana samo dužina tada se ključ tražene dužine generira korištenjem generatora slučajnih brojeva. Ključ kriptiranja sprema se u člansku varijablu *key*.

Implementacija metode *getWeakIV* koristi generator slučajnih brojeva za generiranje inicijalizacijskog vektora (element *IV* strukture *WeakIV*). Inicijalizacijski vektor i ključ kriptiranja zatim se koriste za generiranje prvog bajta pseudoslučajnog RC4 niza (element *Z* strukture *WeakIV*). Metoda

getPacket radi na sličan način. Prilikom generiranja kriptiranog okvira odabire se jedan od nekoliko predefiniranih nekriptiranih okvira. Inicijalizacijski vektor ponovno se generira korištenjem sustavskog generatora slučajnih brojeva i upisuje na odgovarajuće mjesto u okviru. Inicijalizacijski vektor i ključ kriptiranja zatim se koriste za kriptiranje tijela okvira. Tako dobiveni kriptirani okvir vraća se preko parametra metode *packet*.

```
class RandomGenerator: public Generator {
   private:
      vector<byte> key;

   public:
      RandomGenerator(int len);
      RandomGenerator(const vector<byte>& key);

      bool getWeakIV(WeakIV&);
      bool getPacket(vector<byte>& packet);
};
```

Slika 4.10. Deklaracija razreda RandomGenerator

4.4. Glavni program

Glavni program izvodi se u dvije faze. U prvoj fazi provjeravaju se parametri koje korisnik preko komandne linije prosljeđuje programu. Druga faza započinje inicijalizacijom modula *WEPCrack* i *Generator* zadanim parametrima. Glavni program zatim ulazi u beskonačnu petlju u kojoj obavlja dva zadatka: poziva metodu *getWeakIV* modula *Generator* i dobivene inicijalizacijske vektore prosljeđuje modulu *WEPCrack* preko metode *addIV*. Osim toga, glavni program provjerava i listu pronađenih ključeva (dobivenu pozivom metode *getNextKey* razreda *Monitor*). Kada se u listi pojave novi ključevi glavni program ispituje da li je neki od njih upravo ključ kriptiranja okvira. Ključ kriptiranja provjerava se tako da se kriptirani okvir pokuša dekriptirati dotičnim ključem. Kriptirani okvir dobiva se pozivom metode *getPacket* modula *Generator*. Nakon dekriptiranja tijela okvira računa se ICV, tj. CRC32 suma dekriptiranog sadržaja okvira. Ukoliko je izračunata suma jednaka onoj dobivenoj dekriptiranjem okvira, ključ kriptiranja je pronađen. Glavni program ispisuje vrijednost ključa i završava.

4.5. Prevođenje i instalacija

Za prevođenje *WepCrack+* programa koristi se program *make*. Program *make* prevodi *WepCrack+* prema uputama iz *Makefile* datoteke sadržane u direktoriju sa izvornim kôdom. Ukoliko se ne navedu nikakvi parametri, *make* prevodi pojedine module, te ih spaja u izvršnu datoteku:

```
$ make
g++  -I pcap/include  -c -o Generator.o Generator.cc
g++  -I pcap/include  -c -o WEPCrack.o WEPCrack.cc
g++  -I pcap/include  -c -o util.o util.cc
g++  -I pcap/include  -c -o main.o main.cc
g++ Generator.o WEPCrack.o util.o main.o -o wepcrack+ -lpthread
pcap/lib/libpcap.a
```

Linux distribucije obično ne instaliraju *libpcap* biblioteku i pripadna zaglavlja ili koriste starije verzije biblioteke[4]. Da bi se izbjegli problemi koji zbog toga mogu nastati kod prevođenja, *libpcap* biblioteka distribuira se zajedno sa izvornim kôdom programa. Osim *libpcap* biblioteke, *WepCrack+* koristi standardne C (*libc*) i C++ (*libstdc++*) biblioteke, standardnu C biblioteku matematičkih funkcija (*libm*), te biblioteku koja implementira POSIX dretve (*libpthread*).

Instalacija programa svodi se na kopiranje izvršne datoteke u željeni direktoriji:

```
# make install
install -m 0755 wepcrack+ /usr/local/bin
```

WepCrack+ je moguće instalirati i u direktorij različit od pretpostavljenog */usr/local/bin* korištenjem parametra *PREFIX*:

```
$ make install PREFIX=$HOME/bin
install -m 0755 wepcrack+ /home/jdoe/bin
```

4.6. Upute za korištenje

WepCrack+ je tipični Linux komandno-linijski program. Korisnik komunicira s programom preko parametara komandne linije (*argc* i *argv*), dok program rezultate izvođenja prikazuje na standardnom izlazu. Glavni program prevodi se u izvršnu datoteku pod nazivom *wepcrack+*. Pokretanjem programa bez ikakvih parametara dobiva se sljedeći ispis:

```
$ wepcrack+
[main] ERROR: packet source not set

usage: wepcrack+ -i iface
       wepcrack+ -r file
       wepcrack+ -g len
       wepcrack+ -G key

options:
 -d level   set debug level (default: 2)
 -d ml      set debug level for module m to level l
 -f n       fork next key word when more then n guesses
 -g len     use random generator source with random key of length len
 -G key     use random generator source with specified key
 -h         this cruft
 -i iface   get frames from interface
 -k keyid   crack key with keyid (default: any)
 -l num     set key length (default: 13)
 -L n       use -t value for first n key words (default: 2)
 -m mac     crack key for mac address (default: any)
 -M max     try to keep loadaverage at max level (default: 30)
 -r file    get frames from file
 -t n       use top n guesses for lower key words (see -L)(default: 3)
 -T n       use top n guesses for higher key words (default: 1)

debug module names (option -d):
 c  crack
 g  generator
 m  monitor
 M  main
```

4 Samo novije verzije *libpcap* biblioteke ispravno podržavaju okvire definirane 802.11 standardom

Glavni program prijavljuje grešku jer nije naveden izvor okvira i ispisuje upute za korištenje. Iz uputa se vidi da je program moguće koristiti na četiri osnovna načina. Prvi način koristi čitanje okvira direktno sa mrežne kartice:

```
$ wepcrack+ -i wlan0
[main] ERROR: error starting capture: Error opening capture source:
socket: Operation not permitted
```

Prikazana greška naznačuje da program treba pokrenuti sa administratorskim ovlastima:

```
# wepcrack+ -i wlan0
```

Drugi način korištenja odnosi se na čitanje okvira iz datoteke i ne zahtjeva administratorske ovlasi:

```
$ wepcrack+ -r /tmp/frame.dump
```

Ukoliko se ne navedu dodatni parametri, za otkrivanje ključa će se koristiti svi pročitani kriptirani podatkovni okviri. Korištenje okvira je moguće ograničiti navođenjem MAC adrese stanice ili identifikatora ključa. Ograničavanje korištenja okvira korisno je u nekim specifičnim situacijama. Kada se u području signala nalazi više od jedne pristupne točke koje koriste različite ključeve kriptiranja, navodi se MAC adresa pristupne točke čiji ključ kriptiranja treba otkriti:

```
# wepcrack+ -i wlan0 -m 00:04:94:3a:49:30
```

Kada pristupna točka koristi EAP[5], tj. različite ključeve za različite klijente, potrebno je navesti MAC adresu klijenta. Tada se koriste samo okviri u čijim se poljima izvora ili odredišta nalazi upravo navedena MAC adresa. Ako se u mreži istovremeno koristi više od jednog ključa, treba navesti identifikator ključa kojeg treba otkriti:

```
# wepcrack+ -i wlan0 -k 0
```

Ponekad je potrebno postaviti duljinu traženog ključa (pretpostavljena vrijednost je 13 bajtova):

```
# wepcrack+ -i wlan0 -l 5
```

Programsko generiranje okvira zahtjeva navođenje ključa kriptiranja ili bar duljine ključa (u tom slučaju ključ će biti generiran korištenjem generatora slučajnih brojeva):

```
$ wepcrack+ -G 33:ad:33:fe:3a:84:90:32:34:44:54:2f:3d
$ wepcrack+ -g 13
```

Nakon pokretanja programa različiti moduli ispisuju rezultate svog izvođenja na standardni izlaz. Količinu informacija koje se prikazuju moguće je promjeniti za sve:

```
$ wepcrack+ -g 13 -d 4
```

ili samo za pojedine module (npr. modul *Generator*):

```
$ wepcrack+ -g 13 -d g4
```

Kada se navede samo numerička vrijednost, tada se postavlja razina ispisa za sve module. Ako se prije numeričke vrijednosti navede određeno slovo, tada se razina ispisa postavlja samo za traženi modul. Tablica 4.2. opisuje pojedine razine ispisa. Za svaku razinu ispisa navedene su samo

5 Takva mreža može se prepoznati po korištenju EAPoL okvira

informacije koje se dodatno ispisuju, tj. više razine ispisa uključuju niže. U svakodnevnom radu preporuča se korištenje razina 2. Ukoliko se želi detaljniji ispis o napredovanju može se bez bitnijeg usporenja koristiti razina 3. Zbog značajnog usporenja više razine ispisa uputno je koristiti samo u posebnim situacijama kao što su testiranje programa, analiza izvršavanja ili pronalaženje grešaka.

Tablica 4.2. Razine ispisa

Razina	***Značenje***
0	Ispisuju se samo greške i vrijednost ključa (ako je pronađena)
1	Ispisuju se upozorenja
2	Ispisuju se informacije o napredovanju algoritma
3	Ispisuju se detaljnije informacije o napredovanju algoritma
4	Ispisi koji za svaki okvir prikazuju najviše nekoliko redaka informacije
5	Ispisi koji za svaki okvir prikazuju veći broj redaka informacija

Tablica 4.3. prikazuje slova pridružena pojedinim modulima. Za postavljanje razine ispisa određenog modula treba navesti slovo pridruženo tom modulu i željenu razinu ispisa. Za postavljanje razine ispisa 3 modula *Generator* koristi se parametar komande linije *-d g3*.

Tablica 4.3. Oznake modula

Oznaka	***Modul***
c	*CrackThread* dretve
g	Modul *Generator*
M	Glavni program
m	Nadzorna dretva

Pokretanje programa sa razinom ispisa 3 daje mnogo zanimljivih informacija. Na početku svakog retka u uglatim zagradama navedeno je ime modula koji je generirao taj redak. Za *CrackThread* dretve osim naziva navodi se i identifikator dretve. Glavni program najprije ispisuje vrijednosti različitih parametara izvođenja:

```
$ wepcrack+ -r/tmp/x -l5 -d3
[main] IV length is 3
[main] key length is 5
[main] 8 bits per word
[main] fork for top 3 guesses when key word <= 2
[main] fork for top 1 guesses when key word > 2
[main] keep load average below 30
[main] using file /tmp/x
```

Modul *Generator* ispisuje korišteni filter. U prikazanom slučaju filter propušta samo kriptirane podatkovne okvire. Slijedi pokretanje prve *CrackThread* dretve za pronalaženje prvog bajta ključa.

```
[cap_gen] using filter: wlan[0:2] & 0xff40 = 0x0840
[crack/0x0808a708] started __________
[monitor] started 0x0808a708 (__________)
```

Nadzorna dretva svakih 5 sekundi ispisuje stanje pokrenutih *CrackThread* dretvi. Za svaku pojedinu dretvu ispisuje se vrijednost poznatih bajtova ključa. Za nepoznate bajtove ispisuje se __. Nakon toga slijedi ispis deset najvjerojatnijih vrijednosti za bajt kojeg ta dretva određuje. Za svaku od tih vrijednost prikazana je učestalost pojavljivanja. Posljednja četiri elementa ispisuju broj korisnih, broj pregledanih i udio korisnih okvira. Prvi omjer predstavlja statistiku kada se koriste svi okviri, dok drugi omjer opisuje rezultate koji bi se dobili kada bi se koristila *AirSnort* selekcijska funkcija.

```
[monitor] __________     0/a0    0/a1    0/a2    0/a3    0/a4    0/a5
0/a6    0/a7    0/a8    0/a9
0/0        (0.0000%)        0/0        (0.0000%)
```

Pojedine *CrackThread* dretve ispisuju pronađene moguće vrijednosti bajtova ključa. Ispisuje se redni broj okvira, korišteni inicijalizacijskih vektor i moguća vrijednost bajta ključa (navedena u uglatim zagradama). Slovo *A* u uglatim zagradama naznačuje da bi i *AirSnort* dotični okvir koristio za otkrivanje ključa.

```
[crack/0x0808a708] #44548 IV=cb35fe Z=f0 K=<ec>________ [A]
[crack/0x0808a708] #45062 IV=cb36ff Z=79 K=<74>________ [A]
[crack/0x0808a708] #91749 IV=3bc5fe Z=a1 K=<9d>________ [A]
[crack/0x0808a708] #92263 IV=3bc6ff Z=fe K=<f9>________ [A]
[crack/0x0808a708] #175108 IV=cc34fe Z=99 K=<95>________ [A]
[crack/0x0808a708] #175622 IV=cc35ff Z=79 K=<74>________ [A]
[crack/0x0808a708] #222309 IV=3cc4fe Z=e6 K=<e2>________ [A]
[crack/0x0808a708] #222823 IV=3cc5ff Z=6a K=<65>________ [A]
[crack/0x0808a708] #305668 IV=cd33fe Z=f8 K=<f4>________ [A]
[crack/0x0808a708] #306182 IV=cd34ff Z=f7 K=<f2>________ [A]
[crack/0x0808a708] sleeping __________
[crack/0x0808a708] #352869 IV=3dc3fe Z=61 K=<5d>________ [A]
[crack/0x0808a708] #353383 IV=3dc4ff Z=9b K=<96>________ [A]
[monitor] __________     2/74    1/5d    1/ec    1/9d    1/65    1/f9
1/95    1/96    1/e2    1/f4
12/385178   (0.0031%)       12/385178   (0.0031%)
[crack/0x0808ae08] started 74________
[monitor] started 0x0808ae08 (74________)
[crack/0x0808ae08] #44310 IV=cb3587 Z=63 K=74<5e>______
```

Nakon nekog vremena ključ može biti pronađen. Tada se pojavljuje sljedeći ispis i program završava s izvođenjem:

```
[main] FOUND KEY: ce 41 35 3a 4a
```

Tablica 4.4. prikazuje značenje svih parametara komandne linije.

Tablica 4.4. Parametri komandne linije

Parametar	***Značenje***
-d	Povećava ili smanjuje razinu ispisa za sve ili pojedine module
-f	Postavlja vrijednost *forkAt* parametra razreda *Monitor*
-g	Postavlja dužinu ključa za programski generator okvira. Ključ tražene dužine se generira korištenjem generatora slučajnih brojeva
-G	Postavlja ključ kriptiranja programskog generatora okvira
-h	Ispis podržanih opcija sa kratkim opisom
-i	Definira ime mrežnog sučelja za dohvat okvira
-k	Definira identifikator ključa koji se želi otkriti
-l	Definira dužinu traženog ključa kriptiranja
-L	Postavlja vrijednosti parametra *depth* razreda *Monitor*
-m	Postavlja MAC adresu stanice čiji ključ se želi odrediti
-M	Postavlja parametar *loadlimit* razreda *Monitor*
-r	Definira ime datoteke koja sadrži okvire
-t	Definira vrijednost parametra *topN* razreda *Monitor*
-T	Definira vrijednost parametra *topNhi* razreda *Monitor*

5. Rezultati eksperimentiranja

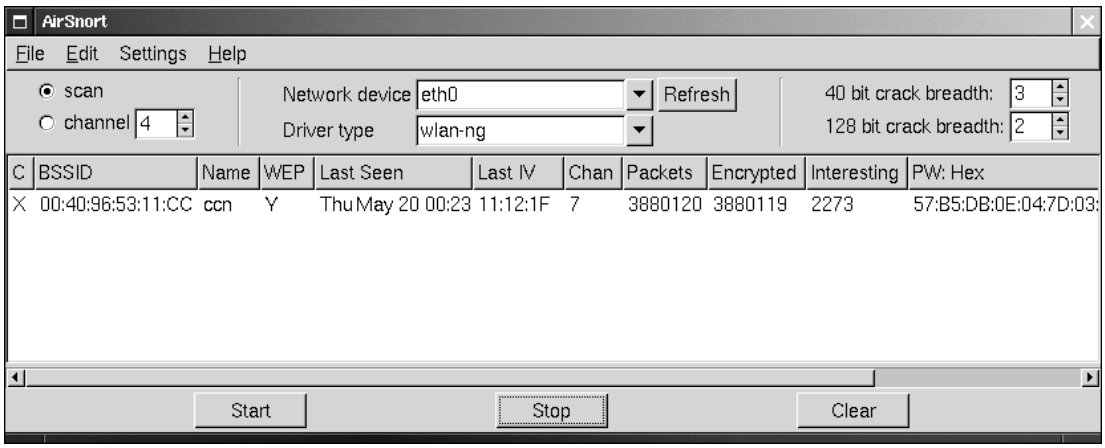
5.1. Uvod

Cilj mjerenja opisanih u ovom poglavlju je određivanje performansi programa. Performanse programa koji određuje WEP ključ moguće je definirati na različite načine. Jedna od najvažnijih mjera je broj okvira potrebnih za određivanje ključa. Kao mjere mogu se koristiti i potrebno vrijeme, procesorska snaga ili memorija. Svaka od navedenih mjera dolazi do izražaja u određenim situacijama. Tako se program koji određuje WEP ključ iz malog broja okvira preferira ako je promet na mreži koja se promatra slab. Tada je prihvatljivo i duže vrijeme izvođenja ako će se time uspjeti otkriti ključ. No dugo vrijeme izvođenja negativno je kada se program izvodi na prijenosnom računalu s ograničenim trajanjem baterije. Procesor i memorija predstavljaju ograničavajuće faktore kada se program izvodi na slabijim računalima (npr. PDA).

Potreban broj okvira predstavlja najzanimljiviju mjeru performansi za programe ovog tipa, pa je tome mjerenju posvećeno najviše pažnje. Radi usporedbe, isto mjerenje provedeno je i za program *AirSnort*. Za preostale, manje bitne mjere, dan je samo red veličine vrijednosti.

5.2. Opis postojeće implementacije - AirSnort

AirSnort vjerojatno je najpopularniji program za otkrivanje WEP ključeva metodom poznatog inicijalizacijskog vektora. Neke od dobrih osobina AirSnort-a uključuju: jednostavno korištenje, pregledno grafičko sučelje, dostupnost na Windows i Linux operacijskim sustavima, podržano sklopovlje itd. Slika 5.1. prikazuje program prilikom otkrivanja jednog od WEP ključeva tijekom izvođenja eksperimenta.



Slika 5.1. AirSnort: uspješno otkriveni ključ

Struktura AirSnort programa donekle je slična strukturi *WepCrack+* programa. AirSnort koristi višedretveno izvođenje. Jedna dretva brine se za osvježavanja grafičkog sučelja i komunikaciju s korisnikom. Preostale dretve se pokreću po potrebi i mogu se podijeliti u dvije skupine: dretve za dohvat okvira i dretve mreže (za otkrivanje ključa određene mreže). Dretva za dohvat okvira dohvaća okvire izravno s mrežne kartice ili iz datoteke.

Dohvaćeni okvir prosljeđuje se dretvi koja obrađuje mrežu (tj. pristupnu točku) kojoj okvir pripada. Ukoliko takva dretva ne postoji, stvara se nova.

Dretva mreže iz pristiglih okvira čita inicijalizacijske vektore i prvu riječ pseudoslučajnog RC4 niza. Pročitani podaci spremaju se u zasebne liste ovisno o kojoj riječi ključa daju informaciju. Koriste se samo poznati slabi inicijalizacijski vektori (definirani u [1] i drugdje). Nakon što su obrađeni svi pristigli okviri i ako su u njima pronađeni novi zanimljivi (potencijalno korisni) inicijalizacijski vektori, dretva mreže pokušava otkriti WEP ključ. Prvo se ispituju svi elementi liste inicijalizacijskih vektora koji otkrivaju prvu riječ ključa. Nakon toga se za najvjerojatnije vrijednosti prve riječi ključa rekurzivno pokreće otkrivanje druge riječi ključa itd. Bitno je primijetiti da se otkrivanje svih riječi ključa mreže izvodi unutar jedne dretve. Nakon pokušaja otkrivanja ključa, dretva završava s izvođenjem. Nova će biti pokrenuta kada se pojave novi okviri iz pripadne mreže.

5.3. Izvođenje mjerenja

Mjerenje je provedeno za deset različitih ključeva dužine pet riječi i za deset različitih ključeva dužine trinaest riječi. Vrijednosti ključeva generirane su korištenjem sustavskog generatora slučajnih brojeva. Cilj mjerenja bio je izmjeriti broj okvira potreban da bi se odredio ključ kriptiranja. Iako je u početku bilo planirano generiranje okvira korištenjem više mrežnih kartica, takvo mjerenje nije bilo moguće provesti zbog nedostataka *AirSnort* programa. Ukoliko bi se okviri generirali na taj način *AirSnort* nebi mogao otkriti niti jedan ključ. Detalji tog problema biti će opisani kasnije. Umjesto toga okviri su se generirali korištenjem *gencases* programa. Za svaki ključ kriptiranja generirano je 10 milijuna okvira. Izvorni kôd *gencases* programa (dio *AirSnort* distribucije) djelomično je izmijenjen za potrebe mjerenja.

Program *gencases* kao parametre od korisnika prima ključ kriptiranja, broj okvira koji treba generirati i ime datoteke u koju se generirani okviri zapisuju. Dobivena datoteka je u *tcpdump* (tj. *libpcap*) formatu tako da ju testirani programi mogu čitati. Program je izmijenjen tako da generirani okviri budu što sličniji prometu koji bi generirale stvarne mrežne kartice. Tako se inicijalizacijski vektori u početku postavljaju na slučajnu vrijednost, a zatim za svaki okvir uvećavaju za jedan. Koriste se dva inicijalizacijska vektora jer se simulira komunikacija između dvije stanice (stanice obično nezavisno odabiru početne inicijalizacijske vektore). Promet koji se geneririira sastoji se od ARP paketa i ICMP Echo paketa. Točnije, svakih dvadeset okvira generira se ARP zahtjev i odgovor. Preostali okviri sadrže naizmjenično pakete ICMP Echo zahtjeva i odgovora. Program sadrži nekriptirane ARP i ICMP okvire. Pomoću brojača određuje se koji će sljedeći okvir biti generiran i koji će se inicijalizacijski vektor koristiti. Okvir se zatim kriptira i sprema u datoteku. Proces se ponavlja dok se ne generira traženi broj okvira. Dobivenu datoteku (veličine oko 1.1GB) programi koriste kao izvor kriptiranih okvira.

Mjeri se broj okvira potreban da se odredi vrijednost ključa. Bitno je napomenuti da dobivene vrijednost nisu precizne već trebaju dati samo red veličine i poslužiti za usporedbu. Razlog tome je što se oba programa izvode višedretveno pa dobivene vrijednosti ovise o učestalosti osvježavanja ispisa i

međudjelovanju dretvi (npr. jedna dretva generira moguće ključeve, a druga ih tek kasnije testira).

5.4. Rezultati mjerenja

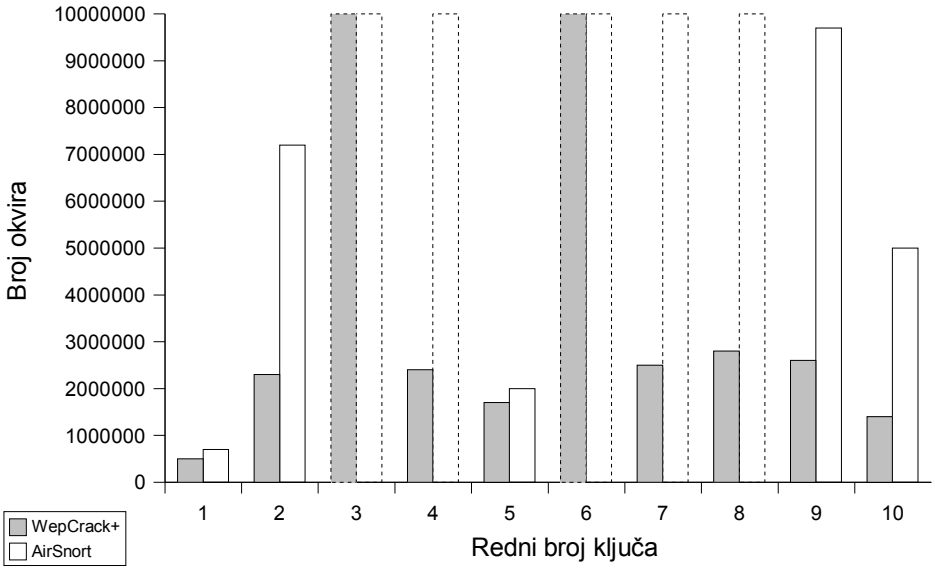
Rezultati dobiveni mjerenjem prikazani su odvojeno za ključeve dužine pet odnosno trinaest riječi. Tablica 5.1. prikazuje dobivene rezultate za ključeve dužine pet riječi. Korišteni ključevi (drugi stupac) u nastavku teksta referencirat će se prema rednim brojevima (prvi stupac). Početni inicijalizacijski vektori navedeni su u tablici radi mogućnosti ponavljanja mjerenja jer je niz generiranih okvira potpuno određen ključem i početnim inicijalizacijskim vektorima. Preostali stupci sadrže broj okvira, odnosno broj zanimljivih okvira koje je određeni program koristio za otkrivanje ključa. Zanimljivi okviri su oni koji daju informaciju o određenim bajtovima ključa. Znak "veće od" u nekim poljima znači da je potrebno više od deset milijuna okvira za otkrivanje ključa, tj. ključ nije otkriven.

Značenje broja zanimljivih okvira donekle se razlikuje za dva programa. Kod *AirSnort*-a ta vrijednost odgovara broju potencijalno korisnih okvira. *WepCrack*+ čuva sve okvire pa je broj potencijalno korisnih okvira jednak ukupnom broju okvira. Zbog toga je kod *WepCrack*+-a broj zanimljivih okvira jednak broju okvira koji su zaista iskorištenih za otkrivanje ključa. Ukoliko ključ nije u potpunosti otkriven, onda je taj broj jednak broju okvira iskorištenih za otkrivanje ispravnih vrijednosti ključa. Ako programi koriste iste okvire za otkrivanje ključa tada će broj zanimljivih okvira *WepCrack*+-a biti manji ili jednak broju zanimljivih okvira *AirSnort*-a jer *WepCrack*+ broji samo one okvire koji su uistinu i dali neku informaciju o ključu. Moguće je da potencijalno zanimljiv inicijalizacijski vektor tijekom izvođenja KSA algoritma uništi neku od vrijednosti u poljima S[1], S[S[1]] ili S[S[1]+S[S[1]]]. Takav će se okvir ubrojiti u zanimljive okvire kod *AirSnort*-a, ali ne i kod *WepCrack*+-a.

Tablica 5.1. Broj okvira potreban za određivanje ključa dugog pet bajtova

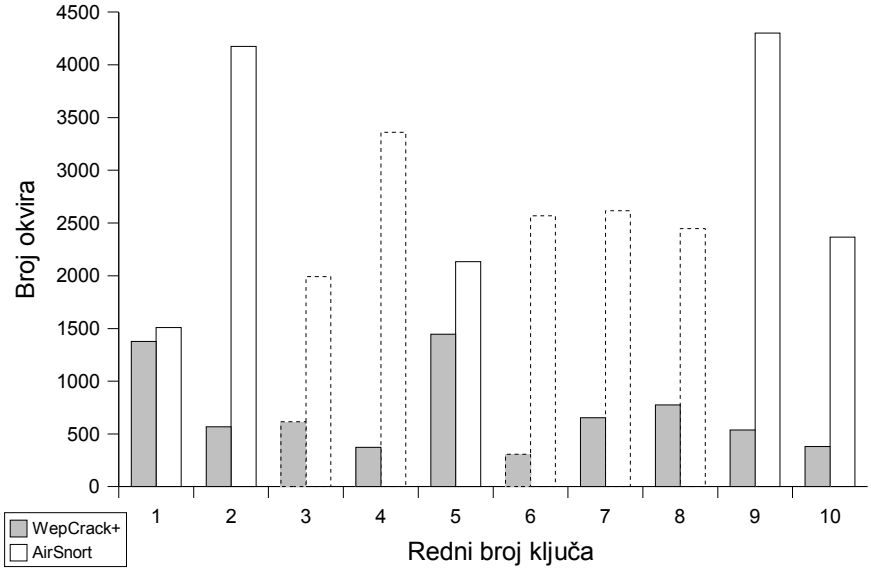
Red. br.	***Vrijednost ključa***	***Početni IV***		***WepCrack+***		***AirSnort***	
				okvira	*zanimljivih*	*okvira*	*zanimljivih*
1.	b3:aa:9f:d3:45	03:51:cc	75:c7:23	500 000	1 378	700 000	1 510
2.	13:dd:0e:1f:59	8d:49:d3	d2:69:16	2 300 000	568	7 200 000	4 174
3.	90:c8:b8:6f:a3	38:be:e5	32:a3:cb	> 10 000 000	> 616	> 10 000 000	> 1 992
4.	f4:84:b2:2d:95	5c:ea:b6	9b:f4:b7	2 400 000	374	> 10 000 000	> 3 360
5.	10:1f:34:6d:ed	fc:2c:84	cb:b4:ce	1 700 000	1 446	2 000 000	2 134
6.	71:3e:e2:b9:4b	87:e4:f3	a6:1f:4a	> 10 000 000	> 308	> 10 000 000	> 2 568
7.	0f:3f:24:f6:29	4c:97:87	2b:ef:ce	2 500 000	654	> 10 000 000	> 2 616
8.	82:2f:e4:da:44	5a:19:0d	73:6b:d0	2 800 000	776	> 10 000 000	> 2 448
9.	71:eb:a5:5f:14	bc:5b:52	2f:31:d5	2 600 000	537	9 700 000	4 300
10.	1c:8d:b8:9e:2f	de:4d:a6	1f:00:f5	1 400 000	380	5 000 000	2 366

Slike 5.2. i 5.3. navedene podatke prikazuju u obliku grafa.



Slika 5.2. Broj pregledanih okvira prije otkrivanje ključa

Ako ključ nije otkriven nakon deset milijuna okvira, u grafovima se pripadne vrijednosti prikazuju crtkanim obrubom. Primjerice, na slici 5.2. crtkanom linijom je prikazano da niti jedan program nije unutar deset milijuna okvira uspio otkriti treći ključ.



Slika 5.3. Broj inicijalizacijskih vektora korištenih za otkrivanje ključa

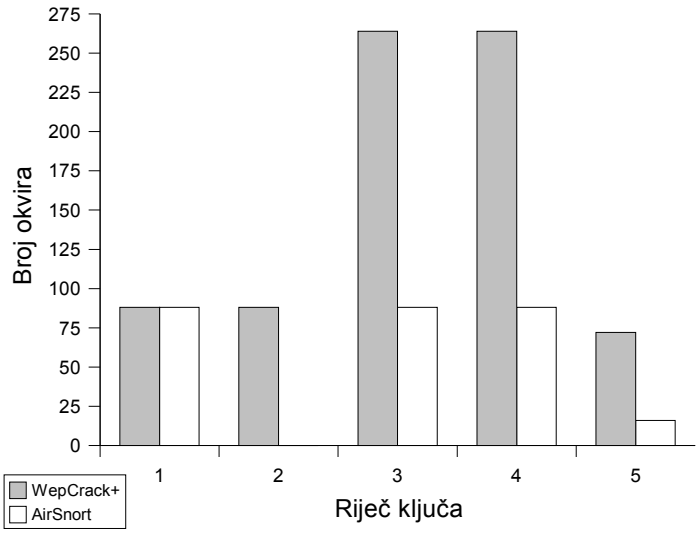
Tablica 5.2. prikazuje raspodjelu korisnih okvira po bajtovima ključa. Za svaki bajt ključa (redak) naveden je broj okvira korištenih za otkrivanje dotičnog bajta u određenom ključu (stupac). Stupci predstavljaju različite ključeve identificirane rednim brojem. Osim ukupnog broja iskorištenih okvira naveden je i broj okvira koji bi bio iskorišten kada bi se koristila selekcija okvira kao u *AirSnort*-u. Ukoliko određeni bajt ključa nije ispravno otkriven,

vrijednost nije navedena. Zbrajanjem vrijednosti korištenih okvira po bajtovima ključa dobiva se broj okvira korištenih za otkrivanje cijelog ključa. Upravo tako se računala vrijednost *WepCrack+* zanimljivih okvira za tablicu 5.1. Vrijednosti u tablici 5.2. dobivene su isključivo korištenjem *WepCrack+* programa jer *AirSnort* ne daje tako detaljne informacije. Broj *AirSnort*-u zanimljivih okvira računao se korištenjem funkcije za selekciju okvira iz izvornog kôda *AirSnort*-a. Ako se određeni okvir može iskoristiti za određivanje neke riječi ključa, funkcija selekcije vraća redni broj te riječi. Ako se okvir ne može iskoristiti funkcija selekcije vraća negativnu vrijednost.

Tablica 5.2. Broj okvira korištenih za otkrivanje pojedinih riječi ključa

Bajt	***Red. br.***	***1.***	***2.***	***3.***	***4.***	***5.***	***6.***	***7.***	***8.***	***9.***	***10.***
1.	AirSnort	270	72	308	72	300	308	78	88	80	40
	WepCrack+	270	72	308	72	303	308	78	88	80	40
2.	AirSnort	252	0	0	0	252	-	0	0	0	0
	WepCrack+	267	72	308	64	301	-	78	88	80	40
3.	AirSnort	264	72	-	40	294	-	78	88	56	40
	WepCrack+	294	216	-	120	397	-	234	264	168	120
4.	AirSnort	260	66	-	36	292	-	78	88	50	40
	WepCrack+	284	198	-	108	395	-	234	264	149	120
5.	AirSnort	3	2	-	2	10	-	6	16	10	12
	WepCrack+	263	10	-	10	50	-	30	72	60	60

Slika 5.4. grafički prikazuje broj okvira korišten za otkrivanje pojedinih riječi tipičnog ključa. Korištene su vrijednosti za ključ sa rednim brojem 8.



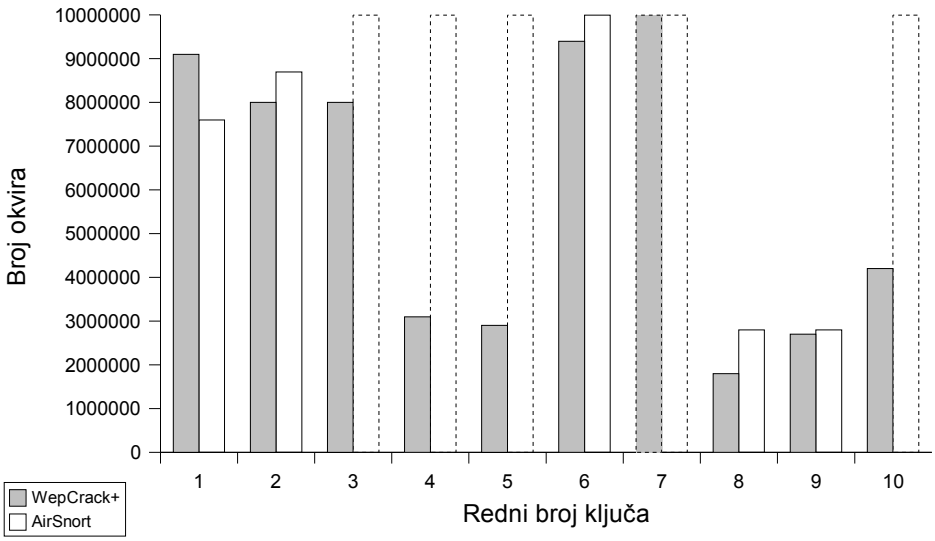
Slika 5.4. Broj okvira korištenih za otkrivanje 8. ključa

Analogno mjerenje provedeno je i za deset ključeva dužine trinaest bajtova. Rezultati tog mjerenja prikazani su u tablicama 5.3. i 5.4. Značenje pojedinih polja jednako je kao i u tablicama 5.1. odnosno 5.2.

Tablica 5.3. Broj okvira potreban za određivanje ključa

Red. br.	***Vrijednost ključa***	***Početni IV***		***WepCrack+***		***AirSnort***	
				okvira	*zanimljivih*	*okvira*	*zanimljivih*
1.	84:e4:b0:db:14:a4:89:5e:86:e5:b4:ff:a3	19:52:4a	cc:1e:ce	9 100 000*	84 784	7 600 000	3 556
2.	18:bc:8c:03:e0:aa:e9:3c:d8:e2:6d:27:de	cc:ea:cf	aa:5d:83	8 100 000*	12 365	8 700 000	5 187
3.	fa:e8:20:cb:0c:22:8e:52:c6:23:5c:42:c0	d3:bc:0b	9b:8a:eb	8 000 000	8 084	> 10 000 000	> 6 437
4.	ec:cb:c6:d8:15:b0:4a:17:ea:0c:e2:5b:26	26:dc:52	63:92:f2	3 100 000	2 097	> 10 000 000	> 3 299
5.	67:21:f6:e4:de:a6:17:47:88:a7:4b:6b:af	35:c9:ce	9f:78:8a	2 900 000	13 121	> 10 000 000	> 3 672
6.	0d:b4:02:ca:51:93:3f:b1:8c:3b:a9:9f:03	65:33:60	c5:8d:d5	9 400 000	9 034	10 000 000*	6 965
7.	d9:13:27:42:6e:59:d3:b9:bd:22:28:2a:4a	b1:0d:9d	65:5b:af	> 10 000 000	> 616	> 10 000 000	> 3 672
8.	ce:41:35:3a:4a:27:fc:30:63:95:09:13:e4	78:0a:87	fb:7e:ac	1 800 000	4 824	2 800 000	4 301
9.	57:b5:db:0e:04:7d:03:02:d2:e7:bf:ca:03	f3:77:c4	a6:5e:c4	2 700 000	4 170	2 800 000	2 273
10.	47:f0:4f:b7:41:f5:a8:81:ef:1c:6f:07:15	b5:b3:38	38:3f:f2	4 200 000	10 252	> 10 000 000	> 3 671

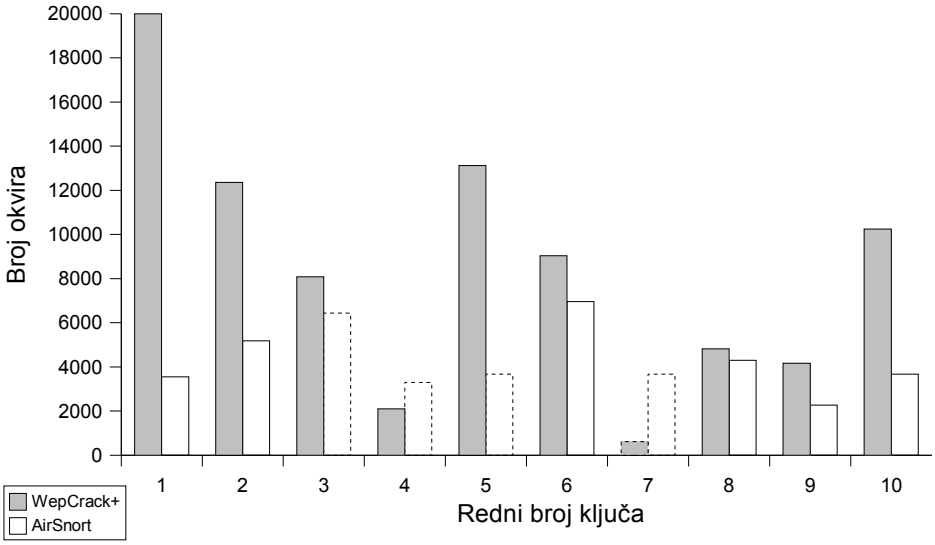
Vrijednosti za ključeve koji nisu otkriveni nakon pregledavanja 10 milijuna okvira prikazane su znakom “veće od”. To znači je potrebno više od naznačenog broja okvira da bi se otkrio ključ. Vrijednosti označene znakom zvjezdica (*), označavaju da dotični ključ nije mogao biti određen korištenjem pretpostavljenih vrijednosti parametara algoritma, već su se morale koristiti izmijenjene vrijednosti. Povećane vrijednosti parametara koristile su se kod *WepCrack*+-a za otkrivanje ključa 1. i 2. Da bi ti ključevi bili određen, bilo je potrebno koristiti pretraživanje dvije najvjerojatnije vrijednosti viših bajtova ključa umjesto pretraživanja samo najvjerojatnije vrijednosti (pretpostavljena vrijednost parametra korištena u ostalim mjerenjima). Slično vrijedi i za ključ 6. kod mjerenja *AirSnort*-om. Da bi *AirSnort* otkrio ključ bilo je potrebno koristiti pretraživanje tri najvjerojatnije vrijednosti za više bajtove ključa, umjesto pretraživanja dvije najvjerojatnije vrijednosti (što je pretpostavljena vrijednost korištena u drugim mjerenjima). Slike 5.5. i 5.6. prikazuju ove



Slika 5.5. Broj okvira pregledanih prije pronalaženja ključa

podatke grafički. Vrijednosti za ključeve koji nisu otkriveni ponovno su prikazane crtkanom linijom (npr. za ključ 7.)

Broj okvira korištenih u *WepCrack*+-u za otkrivanje ključa 1. na slici 5.6. prikazana je iznosom 20000 iako je pravi iznos 84784. To je učinjeno zato da bi se ostale vrijednosti i njihov odnos bolje vidio.



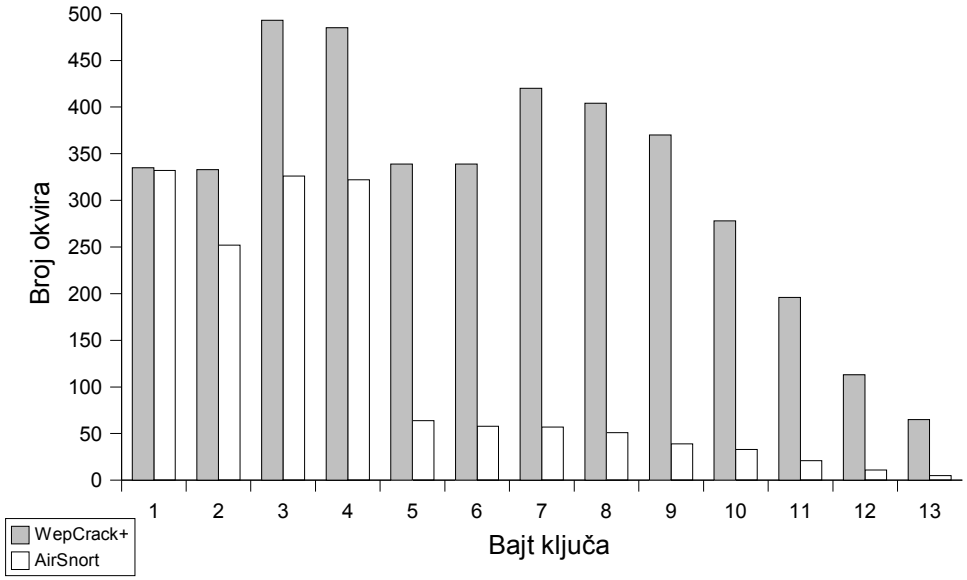
Slika 5.6. Broj okvira korištenih za otkrivanje ključa

Tablica 5.4. sadrži broj okvira koji su korišteni za otkrivanje pojedinih bajtova ključa kada se koriste svi (*WepCrack*+) ili samo poznati slabi inicijalizacijski vektori (*AirSnort*). Kada vrijednost ključa nije određena u potpunosti prikazuju se samo vrijednosti za ispravno određene bajtove ključa (npr. ključ 7.)

Tablica 5.4. Broj okvira korištenih za otkrivanje pojedinih bajtova ključa

Bajt	*Red. br.*	*1.*	*2.*	*3.*	*4.*	*5.*	*6.*	*7.*	*8.*	*9.*	*10.*
1.	AirSnort	530	496	492	94	90	536	308	304	332	132
	WepCrack+	533	499	495	94	90	539	308	307	335	132
2.	AirSnort	252	252	253	0	0	252	0	252	252	0
	WepCrack+	531	497	494	94	90	537	308	305	333	132
3.	AirSnort	524	218	487	83	90	530	-	298	326	132
	WepCrack+	1087	669	974	248	270	1105	-	409	493	396
4.	AirSnort	522	194	368	71	90	528	-	296	322	132
	WepCrack+	1085	582	853	212	270	1103	-	407	485	396
5.	AirSnort	385	160	82	49	90	196	-	294	64	116
	WepCrack+	1368	800	738	243	405	980	-	511	339	464
6.	AirSnort	380	124	122	43	90	148	-	292	58	84
	WepCrack+	34445	620	732	194	450	740	-	510	339	462
7.	AirSnort	512	86	108	27	45	129	-	48	57	56
	WepCrack+	2324	559	702	178	540	842	-	394	420	7112
8.	AirSnort	510	42	94	23	38	96	-	46	51	44
	WepCrack+	2590	577	705	151	9158	624	-	309	404	330
9.	AirSnort	507	26	88	19	62	80	-	43	39	40
	WepCrack+	2877	6805	792	155	558	720	-	370	370	300
10.	AirSnort	506	16	74	16	50	35	-	39	33	28
	WepCrack+	37463	259	629	143	475	595	-	436	278	238
11.	AirSnort	34	13	52	15	15	26	-	31	21	12
	WepCrack+	391	238	416	177	495	494	-	350	196	138
12.	AirSnort	10	10	36	11	22	36	-	24	11	8
	WepCrack+	90	207	414	174	264	433	-	332	113	92
13.	AirSnort	4	2	14	3	4	28	-	16	5	4
	WepCrack+	52	53	140	34	56	322	-	184	65	60

Slika 5.7. grafički prikazuje vrijednosti iz tablice 5.4. za ključ 9.



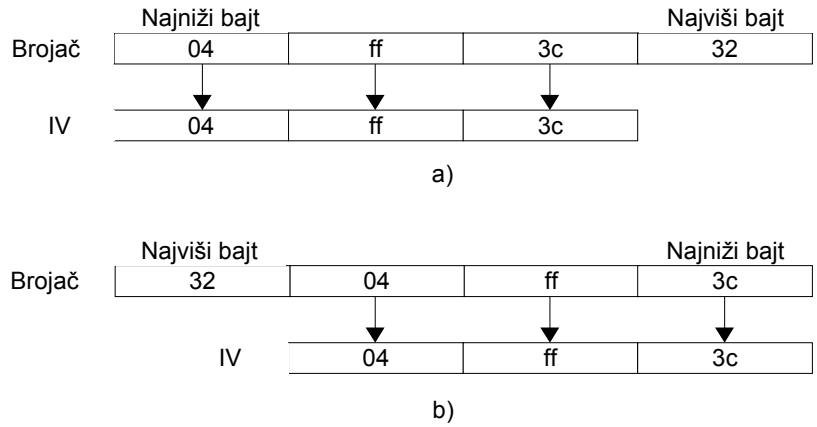
Slika 5.7. Broj okvira korištenih za otkrivanje pojedinih bajtova ključa 9.

6. Analiza rezultata

6.1. Uvod

Za analizu dobivenih rezultata bitno je razumjeti kako pojedini programi odabiru inicijalizacijske vektore koje će koristiti za otkrivanje ključa. Već je spomenuto da *AirSnort* koristi selekcijsku funkciju koja određuje da li se promatrani inicijalizacijski vektor (tj. okvir) može koristiti za otkrivanje određene riječi ključa. Selekcijska funkcija prepoznaje nekoliko klasa slabih inicijalizacijskih vektora. Najpoznatija klasa su svakako vektori oblika (3+*B*, 0xff, *X*)[1]. Pri tome *X* može biti bilo koja vrijednost, dok je *B* redni broj riječi WEP ključa o kojoj vektor daje informaciju. Tako vektori kod kojih je *B*=2 daju informaciju o trećoj riječi ključa. Broj vektora u ovoj klasi je značajan (jer *X* ima 256 različitih vrijednosti) pa se cijeli ključ često može otkriti samo korištenjem te klase. Vektori ove klase su usko grupirani i zauzimaju područje vrijednosti inicijalizacijskog vektora od *0x03ff00* do *0x0fffff*.

Iako za otkrivanje većine riječi ključa *AirSnort* koristi i druge klase slabih vektora, za otkrivanje druge riječi ključa koriste se samo vektori oblika *0x04ffxx*. To se može zaključiti prolaskom kroz cijeli prostor vrijednosti inicijalizacijskog vektora i pozivanjem selekcijske funkcije za svaku od vrijednosti. Pokazuje se da samo vektori iz područja vrijednosti *0x04ff00-0x04ffff* daju informaciju o drugoj riječi WEP ključa. To znači da ako se u ulaznim podacima ne nalaze vektori iz navedenog područja, *AirSnort* neće moći odrediti drugu riječ ključa, a time ni ostatak ključa. Upravo to je razlog zbog kojega mjerenje nije bilo moguće izvršiti na stvarnom sklopovlju. Većina novijih bežičnih mrežnih kartica radi sprječavanja otkrivanja ključa izbjegava generiranje inicijalizacijskih vektora ove klase. Prepoznavanje vektora ove klase je vrlo jednostavno (srednji bajt je uvijek jednak 0xff), a kao što se vidi na primjeru *AirSnort*-a, može imati za posljedicu potpuno onemogućavanje napada. No čak i ako je generiranje vektora ove klase dozvoljeno, moguće je da će biti potrebno sakupiti velik broj okvira dok se počnu generirati vektori ove klase. To ovisi o načinu generiranja inicijalizacijskih vektora.



Slika 6.1. Generiranje inicijalizacijskih vektora

Inicijalizacijski vektori najčešće se generiraju korištenjem brojača. Početna vrijednost brojača dobiva sa izlaza generatora slučajnih brojeva. Dužina brojača obično je veća od tri bajta pa se za inicijalizacijski vektor

koriste samo tri najniža bajta. Na koji će se način bajtovi iz brojača preslikati u inicijalizacijskih vektor ovisi o redoslijedu spremanja bajtova brojača u memoriju. Postoje dvije mogućnosti (slika 6.1.): najniži bajt brojača može se spremati u prvi (a) ili u posljednji (b) bajt inicijalizacijskog vektora. Pošto se vrijednost najnižeg bajta brojača najbrže mijenja, način spremanja određuje i kojim će se redoslijedom generirati inicijalizacijski vektori. Ako se najniži bajt brojača sprema u prvi bajt inicijalizacijskog vektora, jedan vektor oblika *0x04ffxx* generirati će se otprilike svakih 65 000 okvira (broj okvira potreban da srednji bajt inicijalizacijskog vektora postane *0xff*). Ako se najniži bajt brojača sprema u treći bajt inicijalizacijskog vektora tada će se svih 256 vektora oblika *0x04ffxx* generirati zajedno, ali otprilike svakih 16 milijuna okvira. Prilikom generiranja testnih podataka koristila se upravo ova metoda, pa su se pojavljivali slučajevi u kojima *AirSnort* nije mogao odrediti drugu riječ ključa (a time i cijeli ključ) jer opseg generiranih vektora nije uključivao i vrijednosti *0x04ffxx*.

6.2. Ključevi dužine 5 bajtova

Tablica 5.1. pokazuje da se to dogodilo za ključeve sa rednim brojem 3., 4., 6., 7. i 8. Za svaki ključ generirano je 10 milijuna okvira, ali je polovina od toga generirana jednim, a polovina drugim brojačem. Početne vrijednosti brojača također su navedene u tablici. Povećanjem početne vrijednosti brojača za 5 milijuna vidi se da niti jedan od brojača nije dostigao vrijednost *0x04ff00*. Da bi brojač nakon 5 milijuna okvira dostigao vrijednost *0x04ff00* potrebno je da početna vrijednost bude veća od *0xb8b3c0*. Za preostale ključeve *AirSnort* je ispravno odredio vrijednost ključa nakon što je brojač prošao područje vrijednosti *0x04ffxx*. Također se primjećuje da je trenutak prolaska kroz navedeno područje odredio i broj pregledanih okvira. Za prvi ključ područje vrijednosti *0x04ffxx* je bilo vrlo blizu početne vrijednosti prvog brojača i ključ je nađen nakon samo 700 000 okvira. Za deveti ključ brojač je prošao navedeno područje (i ključ je određen) tek nakon što su pregledani gotovo svi generirani okviri.

Isti podaci za *WepCrack+* pokazuju manju ovisnost o klasi vektora (3+B, 0xff, X). To se vidi iz činjenice da je ključ uspješno određen prije nego je vrijednost brojača zašla u područje navedene klase. Iz toga se također može zaključiti da bi *WepCrack+* trebao moći odrediti ključ čak i kada se cijela klasa (3+B, 0xff, X) ne generira. Iz slika 5.2. i 5.3. može se zaključiti da *WepCrack+* obično pronalazi ključ nakon pretraživanja manje okvira i korištenjem manje slabih inicijalizacijskih vektora nego *AirSnort*. Ipak u dva slučaja (ključevi 3. i 6.) niti on nije uspio odrediti ispravan ključ. Tablica 5.2. pokazuje da *WepCrack+* nije uspio odrediti vrijednost drugog odnosno prvog bajta navedenih ključeva, respektivno. Ti su slučajevi dokaz stohastičnosti algoritma zbog koje se ponekad pogrešne vrijednosti bajtova ključa dobivaju kao najvjerojatnije. Kod ključeva 3. i 6. ispravne vrijednosti nisu se nalazile čak niti unutar 10 najvjerojatnijih vrijednosti.

Tablica 5.2. prikazuje koliko je okvira *WepCrack+* koristio za otkrivanje određene riječi ključa, a koliko bi od tih okvira koristio *AirSnort* (prema funkciji selekcije). Primjećuje se da oba programa vrlo često koriste iste vektore za otkrivanje prve riječi ključa. Za drugi bajt kod *AirSnort*-a

karakteristična je vrijednost *0* što je posljedica prije opisane ovisnosti o klasi vektora sa vrijednostima *0x04ffxx*. Čak i kada *AirSnort* pronalazi slabe vektore za drugu riječ ključa (npr. ključ 5.), to je obično manje od broja slabih vektora koje koristi *WepCrack+*. Dakle, već na drugoj riječi ključa vidi se korist od korištenja svih, a ne samo poznatih slabih vektora. Za *AirSnort* se primjećuje da je broj vektora korištenih za različite bajtove ključa obično jednak. To je posljedica činjenice da se slabi inicijalizacijski vektori koje *AirSnort* koristi nalaze obično grupirani. To znači da se nakon pronalaska slabog vektora za određenu riječ ključa, obično samo promjenom jednog bita dobiva inicijalizacijski vektor koji otkriva vrijednost neke druge riječi ključa. *WepCrack+* koristi i druge inicijalizacijske vektore, što se vidi iz vrijednosti za drugu i više riječi ključa. *WepCrack+* na istom broju okvira pronalazi i do tri puta više slabih inicijalizacijskih vektora. Primjetno je da su vrijednosti za posljednji bajt značajno manje. To se posljedica činjenice da se već pregledavanjem malog broja okvira (50 000 – 100 000) može odrediti vrijednost posljednjeg bajta ključa. Pošto se tada izvođenje programa završava broj iskorištenih vektora za posljednji bajt ostaje relativno nizak. Slika 5.4. zorno prikazuje navedene činjenice na vrijednostima za ključ 8.

6.3. Ključevi dužine 13 bajtova

Rezultati za ključeve dužine 13 bajtova (tablica 5.3.) pokazuju tendencije uočene kod ključeva dužine 5 bajtova. *AirSnort* za četiri ključa (4., 5., 7. i 10.) nije uspio odrediti ključ zbog prije opisanih problema s određivanjem drugog bajta ključa. Za ključ 7. niti jedan program nije uspio odrediti ispravnu vrijednost drugog bajta. Čak niti nakon pregledavanja svih 10 milijuna okvira ispravna vrijednost nije bila među deset navjerovatnijih. Za tri ključa (označena zvjezdicama) bilo je potrebno proširit parametre pretraživanja da bi se ključ odredio. To je vidno povećalo i vrijeme izvođenja programa. Iz slike 5.5. vidljivo je da *WepCrack+* i dalje gotovo uvijek koristi manje okvira za određivanje ključa. Osim što koristi manje okvira, u njima pronalazi više onih koje je moguće iskoristiti za otkrivanje ključa. To znači da je omjer broja korisnih po broju pregledanih okvira veći za *WepCrack+*. Iz ovoga se može zaključiti da je *WepCrack+* pogodniji kada je za analizu dostupno manje okvira (npr. kada je promet na mreži slab ili je sakupljen samo ograničeni broj okvira).

Slika 5.7. prikazuje broj okvira korištenih za otkrivanje pojedinih bajtova ključa 9. Broj okvira korištenih za određivanje prvog bajta ponovno je gotovo jednak, iako *WepCrack+* očito pronalazi nekolicinu okvira koje *AirSnort*-ova funkcija selekcije na prepoznaje. Već na drugom bajtu vidi se korist od korištenja svih okvira jer *WepCrack+* pronalazi zamjetno veći broj korisnih okvira. Na sljedećim bajtovima taj omjer još više raste. Prema posljednjim bajtovima ključa broj korištenih okvira pada, ali samo zato što je vrijednost tih bajtova moguće odrediti i nakon pregledavanja manjeg broja okvira. Ukoliko bi se promotrio udio korisnih u broju pregledanih okvira za *WepCrack+* bi se uočio rast prema posljednjim bajtovima ključa. Tako za ključ 6. udio korisnih okvira rast sa 0.0057% na prvom bajtu do čak 0.0367%, što je povećanje od oko šest puta. Za *AirSnort* taj omjer je za sve bajtove približno jednak i kreće se oko 0.0031% sa manjim odstupanjima. Iz svega ovoga može se zaključiti da je WepCrack+-u najviše okvira potrebno za određivanje prvih bajtova

ključa. Kada se njihove vrijednosti točno odrede, obično će biti dovoljno okvira i za određivanje ostatka ključa. Ako je na raspolaganju manji broj okvira (nedovoljan za određivanje prvih bajtova), uz dovoljno vremena, mogu se pretražiti sve vrijednosti za, na primjer, prva dva bajta ključa, dok se preostali određuju uobičajenim pretraživanjem samo najvjerojatnijih vrijednosti. Sada je jasno i zašto *WepCrack+* pretražuje više (obično tri) najvjerojatnijih vrijednosti za prva dva bajta ključa, dok se za više bajtove ključa pretražuje samo jedna najvjerojatnija vrijednost.

6.4. Prostorna i vremenska složenost

Provedena mjerenja pokazuju da *WepCrack+* tijekom izvođenja koristi više memorije i procesorskog vremena od *AirSnort*-a. Zauzeće memorije u izravnoj je vezi sa brojem inicijalizacijskih vektora koji se čuvaju. Pošto *WepCrack+* čuva sve, a *AirSnort* samo odabrane vektore za očekivati je da će i zauzeće memorije *WepCrack+* biti veća. Nakon pokretanja sam *WepCrack+* program zauzima manje od megabajta memorije[1], no čitanjem okvira počinje se stvarati lista inicijalizacijskih vektora i memorija se polako puni. Mjerenja pokazuju da lista koja sadrži oko milijun inicijalizacijskih vektora zauzima oko 30MB memorije. To znači da čitanje svih 10 milijuna testnih okvira pri kraju izvođenja daje zauzeće memorije od oko 300MB. Zauzeće memorije *AirSnort*-a nakon pokretanja iznosi oko 5MB, što je posljedica korištenja grafičkog sučelja. Nakon čitanja 10 milijuna okvira i čuvanja oko 7 000 inicijalizacijskih vektora korištena memorije se povećava tek neznatno – na 7MB.

Slično kao za memoriju može se zaključiti i za trajanje izvođenja. Vrijeme izvođenja ovisi prije svega o računalu na kojemu se program izvodi, a zatim i o korištenom algoritmu i broju okvira koje je potrebno pregledati da bi se otkrio ključ. *AirSnort* traženje ključa obavlja u jednoj dretvi pomoću rekurzivnih poziva. Prvo se određuju najvjerojatnije vrijednosti za prvu riječ ključa, a zatim se rekurzivno poziva određivanje druge riječi za najvjerojatnije vrijednosti prve riječi. Za koliko će se najvjerojatnijih vrijednosti pretraživanje rekurzivno pozivati (engl. *breadth*) parametar je koji najviše utječe na vrijeme izvođenja. Prilikom testiranja koristile su se pretpostavljene vrijednosti koje za prvih pet bajtova (tj. za ključeve dužine pet bajtova) određuju pretraživanje tri najvjerojatnije vrijednosti dok se za ostale bajtove pretražuju samo dvije najvjerovatnije vrijednosti. Ovakve vrijednosti parametara dobro su postavljene i kada ima dovoljno korisnih inicijalizacijskih vektora određuju ključ u vrlo kratkom vremenu (manje od 5 minuta[2]). Ukoliko ključ nije pronađen sa pretpostavljenim vrijednostima parametara, izvođenje je ponovno pokrenuto sa pretraživanjem tri (umjesto dvije) najznačajnije vrijednosti viših riječi ključa. Time se pretraživanje produžava, ali ne značajno (oko 15 minuta).

WepCrack+ koristi posebnu dretvu za otkrivanje svake riječi ključa. Iako je teoretski moguće koristiti isti rekurzivni algoritam kao u *AirSnort*-u, to bi još više produljilo pretraživanje jer bi trebalo sačekati da se pregledaju svi okviri da bi se odlučilo koji su najbolji kandidati za prvi bajt ključa. Obradom svakog

1 U to su uključene i sve potrebne biblioteke
2 Program se izvršavao na osobnom računalu sa 384MB RAM-a i procesorom na 1.3GHz

bajta u posebnoj dretvi moguće je ranije pokrenuti otkrivanje drugog bajta za najvjerojatnije vrijednosti prvog bajta ključa. Pretpostavljene vrijednosti algoritma uključuju pretraživanje tri najvjerojatnije vrijednosti za prvi i drugi bajt. Za preostale bajtove pretražuje se samo najvjerojatnija vrijednost. Iako se na ovaj način pretražuje manje vrijednosti nego kod *AirSnort*-a, pretpostavlja se da će veći broj korištenih vektora gotovo uvijek kao najvjerojatniju vrijednost za više bajtove postaviti ispravnu vrijednost. Problem koji se pojavljuje kod ovakvog načina rada je da velik broj dretvi bitno usporava rad tako da pojedine dretve vrlo sporo napreduju. Za ključeve od 5 bajtova pokreće se oko 25 dretvi dok se za ključeve od 13 bajtova pokreće gotovo 100 dretvi. To direktno utječe na performanse, pa izvođenje za ključeve dužine 5 bajtova traje 20 i više minuta (u ovisnosti o broju pregledanih okvira). Za ključeve dužine 13 bajtova situacija je bitno lošija i kreće se od 30 minuta do sat vremena i više. Pretraživanjem dvije najvjerovatnije vrijednosti za više bajtove ključa izvođenje se redovito produžava na više od sat vremena. Iako je ovakvo produženo vrijeme izvođenja očekivano sobzirom da se pregledava više inicijalizacijskih vektora, to je često nepraktično. Zbog toga je uputno prvo pokušati sa *AirSnort*-om, a tek za zahtjevnije ključeve ili u slučaju malog broja okvira koristiti *WepCrack+*. Veće brzine izvođenja mogle bi se postići distribuiranim izvođenjem na više računala ili korištenjem heurističkih algoritama koji bi davali prednost u izvođenju onim dretvama za koje se smatra da vode ka cilju.

Zauzeće memorije *WepCrack+*-a može donekle biti smanjeno korištenjem neke druge strukture podataka za pohranu inicijalizacijskih vektora. Iako je korištenje liste za pohranu inicijalizacijskih vektora praktično jer sva potrebna memorija ne mora biti alocirana u linearnom slijedu, u ovom slučaju to ima i jedan bitan nedostatak. Uz pretpostavku da su STL liste ostvarene samo pomoću dva pokazivača (na prethodni i sljedeći član liste) dužine četiri bajta, memorija koju zauzima jeden element liste je tri puta veća od podataka (4 bajta) koji se čuvaju u čvoru liste. Slijedno spremanje inicijalizacijskih vektora zahtijevalo bi tri puta manje prostora. Pošto su svi inicijalizacijski vektori jednako dugi (tri bajta IV i jedan bajt prva riječ pseudoslučajnog niza) moguće ih je spremiti i u neformatiranu datoteku. Pohrana 10 milijuna inicijalizacijskih vektora u takvu datoteku zahtijevala zahtijevala bi oko 40MB prostora. Takva datoteka zatim bi se mapirala u memoriju programa i pristupalo bi joj se kao polju. Manji broj cijelih okvira spremio bi se u posebnu datoteku i koristio samo za testiranje pronađenih ključeva. Na taj način bi se zahtijevani prostor na disku i u memoriji bitno smanjio. Umjesto više od 1GB podataka, koliko je korišteno prilikom opisanih mjerenja, ista bi se količina informacija mogla pohraniti na prostor manji od 50MB, kako na disku tako i u memoriju. To bi vjerojatno pozitivno utjecalo i na brzinu izvršavanja programa.

Brzina *WepCrack+*-a osim optimizacijom postojećeg algoritma, mogla bi se povećati korištenjem i drugih poznatih slabosti RC4 algoritma. Tako je tijekom mjerenja primijećeno da se često, posebno kod ključeva dužine trinaest bajtova, dobivaju ključevi koji nisu potpuno ispravni, ali su vrlo slični traženom ključu. *WepCrack+* je često generirao ključeve koji imaju samo par neispravnih bajtova oko sredine ključa. Takve ključeve bilo bi korisno

sakupljati, analizirati i na osnovu njih generirati nove. Primjerice, kada bi bila pronađena dva neispravna ključa koji se razlikuju samo u sedmom bajtu, mogao bi se generirati niz ključeva koji sadrže zajedničke bajtove ključa, dok bi se na mjestu sedmog bajta isprobale sve moguće vrijednosti. Takva analiza izvodiva je u razumnom vremenu čak i ako se ključevi razlikuju za dva ili tri bajta. Algoritam bi se mogao dodatno poboljšati kada se nebi morale ispitivati sve moguće vrijednosti različitih bajtova već, na primjer, samo dvadeset prema *WepCrack*+-u najvjerojatnijih.

Dobiveni rezultati mogu poslužiti i kao osnova za određivanje parametara sigurnosti bežične mreže. Većina postojećih bežičnih mreža zasniva sigurnost na nekom tipu EAP autentifikacije i dinamičkom WEP-u. Dinamični WEP zapravo znači izmjenu WEP ključeva prilikom svake EAP autentifikacije. Pitanje koje se pri tome postavlja je koliki treba biti period između dvije autentifikacije (tj. trajanje generiranog WEP ključa). Uz pretpostavku da se koristi ključ dužine 13 bajtova, iz mjerenja se može zaključiti da je za otkrivanje ključa redovito potrebno više od milijun okvira. Prema tome WEP ključ potrebno je mijenjati otprilike svakih milijun okvira ili češće. Dodatni problem je što najčešće nije moguće definirati broj razmijenjenih okvira već samo vremenski period promjene ključa. Konkretne vrijednosti najbolje je dobiti mjerenjem na promatranoj mreži. Pri tome se utvrđuje kojim intenzitetom klijenti generiraju okvire, a iz tog i koliko će im vremena trebati za generiranje kritičnog broja okvira. Jednostavnim modeliranjem može se odrediti red veličine tog perioda. Ako se pretpostavi prosječna veličina okvira od 1000 bajtova i propusnost mreže od 11Mbps, za generiranje milijun okvira potrebno je bar 15 minuta. Unutar 15 minuta napadač će vrlo teško čak i pri vrlo intenzivnom prometu sakupiti dovoljno okvira da bi mogao otkriti ključ kriptiranja. Ukoliko intenzitet prometa na mreži nije jak, uobičajeni su duži periodi između autentifikacija (npr. 30 minuta). Ako se koriste mreže veće brzine (npr. 56Mbps) dužina perioda se proporcionalno smanjuje – ako je mreža pet puta brža potrebno je pet puta češće provoditi autentifikaciju (npr. svakih 5 minuta). Korištenjem sigurnijih algoritama enkripcije (npr. TKIP ili CCMP) periode između autentifikacija moguće je značajno povećati.

7. Zaključak

U okviru ovog diplomskog rada izrađen je program za otkrivanje ključa WEP enkripcije. Algoritam otkrivanja ključa poboljšan je tako da koristi sve, a ne samo određene okvire podataka, kako je tu uobičajeno u postojećim implementacijama.

Usporedna mjerenja pokazala su da izrađeni program često otkriva ključ kriptiranja korištenjem manjeg broja okvira nego postojeća implementacija. Dobiveni rezultati iskorišteni su i za određivanje vremenskog perioda u kojem je sigurno koristiti određeni ključ kriptiranja.

Mjerenja su također pokazala veću prostornu i vremensku složenost od postojeće implementacije, pa je tu najveća prilika za poboljšanje. Prostorna složenost može se smanjiti kompaktnijim spremanjem podataka. Smanjivanje vremenske složenosti teži je problem i uključivao bi korištenje heuristike i iskorištavanje drugih slabosti RC4 algoritma.

Literatura

[1] S. Fluhrer, I. Mantin, A. Shamir, Weaknesses in the Key Scheduling Algorithm of RC4, 2001.

[2] B. Schneier, Applied Cryptography, Second Edition: Protocols, Algorithms and Source Code, John Wiley & Sons, Inc., 1996.

[3] ANSI/IEEE, 802.11: Wireless LAN Medium Access Control, 1999.

[4] IEEE, 802.1X: Port-Based Network Access Control, 2001.

[5] L. Blunk, J. Vollbrecht, RFC 2284: PPP Extensible Authentication Protocol (EAP), 1998.

[6] M. Gast, 802.11 Wireless Networks: The Definitive Guide, O'Reilly, 2002.

[7] B. Adoba, D. Stanley, J. Walker, EAP Method Requirements for Wireless LANs, 2004.

[8] B. Adoba, D. Simon, RFC 2716: PPP EAP TLS Authentication Protocol, 1999.

[9] S. Thomas, SSL and TLS Essentials: Securing the Web, Wiley, 2000.

[10] M. Bellare, H. Krawczyk, R. Canetti, RFC 2104: HMAC: Keyed-Hashing for Message Authentication, 1997.

[11] C. Allen, T. Dierks, RFC 2246: The TLS Protocol Version 1.0, 1999.

[12] A. Palekar, D. Simon, G. Zorn, J. Salowey, H. Zhou, S. Josefsson, Protected EAP Protocol (PEAP) Version 2, 2003.

[13] P. Funk, S. Blake-Wilson, EAP Tunneled TLS Authentication Protocol (EAP-TTLS), 2004.

[14] V. Kamath, A. Palekar, M. Wodrich, Microsoft's PEAP version 0 (Implementation in Windows XP SP1), 2002.

[15] H. Andersson, S. Josefsson, G. Zorn, D. Simon, A. Palekar, Protected EAP Protocol (PEAP), 2002.

[16] IEEE, 802.11 Amendment 6: Medium Access Control (MAC) Security Enhancements, 2004.

[17] D. Whiting, R. Housley, N. Ferguson, RFC 3610: Counter with CBC-MAC (CCM), 2003.

[18] J. Šribar, B. Motik, Demistificirani C++, Element, 2001.

Dodatak A: Primjeri okvira

Kriptirani okvir

U nastavku je prikazan primjer kriptiranog okvir korištenog u testiranju. Okvir je kriptiran ključem sa rednim brojem 7. koji ima vrijednost: d9:13:27:42:6e:59:d3:b9:bd:22:28:2a:4a. Okvir sadrži ARP zahtjev.

```
IEEE 802.11
    Type/Subtype: Data (32)
    Frame Control: 0x4108 (Normal)
        Version: 0
        Type: Data frame (2)
        Subtype: 0
        Flags: 0x41
            DS status: Frame is entering DS (To DS: 1  From DS: 0) (0x01)
            .... .0.. = More Fragments: This is the last fragment
            .... 0... = Retry: Frame is not being retransmitted
            ...0 .... = PWR MGT: STA will stay up
            ..0. .... = More Data: No data buffered
            .1.. .... = WEP flag: WEP is enabled
            0... .... = Order flag: Not strictly ordered
    Duration: 117
    BSS Id: 00:40:96:53:11:cc (00:40:96:53:11:cc)
    Source address: 00:0d:28:4d:cb:f1 (00:0d:28:4d:cb:f1)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Fragment number: 0
    Sequence number: 1685
    WEP parameters
        Initialization Vector: 0x9d0db1
        Key: 0
        WEP ICV: 0x4eed1797 (correct)
Logical-Link Control
    DSAP: SNAP (0xaa)
    IG Bit: Individual
    SSAP: SNAP (0xaa)
    CR Bit: Command
    Control field: U, func = UI (0x03)
        000. 00.. = Unnumbered Information
        .... ..11 = Unnumbered frame
    Organization Code: Encapsulated Ethernet (0x000000)
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: 00:0d:28:4d:cb:f1 (00:0d:28:4d:cb:f1)
    Sender IP address: 169.254.127.178 (169.254.127.178)
    Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
    Target IP address: 169.254.127.1 (169.254.127.1)

Frame:

0000  08 41 75 00 00 40 96 53 11 cc 00 0d 28 4d cb f1   .Au..@.S....(M..
0010  ff ff ff ff ff ff 50 69 b1 0d 9d 00 2b e0 fb f6   ......Pi....+...
0020  63 f8 b3 e7 c9 e9 f8 2a ab 63 50 bd 61 cd 93 c0   c......*.cP.a...
0030  c8 77 11 4c 7e 03 95 88 12 e2 32 8d eb b9 55 1c   .w.L~.....2...U.
0040  c7 73 a5 b8 02 63 5b e1 13 ec c8 9c 54 a2 ef 3b   .s...c[.....T..;
0050  50 f2 4e ed 17 97                                 P.N...

Decrypted WEP data:
```

```
0000  aa aa 03 00 00 00 08 06 00 01 08 00 06 04 00 01   ................
0010  00 0d 28 4d cb f1 a9 fe 7f b2 00 00 00 00 00 00   ..(M............
0020  a9 fe 7f 01 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00                                 ......
```

Autentifikacija: otvoreni sustav

```
IEEE 802.11
    Type/Subtype: Authentication (11)
    Frame Control: 0x00B0 (Normal)
        Version: 0
        Type: Management frame (0)
        Subtype: 11
        Flags: 0x0
            DS status: Not leaving DS or network is operating in AD-HOC
mode (To DS: 0  From DS: 0) (0x00)
            .... .0.. = More Fragments: This is the last fragment
            .... 0... = Retry: Frame is not being retransmitted
            ...0 .... = PWR MGT: STA will stay up
            ..0. .... = More Data: No data buffered
            .0.. .... = WEP flag: WEP is disabled
            0... .... = Order flag: Not strictly ordered
    Duration: 314
    Destination address: 00:40:96:53:11:cc (00:40:96:53:11:cc)
    Source address: 00:0d:28:4d:cb:f1 (00:0d:28:4d:cb:f1)
    BSS Id: 00:40:96:53:11:cc (00:40:96:53:11:cc)
    Fragment number: 0
    Sequence number: 1681
IEEE 802.11 wireless LAN management frame
    Fixed parameters (6 bytes)
        Authentication Algorithm: Open System (0)
        Authentication SEQ: 0x0001
        Status code: Successful (0x0000)

0000  b0 00 3a 01 00 40 96 53 11 cc 00 0d 28 4d cb f1   ..:..@.S....(M..
0010  00 40 96 53 11 cc 10 69 00 00 01 00 00 00         .@.S...i......
```

Za drugi okvir u protokolu autentifikacije prikazano je samo tijelo okvira pošto je zaglavlje gotovo identičan kao u prvom okviru.

```
IEEE 802.11 wireless LAN management frame
    Fixed parameters (6 bytes)
        Authentication Algorithm: Open System (0)
        Authentication SEQ: 0x0002
        Status code: Successful (0x0000)

0000  b0 00 d5 00 00 0d 28 4d cb f1 00 40 96 53 11 cc   ......(M...@.S..
0010  00 40 96 53 11 cc f0 6a 00 00 02 00 00 00         .@.S...j......
```

Autentifikacija dijeljenom tajnom

```
IEEE 802.11
    Type/Subtype: Authentication (11)
    Frame Control: 0x00B0 (Normal)
        Version: 0
        Type: Management frame (0)
        Subtype: 11
        Flags: 0x0
            DS status: Not leaving DS or network is operating in AD-HOC
mode (To DS: 0  From DS: 0) (0x00)
            .... .0.. = More Fragments: This is the last fragment
            .... 0... = Retry: Frame is not being retransmitted
            ...0 .... = PWR MGT: STA will stay up
            ..0. .... = More Data: No data buffered
            .0.. .... = WEP flag: WEP is disabled
```

```
            0... .... = Order flag: Not strictly ordered
    Duration: 258
    Destination address: 00:09:41:41:90:98 (00:09:41:41:90:98)
    Source address: 00:09:41:1b:e9:77 (00:09:41:1b:e9:77)
    BSS Id: 00:09:41:41:90:98 (00:09:41:41:90:98)
    Fragment number: 0
    Sequence number: 917
IEEE 802.11 wireless LAN management frame
    Fixed parameters (6 bytes)
        Authentication Algorithm: Shared key (1)
        Authentication SEQ: 0x0001
        Status code: Successful (0x0000)

0000  b0 00 02 01 00 09 41 41 90 98 00 09 41 1b e9 77   ......AA....A..w
0010  00 09 41 41 90 98 50 39 01 00 01 00 00 00         ..AA..P9......
```

Zaglavlje drugog okvira izostavljeno je jer je jednako kao u prvom okviru.

```
IEEE 802.11 wireless LAN management frame
    Fixed parameters (6 bytes)
        Authentication Algorithm: Shared key (1)
        Authentication SEQ: 0x0002
        Status code: Successful (0x0000)
    Tagged parameters (130 bytes)
        Tag Number: 16 (Challenge text)
        Tag length: 128
        Tag interpretation: Challenge text:
\275P\266\341\024v9\341gY7\244\023\221\300\366@\325\3305\273\251\245\\304\
231x~\375\237\345r^;\214b\022\332\253\306\375#\232\371\374Z?

0000  b0 00 02 01 00 09 41 1b e9 77 00 09 41 41 90 98   ......A..w..AA..
0010  00 09 41 41 90 98 60 44 01 00 02 00 00 00 10 80   ..AA..`D........
0020  bd 50 b6 e1 14 76 39 e1 67 59 37 a4 13 91 c0 f6   .P...v9.gY7.....
0030  40 d5 d8 35 bb a9 a5 5c c4 99 78 7e fd 9f e5 72   @..5...\..x~...r
0040  5e 3b 8c 62 12 da ab c6 fd 23 9a f9 fc 5a 3f 30   ^;.b.....#...Z?0
0050  79 fe 41 3c 9c 71 5b f7 35 7b a5 fa 26 8a 14 57   y.A<.q[.5{..&..W
0060  17 74 20 fe b9 30 19 cc 57 82 a3 4d bd 48 45 23   .t ..0..W..M.HE#
0070  c4 b9 1c f9 a0 c0 ea 95 2e 75 03 81 70 9b e3 a4   .........u..p...
0080  da 19 02 59 13 09 c8 ac 97 af 5a 05 f9 e5 24 58   ...Y......Z...$X
0090  91 ae 2b c9 45 de e5 73 1e 61 b1 41 37 72 a1 90   ..+.E..s.a.A7r..
```

Zaglavlje trećeg okvira razlikuje se utoliko što se koristi WEP. Korišteni ključ kriptiranja ima vrijednost 12:34:56:78:90.

```
IEEE 802.11
    Type/Subtype: Authentication (11)
    Frame Control: 0x40B0 (Normal)
        Version: 0
        Type: Management frame (0)
        Subtype: 11
        Flags: 0x40
            DS status: Not leaving DS or network is operating in AD-HOC
mode (To DS: 0  From DS: 0) (0x00)
            .... .0.. = More Fragments: This is the last fragment
            .... 0... = Retry: Frame is not being retransmitted
            ...0 .... = PWR MGT: STA will stay up
            ..0. .... = More Data: No data buffered
            .1.. .... = WEP flag: WEP is enabled
            0... .... = Order flag: Not strictly ordered
    Duration: 258
    Destination address: 00:09:41:41:90:98 (00:09:41:41:90:98)
    Source address: 00:09:41:1b:e9:77 (00:09:41:1b:e9:77)
    BSS Id: 00:09:41:41:90:98 (00:09:41:41:90:98)
    Fragment number: 0
    Sequence number: 918
```

```
    WEP parameters
        Initialization Vector: 0x000004
        Key: 0
        WEP ICV: 0xbf479d86 (correct)
IEEE 802.11 wireless LAN management frame
    Fixed parameters (6 bytes)
        Authentication Algorithm: Shared key (1)
        Authentication SEQ: 0x0003
        Status code: Successful (0x0000)
    Tagged parameters (130 bytes)
        Tag Number: 16 (Challenge text)
        Tag length: 128
        Tag interpretation: Challenge text:
\275P\266\341\024v9\341gY7\244\023\221\300\366@\325\3305\273\251\245\\304\
231x~\375\237\345r^;\214b\022\332\253\306\375#\232\371\374Z?

Frame:

0000  b0 40 02 01 00 09 41 41 90 98 00 09 41 1b e9 77   .@....AA....A..w
0010  00 09 41 41 90 98 60 39 04 00 00 00 78 d8 0e 60   ..AA..`9....x..`
0020  e6 d2 a0 74 56 a3 b7 b8 44 da 14 21 fe 91 85 b6   ...tV...D..!....
0030  c8 62 bd 93 26 03 9e 55 1d b7 01 08 65 92 71 c5   .b..&..U....e.q.
0040  82 82 39 2e f7 80 88 7e a3 a3 ee e9 c1 f1 c9 9d   ..9....~........
0050  68 84 a9 3d a3 a2 36 c2 61 a3 01 53 3c 29 f8 eb   h..=..6.a..S<)..
0060  ab 18 76 d7 70 b2 b3 ea e5 0f e8 68 fb 4f 8f 81   ..v.p......h.O..
0070  9a 98 e2 00 71 6d 2b 2d a8 dc 60 31 98 4c 00 89   ....qm+-..`1.L..
0080  d2 91 6d 0f 12 5a 5d 59 24 90 86 29 bd c0 c9 b6   ..m..Z]Y$..)....
0090  d2 e7 0e d3 56 a0 4b 51 5d 3e 4e c9 8e 42 33 95   ....V.KQ]>N..B3.
00a0  29 19 a0 e8 bf 47 9d 86                         )....G..

Decrypted WEP data:

0000  01 00 03 00 00 00 10 80 bd 50 b6 e1 14 76 39 e1   .........P...v9.
0010  67 59 37 a4 13 91 c0 f6 40 d5 d8 35 bb a9 a5 5c   gY7.....@..5...\
0020  c4 99 78 7e fd 9f e5 72 5e 3b 8c 62 12 da ab c6   ..x~...r^;.b....
0030  fd 23 9a f9 fc 5a 3f 30 79 fe 41 3c 9c 71 5b f7   .#...Z?0y.A<.q[.
0040  35 7b a5 fa 26 8a 14 57 17 74 20 fe b9 30 19 cc   5{..&..W.t ..0..
0050  57 82 a3 4d bd 48 45 23 c4 b9 1c f9 a0 c0 ea 95   W..M.HE#........
0060  2e 75 03 81 70 9b e3 a4 da 19 02 59 13 09 c8 ac   .u..p......Y....
0070  97 af 5a 05 f9 e5 24 58 91 ae 2b c9 45 de e5 73   ..Z...$X..+.E..s
0080  1e 61 b1 41 37 72 a1 90                         .a.A7r..
```

Zaglavlje posljednjeg okvira slično je kao u prva dva okvira (ne koristi se enkripcija).

```
IEEE 802.11 wireless LAN management frame
    Fixed parameters (6 bytes)
        Authentication Algorithm: Shared key (1)
        Authentication SEQ: 0x0004
        Status code: Successful (0x0000)

0000  b0 00 02 01 00 09 41 1b e9 77 00 09 41 41 90 98   ......A..w..AA..
0010  00 09 41 41 90 98 80 44 01 00 04 00 00 00         ..AA...D......
```