Univerza v Ljubljani

PRETOČNI ŠIFRIRNI POSTOPKI

seminarska naloga pri predmetu porazdeljeni informacijski sistemi in celovitost podatkov

mentor: dr.Sašo Tomažič, univ. dipl. inž.

avtor: Franci Logar, univ. dipl. Inž.

Ljubljana, maj 2006

KAZALO

POVZETEK

V seminarski nalogi sem opisal pretočne šifrirne postopke kot razred simetričnih šifrirnih postopkov, za katere je značilno predvsem šifriranje znak za znakom (oz. majhnih količin podatkov naenkrat) in stalno spreminjajoča transformacijska funkcija. Uporabljamo jih na lahko na različnih OSI nivojih povsod tam, kjer so zahteve po veliki prepustnosti šifriranja.

Pretočni šifrirniki temeljijo na ideji Vernanovega šifrirnika, ki je edini teoretično varen način šifriranja. Pri pretočnih šifrirnih postopkih se čistopis običajno šifrira kar z XOR mešanjem s psevdonaključnim nizom, ki je funkcija skrivnega ključa.

Pretočne šifrirne postopke lahko razdelimo v dve skupini. Prva skupina so strojni šifrirni postopki, ki jih odlikuje predvsem hitrost. Temeljijo na teoriji LFSR registerskih vezij in so bili v preteklosti široko uporabljeni. Njihova teorija pa je osnova tudi za razvoj programskih postopkov. Standardni GSM pretočni šifrirni postopek je imenovan A5/1, ki pa sedaj ni več varen. Algoritem TRIVIUM je nov, odlikujejo pa ga velika pretočnost in enostavna izvedba.

Druga skupina pretočnih postopkov so programski postopki (algoritmi), ki so se uveljavili še posebej z razvojem internetnih aplikacij. V nalogi sem opisal izdelavo več različnih postopkov. SEAL je eden izmed prvih predlaganih postopkov, zanimiv pa je zaradi mešanja ključa in sekvenčne številka z zgoščevalno funkcijo in zaradi velike hitrosti šifriranja. PHELIX je nov predlog odlikujejo pa ga hitrost in dvojna funkcionalnost (zasebnost in verodostojnost). RC4 je najuspešnejši pretočni algoritem, saj se je uveljavil v veliko standardnih aplikacijah na področju komunikacij in informatike.

Projekt eSTREAM je najnovejši poskus razvoja novih varnih pretočnih algoritmov. Dosedanja testiranja ki so se končala februarja 2006 so izločila 2 tretjini kandidatov, ostali pa gredo v drugo fazo testiranj, kjer bo v ospredju predvsem varnost. Oba kandidata FHELIX in TRIVIUM, ki sem jih opisal v seminarski nalogi sta se uvrstila v drugi krog in trenutno še ni znan noben uspešen napad nanje.

UVOD

Pretočni šifrirni postopki (ang. stream ciphers) so pomemben razred simetričnih šifrirnih postopkov, ki so teoretično izredno dobro analiziran in omogočajo zelo varne šifrirne gradnike (primitive). Uporaba pretočnih šifrirnih postopkov ima v kriptografiji dolgo zgodovino . Najslavnejši stroj z zasnovo pretočnega šifrirnika pa je zagotovo nemška Enigma iz druge svetovne vojne. Tudi znameniti 'rdeči telefon' med Moskvo in Washingtonom v času hladne vojne je bil pretočni šifrirnik.

Najpomembnejša lastnost pretočnih algoritmov je šifriranje čistopisa bit za bitom, ki je za velikostni razred hitrejše od sorodnih bločnih šifrirnikov. Pretočni algoritmi uporabljajo časovno spreminjajočo šifrirno tranformacijo in temeljijo na ideji šifrirnika z enkratno uporabo ključa (ang. one time key pad).

Pretočni algoritmi se v aplikacijah, kjer se zahteva velik bitni pretok in kjer so omejene pomnilniške zmogljivosti, zato so še posebej primerni za uporabo v telekomunikacijskih omrežjih. Še ena lastnost pretočnih šifrirnikov pogojuje njihovo oporabo v telekominikacijskih omrežjih in sicer majhna soodvisnost napak(ang. error propagation), ki je nujna v okoljih z veliko motnjami (npr. brezžična omrežja).

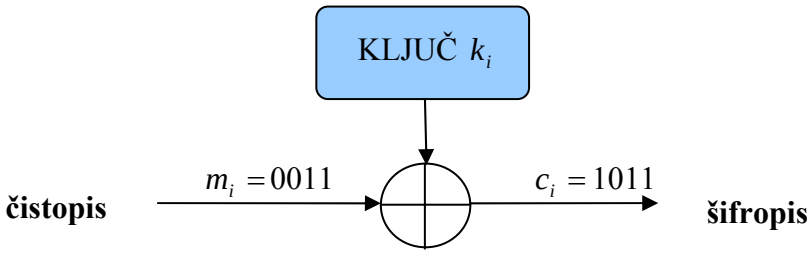
Čeprav je področje pretočnih algoritmov teoretično izredno dobro pokrito, pa je bilo do zdaj objavljenih zelo malo standardov iz tega področja, saj so bili pretočni algoritmi v preteklosti predvsem lastniški in njihova notranja zgradba vsaj poslovna, če že ne državna skrivnost. Sodoben trend uporabe teh postopkov v telekomunikacijah pa zahteva pretočne postopke, katerih zgradba ne bila skrivnost, kljub temu pa mora biti varnost samega postopka še vedno zelo visoka.

V prvem delu seminarske naloge je opisan osnovni princip zgradbe pretočnih šifrirnikov. V drugem delu je podana gradnja psevdonaključnih generatorjev, ki so hkrati najpomembnejši gradnik vseh strojnih pretočnih postopkov. Strojni pretočni postopki temeljijo na gradnji z registerskimi vezji oz. LFSR vezji. V tretjem delu so predstavljeni programski pretočni postopki in še posebej najpomembnejši algoritem RC4 in njegovo uporabo v telekomunikacijah in informatiki. V zadnjem poglavju sem opisal razvoj pretočnih postopkov, kjer prav zdaj poteka velik projekt imenovan eSTREAM. Sam projekt ni pomemben zaradi nove standardizacije (čeprav naj bi k njej tudi pripomogel), pač pa predvsem zaradi načina dela, ki nam predstavi problematiko in prednosti sodobnih pretočnih algoritmov.

OSNOVNA TEORIJA PRETOČNIH POSTOPKOV

Pretočni šifrirni algoritem je postopek, kjer posamezne elemente čistopisa šifriramo in dešifriramo s pomočjo neke naključne oz. psevdonaključne sekvence. Značilnost pretočnih algoritmov je šifriranje podatkovnega niza ali sporočil (kot so okviri in IP paketi) bit za bitom (v programskih algoritmih pa šifriramo majhne podatkovne enote npr. bajt). Šifriranje lahko poteka preko različnih operacij, vendar se je v praksi uveljavila predvsem enostavna operacija XOR (modulo 2 seštevanje) kot je prikazano na sliki 1. Kjer so elementi m_i, c_i in k_i posamezni biti šifropisa, čistopisa in psevdonaključnega niza oziroma ključa. Pretočni

šifrirniki imajo še eno značilnost in sicer 'spomin' torej neko notranje stanje, ki vpliva na transformacijsko funkcijo, ki se ves čas spreminja.



slika 1 – enostaven princip pretočnega šifriranja

Na ta način se problematika izdelava pretočne algoritma v bistvu spremeni v problematiko kako ustvariti psevdonaključni generator, katerega sekvenca se po statističnih lastnostih ne bo razlikovala od naključnega niza bitov. Najpomembnejši je seveda ključ, katerega lastnosti se bodo izražale tudi v lastnostih šifropisa, ki mora biti čimbolj naključen.

Vernanov šifrirnik

Ena izmed prvih metod pretočnega algoritma je bil Vernam-ov algoritem, ki so ga razvili v prvih desetletjih 20. stoletja z namenom kriptiranja teleprinterskih sporočil [2]. Ta algoritem je deloval po principu iz slike 1, za ključ pa je uporabil dovolj velik krožni trak s katerim je šifriral svoje čistopise. Varnost takega postopka je odvisna od dolžine psevdonaključnega niza. Če pa bi bili posamezni biti niza povsem naključni, potem bi tak šifrirnik imenovali enkratni šifrirnik (ang. one time pad).

Shannon je dokazal, da je za brezpogojno varnost simetričnega šifrirnega postopka potrebno izbrati tak ključ, katerega entropija bo večja ali enaka entropiji čistopisa - samo na ta način lahko zagotovimo teoretično brezpogojno varnost komunikacije, saj bi bila naključnost šifropisa neodvisna od naključnosti čistopisa. Iz teh teoretičnih idej se je rodila ideja o pretočnem šifrirnem algoritmu, ki se enkratnemu šifrirniku po varnosti lahko samo zgleduje, nikoli pa je ne bo dosegel.

Zgoraj navedena ugotovitev pa nakazuje tudi na to, da mora biti dolžina ključa enaka vsaj dolžini čistopisa. Ta zahteva pa je v simetričnem kriptosistemu nezaželena zaradi težav pri hranjenju in distribuciji ključev (velik pomnilnik, slaba izkoriščenost prenosa), zato v pretočnih šifrirnih algoritmih vedno uporabljamo psevdonaključni niz, ki izhaja iz nekega manjšega skrivnega ključa. Problem psevdonaključnih nizov pa je v tem da so periodični, kar bi pomenilo da bi se lahko dve sporočili šifrirali z istim ključem. V tem primeru bi napadalec lahko izvedel matematično operacijo:

$$c_1 \otimes c_2 = (K \otimes m_1) \otimes (K \otimes m_2) = (K \otimes K) \otimes (m_1 \otimes m_2) = m_1 \otimes m_2$$

Na ta način bi apadalec dobil pomembno informacijo o lastnostih obeh čistopisov, če pa bi vedel še kakšen promet poteka po komunikacijski poti (npr. okvir, paket) bi takoj pridobil informacijo o skrivnem kjuču in s tem bi bil algoritem razbit.

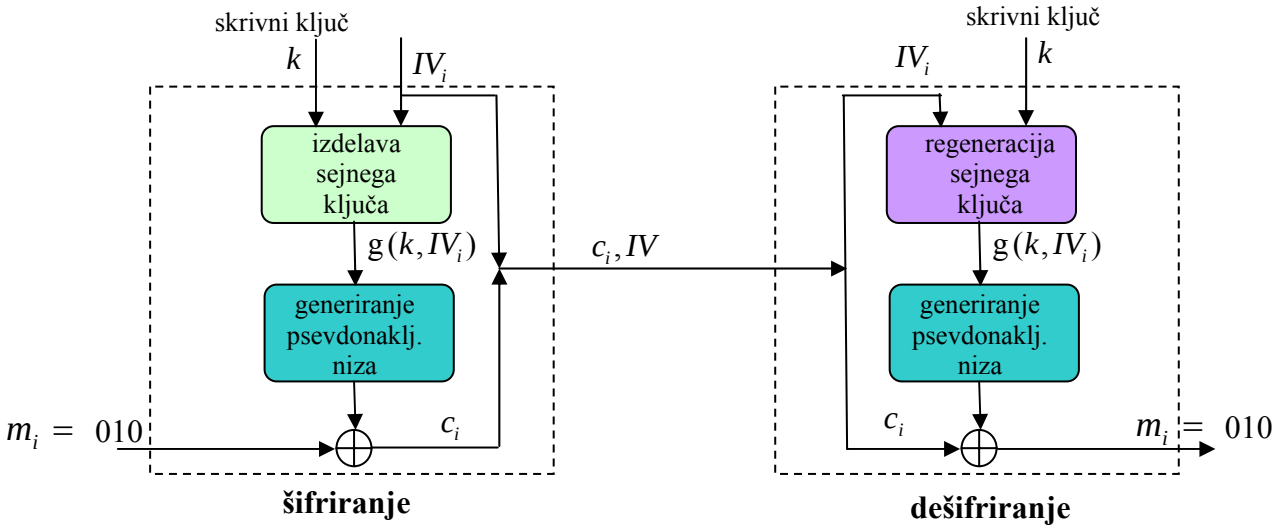
V sodobnih telekomunikacijah je prvo pravilo pretočnega šifriranja, da se sporočil nikoli na šifrira z istim ključem. Ker pa je to pravilo zelo zahtevno se končni sejni ključ razdeli na dva

dela. Prvi del je interni skrivni ključ k, ki je skrivnost, ki si jo delita obe strani in si jo tudi izmenjujeta po nekem varnem kanalu (npr. asimetrični postopek za izmenjavo ključev, itd). Drugi del ključa, pa je neka javno znana vrednost in služi za to, da se vsako sporočilo šifrira in prenaša z nekoliko drugačnim ključem. Ta javna številka ima v literaturi različna imena največkrat so označuje kot sekvenčna številka, inicializacijski vektor, inicializacijska vrednost (IV) ali pa kar z angleško kratico 'nonce' (ang. number used once). Pravila, na kašen način naj se ustvarjajo te oznake sporočil ni. Največkrat se prenašajo zaporedno, kot števci sporočil ali pa psevdonaključno pa nekem dodatnem algoritmu. S tem smo spremenili zgornje pravilo v naslednjega: Sporočil se nikoli ne šifrira z istim parom (k_i, IV_i). Kar nam pove da se lahko z enim skrivnim ključem pošlje samo določeno število sporočil.

Oba dela ključa se nato na sprejemnikovi strani združita in tvorita končni sejni ključ. Ta združitev pa v varnih algoritmih ni preprosto spajanje, ampak neka kompleksna funkcija, s katero nastavimo notranje stanje algoritma in ga pripravimo za šifriranje.
Na sliki 2 je predstavljena blok shema sodobnega pretočnega šifrirnika.

Tak niz seveda ni več naključen, zato tudi ne more zagotavljati absolutne varnosti komunikacije. Kljub temu se lahko z ustreznimi tehnikami oblikuje psevdonaključni niz, ki bo imel tako zelo podobne statistične lastnosti, da ga morebitni napadalec v nekem kritičnem času z omejenimi sredstvi ne bo mogel prepoznati [27].

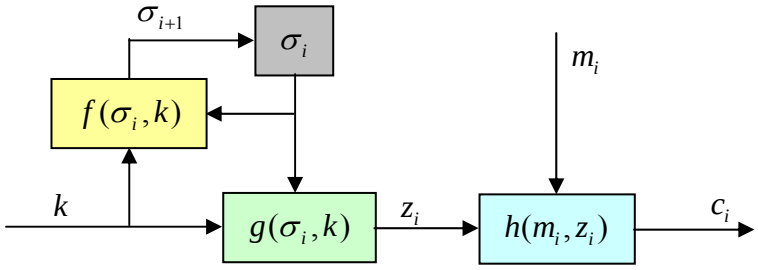
Kriptoanaliza pretočnih šifrirnih postopkov se ukvarja predvsem z dvema vprašanjema: prvo vprašanje je kako dolg niz šifropisa potrebujemo, da ga ločimo od povsem naključne bitne sekvence. Drugo vprašanje je na kakšen način bi uganili notranje stanje postopka in s tem pridobili informacijo o skrivnem ključu. S podrobno kriptoanalizo postopkov se v tej seminarski nalogi ne bom ukvarjal.



Slika 2 – pretočni šifriranje s pomočjo vrednosti IV

sinhroni – asinhroni pretočni algoritmi

Poznamo sinhrone in asinhrone pretočne postopke. Pri sinhronih šifrirnikih je ključ povsem neodvisen od šifropisa ali čistopisa. Posamezne vrednosti pa bi bile podane s funkcijami v sliki 3, kjer je narisana tudi blok shema sinhronega šifrirnika. Funkcija $f()$ je takoimenovana funkcija naslednjega stanja (ang. next state function), funkcija $g()$ je funkcija, ki proizvaja psevdonaključni niz z_i, $h()$ pa je funkcija, ki dejansko šifrira ta niz s čistopisom in se imenuje izhodna funkcija simetričnega pretočnega algoritma. σ_i je začetno notranje stanje oz. začetna vrednost pretočnega postopka. Na sprejemni strani so v prisotne enake funkcije, samo da se šifropis dešifrira z obratno funkcijo $h^{-1}()$.

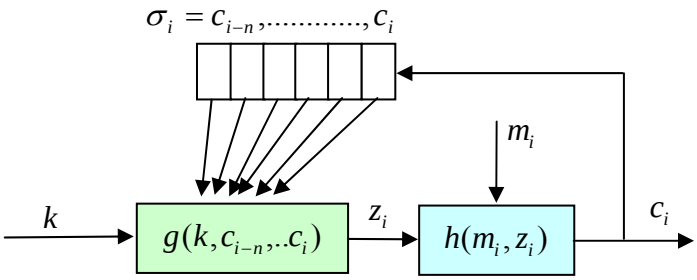


slika 3 – splošna shema sinhronega pretočnega algoritma

Sinhroni pretočni algoritem ima določene lastnosti. Prva in najbolj očitna lastnost je potreba po sinhronizaciji. Sprejemnik in oddajnik morata biti povsem sinhronizirana, uporabljati morata enak ključ in enako začetno stanje. Če pride do izgube pri prenosu aktivnega napada (torej se biti nizu dodajajo ali odvzemajo) se sinhronizacija takega algoritma poruši in potrebno je izvesti posebno proceduro za resinhronizacijo, ki se izvede z dodatnimi markerji znotraj šifropisa ali pa s poizkušanjem vseh začetnih stanj. En način resinhronizacije je prikazan že na sliki 2. Druga lastnost sledi iz prve in sicer občutljivost na aktivne napade, kjer bi napadalec lahko opazoval učinke svoje aktivnosti. Tretja lastnost sinhronih pretočnih algoritmov je neširjenje napake – če se pokvari en bit šifropisa, se s tem ne ogrozi nobenega drugega bita.

Večina do sedaj predlaganih sinhronih pretočnih algoritmov so bili binarni aditivni šifrirniki, kar pomeni da so imeli za vhode binarne nize, šifrirna funkcija $h()$ pa je bila kar XOR funkcija. V praksi se bolj uporabljajo sinhroni pretočni algoritmi.

Za asinhroni oz. samosinhronizacijski pretočni algoritem velja, da je izhodni šifropis funkcija ključa in fiksnega števila preteklih bitov šifropisa. Na sliki 4 je prikazana blok shema asinhronega šifrirnika, kjer je σ stanje v registrih, $g()$ je funkcija psevdonaključnega generatorja z izhodom Zi, h() pa izhodna funcija, ki šifrira čistopis. Tak pretočni postopek zahteva tudi začetno stanje σ_0. Trenutno najpogostejši asinhroni pretočni šifrirniki so v bistvu bločni šifrirniki v enobitnem povratnem načinu (CFM – cipher feedback mode).



slika 4 – splošna shema asinhronega pretočnega algoritma

Tudi asinhroni pretočni šifrirniki imajo več značilnosti. Prva značilnost je samosinhronizacija ob morebitni izgubi sinhronizacije. Pri tem procesu se ob dešifriranju izgubi določen fiksen del bitov, potem pa je šifrirnik spet v sinhronizmu z šifropisom. Druga značilnost, ki pa izhaja iz prve je omejeno širjenje napake. Posledice te lastnosti pa so iste, torej napačno dešifriranje n bitov. Tretja lastnost je občutljivost na aktivne napade, ki ima pozitivno in negativno lastnost. Aktivni napad je laže odkriti, ker sprememba enega bita povzroči več napačno dešifriranih sporočil, s čimer se poveča verjetnost da bo napad odkrit. Negativna lastnost glede napadov pa je samosinhronizem, ki na drugi strani omogoča modifikacijo šifropisa brez izgube sinhronizma. Zadnja lastnost je posledica odvisnosti izhoda od prejšnjih bitov, kar vpliva na statistične lastnosti čistopisa, ki se skrijejo oz. razpšijo po šifropisu. Na ta način je tak šifrirnik varnejši pred napadom z redundanco čistopisa (ang. plaintext redundancy attack) [1].

Pretočne algoritme lahko razdelimo na strojne (hardverske) in programske (softverske) algoritme. V nadaljevanju bom opisal vsako skupino posebej.

PRETOČNI ALGORITMI NA OSNOVI ´FSR´ VEZIJ

Strojni pretočni algoritmi so v praksi dobro preiskušena veja kriptografije, ki je tudi teoretično izredno obdelana. Velika večina teh algoritmov temelji na ideji aditivnega pretočnega algoritma, ki za svoj ključ uporablja psevdonaključni generator (PN generator) na osnovi FSR vezij.

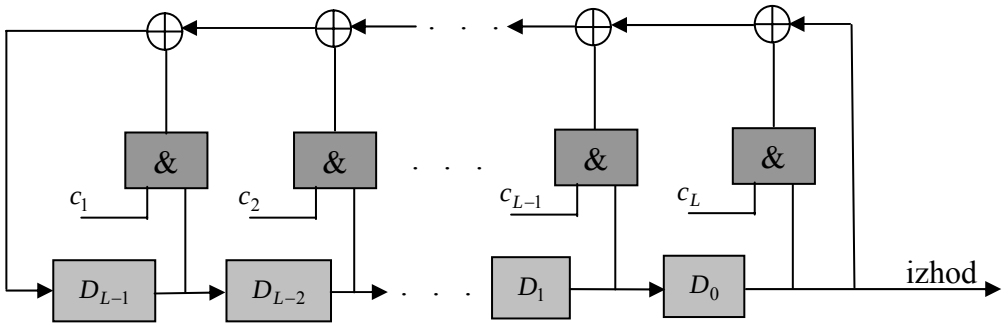
V nadaljevanju bom predstavil teorijo in prakso gradnje PN generatorjev, ki lahko s pomočjo enostavnih modifikacij tvori pravi strojni pretočni postopek z vsemi potrebnimi lastnostni (naključnost PN niza, IV vrednosti, inicializacije, itd.)

Vezja pomikalnih registrov s povratno zanko oz. FSR vezja (ang,. Feedback Shift Registers) so osnovni gradnik strojnih pretočnih šifrirnikov, saj njihove kombinacije omogočajo zelo poglobljeno matematično analizo svojih izhodov - torej bitnih sekvenc s psevdonaključnimi lastnostmi. V nadaljevanju bom predstavil linearna in nelinearna FSR vezja in nekatere tehnike, s katerimi izboljšamo statistične lastnosti njihovih psevdonaključnih nizov.

Linearna FSR vezja

Linearna FSR vezja oz. polinomski delilniki so uporabljeni v večini PN generatorjev zaradi enostavne strojne izvedbe (ter z njo povezanih zakasnitev in drugih ugodnih lastnosti), enostavne matematične analize delovanja in dobrih statističnih lastnosti njihovih psevdonaključnih sekvenc.

Polinomski delilniki oz. vezja LFSR(ang. Linear FSR) so zaporedno povezani pomikalni registri s povratno zanko. L-polinomski delilniki so sestavljeni iz L-stopenj registrov (D-flipflop), katerih izhode označujemo kot 0,1,…, L-1, kjer vsak shrani po eno binarno vrednost, ki jo ob ustreznem trenutku (proženje s taktom) prenese na svoj izhod. Na 0-ti stopnji pa je izhod že bit končne psevdonaključne sekvence. Na sliki 5 vidimo primer splošnega LFSR vezja, ki ima kar L povratnih vezav, kjer s pomočjo linearne operacije (XOR) določajo nov vhod na L-ti stopji FSR-vezja.



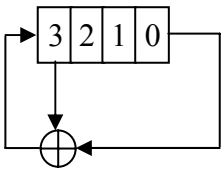
slika 5 – splošno linearno FSR vezje

Tak polinomski delilnik lahko opišemo z matematično enačbo oz. polinomom : $C(D) = 1 + c_1 D + c_2 D^2 + ... + c_L D^L$ z začetnim stanjem $s = [s_{L-1}, s_{L-2}, ..., s_0]$; $s_i \in \{0,1\}$

Edina omejitev pri začetnem stanju je, da ne sme biti sestavljeno iz samih ničel. Opisani polinomski delilnik ima zanimivo lastnost, da je pri danemu začetnemu stanju s_0 izhod delilnika enak

$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + ... + c_L s_{j-L}) \bmod 2 \quad \text{za vsak} \quad j \geq L$$

Za primer naj opišem konkreten polinom 4-stopnje, ki ga opišemo kot $C(D) = 1 + D + D^4$
Shema takega polinomskega delilnika je na sliki 6.



slika 6 – realno FSR vezje L=4

Konkreten potek izhodnih stanj takega delilnika je v tabeli 1, ki podaja časovni potek stanj t = 0,1,2,3,….. pri začetnem stanju FSR vezja $s = [0,1,1,0]$ na izhodih posameznih pomikalnih registrov. Izhod registra D_0, ki je hkrati tudi izhod iz polinomskega delilnika je podan s sekvenco $s = [0,1,1,0,0,1,0,0,0,1,1,1,1,0,1,....]$, ki pa je periodična, saj se sekvenca ponovi z natanko 15 stanji. Torej $s_L = s_{L+15}$

t	D3	D2	D3	D0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	0	1	1	0

začetno stanje

izhodna sekvenca

tabela 1 – prikaz vseh stanj v FSR vezju iz slike 5

Periodičnost izhodne sekvence ne velja za vsako registersko vezje, pač pa samo za posebna vezja, ki jih lahko opišemo s t.i. nesingularnimi polinomi. Z nesingularnimi polinomi pa opišemo vezja, ki imajo izhod v direktni povratni zanki torej člen $c_L \neq 0$. Za polinome z nižjimi stopnjami (torej brez povratne vezave na koncu) pa velja, da bodo imeli nekaj začetnih vrednosti izven kasnejše periode[1].

V teoriji polinomskih delilnikov obstaja veliko različnih lastnosti sekvenc, ki jih ustvarijo posamezni polinomi. Najpomembnejša je definicija primitivnega polinoma, ki nam omogoča izbor pravilnih členov povratne vezave, s čimer dosežemo tako psevdonaključno sekvenco, ki bo imela glede na stopnjo FSR vezja maksimalno dolžino. Dolžina sekvence ene periode ustvarjene s takim primitivnim polinomom je enaka $2^N - 1$ bitov, kar je zelo zanimivo za področje kriptografije saj nam lahko omejuje sposobnost šifrirnega sistema. V [1] je naveden algoritem za iskanje primitivnega polinoma poljubne stopnje, kot zanimivost so v tabeli 2 zbrani nekateri polinomi večjih potenc, katerih realizacija pa ni vedno praktična, saj zahtevajo dolgo resinhronizacijo.

Take sekvence izpolnjujejo nekatere pogoje za psevdonaključno sekvenco [1].

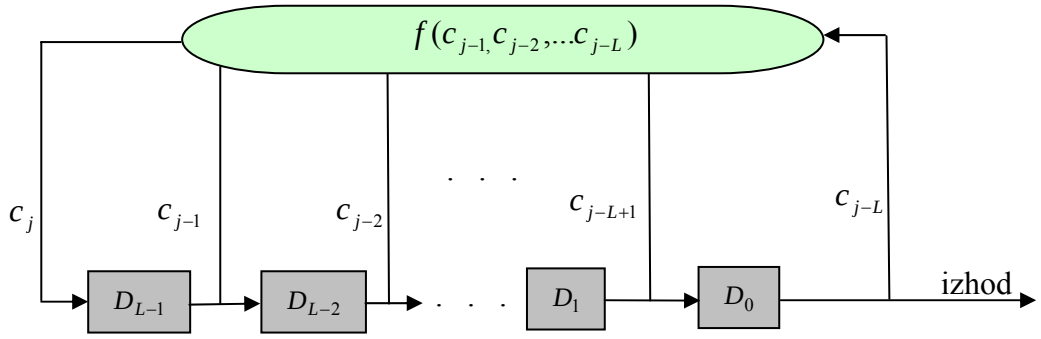
L	primitivni polinom	dolžina bitov v periodi
5	$1+D+D^4$	15
21	$1+D^3+D^{20}$	1048575
51	$1+D^{26}+D^{27}+D^{50}$	1125899906842623
101	$1+D^{37}+D^{100}$	1.26 E+30
151	$1+D^{53}+D^{150}$	1.42E+45
201	$1+D+D^{41}+D^{42}+D^{200}$	1.6E+60

tabela 2 – primitivni polinomi nekaterih višjih potenc

V teoriji se za opredeljevanje naključnih sekvenc uporabljamo še en pojem in sicer linearna kompleksnost, ki je definirana kot velikost polinomskega delilnika, s katerim bi se dalo tako psevdonaključno sekvenco izdelati. Linearna kompleksnost nekega LFSR vezja je manjša ali kvečjemu enaka številu registrov v tem vezju. Tudi linearna kompleksnost neke sekvence nam pove veliko o kriptografski uporabnosti take sekvence oz. generatorja sekvenc.

nelinearna FSR vezja

Nelinearna FSR vezja, so prav tako sestavljena iz verig pomikalnih registrov, le da v tem primeru med izhodi posameznih stopenj ne postavimo neke linearne operacije, ampak neko nelinearno operacijo Boolove algebre (npr. seštevanje po modulu n; n>2, rotiranje digitalne besede, itd.).



slika 7 – splošno nelinearno FSR vezje

Na sliki 7 lahko vidimo osnovni princip nelinearnega FSR vezja, kjer so izhodi posameznih stopenj preko povratne zanke v obliki funkcije $f(s_{j-1},s_{j-2},...s_{j-L})$ vezani na sam začetek verige. Izhodne sekvence pridobljene s takimi vezji izkazujejo podobne statistične značilnosti kot linearna vezja. Pravilno oblikovana funkcija pa lahko doseže tudi periode dolžin 2^n. Taka nelinearna FSR vezja imenujemo De Bruijn-ova FSR vezja, izhodne sekvence pa De Bruijn-ove sekvence.

UPORABA LFSR VEZIJ V KRIPTOGRAFIJI

Tehnike razbijanja linearnosti

FSR vezja se množično uporabljajo v strojnih pretočnih šifrirnikih za generatorje niza, ki v šifrirniku vnašajo ključ (ang. keystream) zaradi enostavne hardverske izvedbe, dolgih period psevdonaključnih sekvenc in dobrih statističnih lastnosti, ki jih le-te izkazujejo. Kljub temu pa take sekvence za sodobno kriptografijo še zdaleč niso dovolj varne. Napadalec, ki bi analiziral šifropis bi lahko z znanimi postopki (Berlekamp-Masseyjev algoritem) določil linearno kompleksnost sekvence in določil tudi LFSR vezje, ki tako sekvenco proizvaja. To bi lahko storil že iz 2L vzorcev (L je velikost FSR vezja). S primerjavo čistopisa in njihovega šifropisa (napad preko izbranega čistopisa oz. plaintext attack) pa bi dokončno določil začetno stanje generatorja oz. ključ. Operacija bi se zapisala: $m_i \oplus c_i; 1 \le i \le n$. Formula nam pove, da če napadalcu uspe rekontruirasti celotno sekvenco, je s tem pridobil tudi čištopis. Zato se v varnih aplikacijah vedno uporablja kompleksnejši sistem, ki pa je sestavljen iz enostavnih gradnikov opisanih zgoraj.

Psevdonaključna sekvenca realizirana z FSR vezji mora imeti določene lastnosti oz. potrebne pogoje (ne pa zadostne !) da je kriptografsko varna in sicer: dolga perioda, velika linearna kompleksnost (torej so FSR vezja opisana z primitivnim polinomom) in dobre statistične lastnosti. Zaradi enostavnosti izvedbe je zaželeno, da imajo FSR vezja čimmanj povratnih členov, s tem pa se zmanjšuje kriptografska vrednost samega generatorja, saj je tak generator bolj občutljiv na določene tipe napadov.

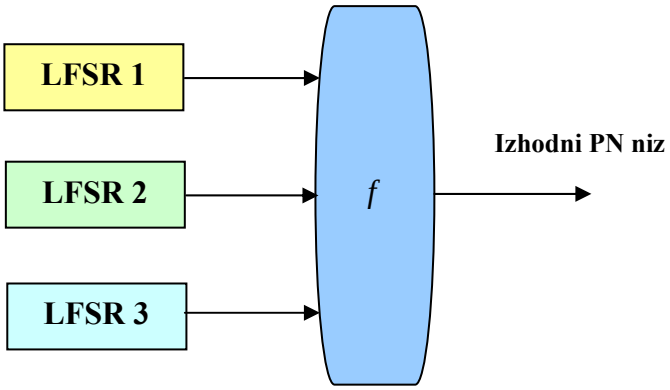
Obstajata dva principa uporabe FSR psevdonaključnih generatorjev in sicer skrivni in javni. Prvi princip je skrivni, kjer je zgradba (posamezne povratne vezave) samega FSR vezja tajna, tako kot je tajen tudi začetni ključ (začetno stanje FSR vezja). Drugi princip je javni, kjer pa je zgradna FSR vezja splošno znana, tajen je samo ključ.

Uporabljana sta oba principa, prvi je kriptografsko varnejši, drugega pa je lažje realizirati in je primernejši za javno uporabo. V splošnem je trend v telekomunikacijah uporaba javnega pretočnega algoritma, seveda pa mora biti algoritem dovolj kompleksen, da je zagotovljena določena stopnja varnosti, ki je napadalec z omejenimi sredstvi ne more ogroziti.

Ker mora biti sodoben šifrirni postopek odporen na napade ki so bili opisani zgoraj, se enostavne psevdonaključne generatorje preoblikuje z določenimi tehnikami, s katerimi bi bila izhodna psevdonaključna sekvenca precej robustnejša. Te tehnike so: nelinearno kombiniranje sekvenc, nelinearno filtriranje LFSR stanj in medsebojno proženje LFSR vezij. V nadaljevanju so podrobneje opisane vse tri tehnike, ki jih sodobni pretočni algoritmi tudi uporabljajo.

Nelinearno kombiniranje sekvenc

Ena od idej, kako izničiti linearnost LFSR vezij, je izbrati več takih vezij in izvesti neko nelinearno funkcijo (v Boolovi algebri) nad njihovimi sekvencami. Ideja takega psevdonakjučnega generatorja je dana na sliki 8, kjer je f podana kot nelinearna funcija imenovana tudi kombinacijska funkcija(ang. combining function). Da pa je izhodna sekvenca kriptografsko varnejša mora f ustrezati določenim kriterijem.



slika 8 – nelinearna kombinacija PN nizov

Prvi kriterij je visok nelineani red funkcije f. Vsaka Boolova funcija se da zapisati kot XOR produkt med členi sestavljenimi iz produktov (operacija AND) posameznih spremenljivk. Tak zapis imenujemo algebraična oblika funkcije. Tu se pojavi izraz nelinearni red (ang. non-linear order) funkcije f, ki je definiran kot maksimalno število spremenljivk, ki nastopa v členih funkcije. Primer: nelinearni red funkcije $f = x_2 x_4 + x_1 + x_1 x\ x_3$ je tri zaradi zadnjega člena v zapisu.

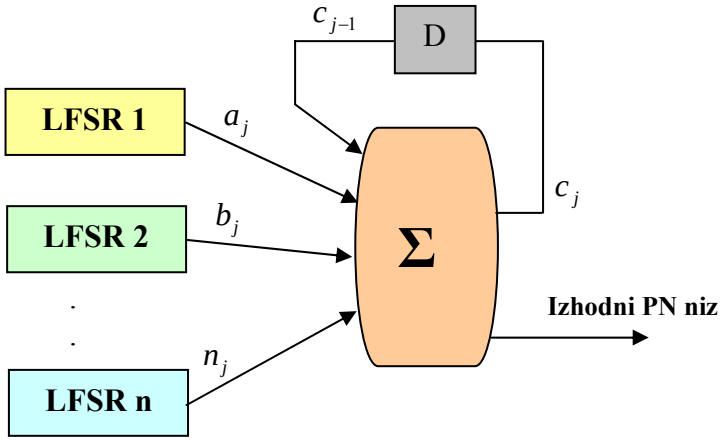
Če hočemo imeti nelinearni psevdonaključni generator visoke linearne kompleksnosti, moramo zato uporabiti funkcijo visokega nelinearnega reda [1].

Drugi kriterij za funkcijo f je odpornost proti korelacijskim napadom. Korelacijski napadi se uporabljajo v primeru, da so FSR vezja posameznih vhodov funkcije f in tudi sama funcija znani. V primeru da se za znano funkcijo f uporabi n vhodov z znanimi FSR vezji $R_1, R_{2,}, ..., R_n$, potem bi bilo skupno število ključev enako produktu vseh možnih stanj, torej teoretično $št.kljucev = \prod_{i=1}^{n} (2^{L_i} - 1)$. Če pa bi obstajala neka visoka soodvisnost med psevdonaključnim nizom in posameznimi vhodi funkcije (torej izhodi vezij R_i), bi se začetno stanje uganilo že po $\sum_{i=1}^{n} (2^{L_i} - 1)$ poskusih, kar je velik napredek pri dešifriranju nelinearnega generatorja psevdonaključnih signalov. Korelacijski napadi so zelo pogosta oblika napadov na sodobne šifrirne algoritme.

Zgoraj našteta dejstva nam nakazujejo, da je za psevdonaključni generator vsakega pretočnega postopka pomembno, da ni statistične odvisnosti med kratkimi FSR sekvencami in končnim psevdonaključnim nizom, kar merimo s testom korelacijske imunosti m-te stopnje. Ta test trdi, da je Boolova funkcija f z n vhodi korelacijsko imuna (stopnje m), če je za vsako kombinacijo podmnožice vhodnih spremeljivk $X_1, X_2, ... X_m; 1 < m < n$ korelacijska funkcija med $f(X_1, X_{2,...}, X_n)$ in to podmnožico enaka 0.

Zgornja lastnost pomeni, da pri kombinacijski funkciji vedno izbiramo kompromis med doseganjem visokega nelinearnega reda in visoko korelacijsko imunostjo. To nerodno lastnost kombinacijskih funkcij pa lahko izboljšamo, če ji dodamo tudi spominske elemente[1].

Primer takega nelinearnega generatorja je sumacijski generator, ki je podan na sliki 9. Sestavljen je iz n psevdonaključnih generatorjev, ki so vhodi v sumacijsko funkcijo, ta pa sešteva posamezne vhode skupaj s posebnim prenosnim (ang. carry) bitom, ki v kombinacijsko funkcijo vnaša spomin. Izhodni psevdonaključni bit se računa kot suma mod 2 vhodov. Izračunani prenosni bit pa vrne vrednost na C vhod. Vrednost C opredeli posebna formula.



slika 9 – sumacijski generator PN niza

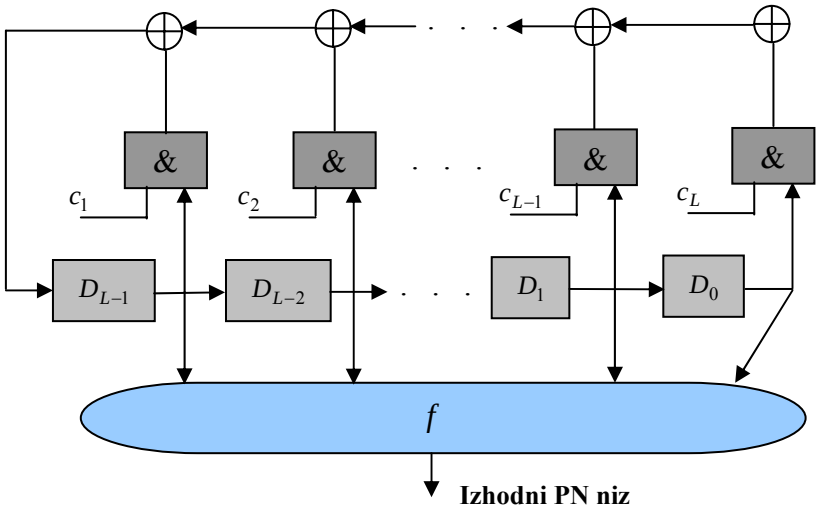
Obstajajo pa tudi določene zahteve za tako vezje: dolžine takih vezij med seboj ne smejo biti deljive. Skrivni ključ je sestavljen iz začetnih stanj posameznih LFSR vezij in začetnega 'spominskega' bita C_0. Taka kombinacijska funkcija bi imela veliko periodo, veliko linearno kopleksnost in korelacijsko imunost, seveda pa obstajajo tudi napadi zoper tak psevdonaključni generator.

nelinearno filtriranje LFSR stanj

Naslednja ideja, kako izničiti linearnost LFSR vezij je uporaba nelinearne funkcije znotraj enega LFSR vezja. Za vhode v to Boolovo funkcijo f uporabimo kar izhode posameznih pomikalnih registrov, funkcijo pa imenujemo filterska funkcija. Podrobna analiza te filterske funkcije nam pove, da bo izhodni psevdonaključni niz imel najboljše lastnosti, če bo imela algebraična oblika filtrske funkcije čimveč členov različnih redov tja do reda m. Splošen model te tehnike je predstavljen na sliki 10.

Primer takega generatorja je t.i. *knapsack* generator, ki je definiran z LFSR vezjem dolžine L (primitivni polinom) in številom $Q = 2^L$ (imenovan modulus). Skrivni ključ je sestavljen iz uteži $a_1, a_2, ...a_L$ (L-bitna cela števila) in začetnega stanja LFSR vezja. V času j se izračuna vrednost $S_j = \sum_{j=1}^{L} x_i a_i \bmod Q$, kjer so vrednosti x_i binarna stanja posameznih LFSR stopenj v

trenutku j. Ko je opravljena vsota spremenjena v binarno obliko, se ji še odvzame določene bite ostali pa tvorijo del izhodne psevdonaključne sekvence



slika 10 – uporaba nelinearne filterske funkcije

neenakomerno proženje LFSR vezij

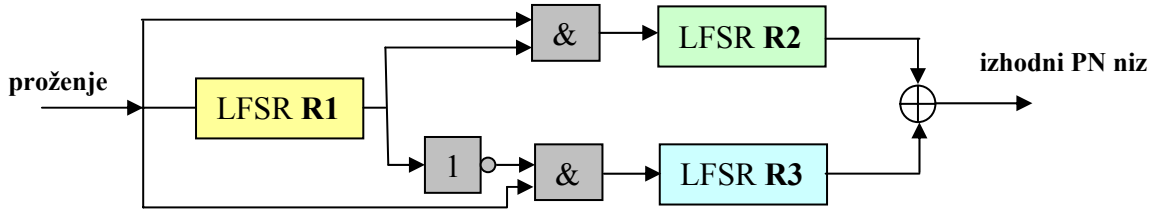
Tretji način izboljšave LSFR sekvenc pa je povezan z neenakomernostjo proženja v LFSR vezij. Za vsa linearna in nelinearna FSR vezja, ki so bila predstavljena do sedaj, velja da so bila prožena z enotno uro, ki poskrbi za sihroniziran pomik bitov po FSR vezju. Taka vezja imajo slabost, da so občutljiva na korelacijske napade. Ta lastnost je povzročila razvoj drugačnih vrst psevdonaključnih generatorjev, ki delujejo na principu medsebojnega proženja, torej izhod enega LFSR vezja (kontrolna sekvenca) določa kontrolni vhod za proženje drugega vezja oz. več drugih vezij(generirane sekvence). Na ta način lahko dobimo robustne gradnike za moderne pretočne algoritme.

V teoriji [5] je znanih več vrst generatorjev z neenakomernim proženjem:

- 'stop and go' generatorji
- kaskadni generatorji
- generatorji z alternirajočim korakom
- skrčevalni generatorji
- samoskrčevalni generatorji

Kot primer bom navedel dva tipa generatorjev z neenakomernim proženjem LFSR vezij.

Prvi tip je generator z alternirajočim korakom predstavljen na sliki 11. Ta generator vsebuje tri LFSR vezja ($R_1 - R_3$), kjer izhod prvega vezja proži delovanje drugih dveh. Izhodna sekvenca je XOR kombinacija sekvenc R_2 in R_3. Ker je proženje R_3 negirano glede na R_2 je XOR vratih vedno ena nova in ena stara vrednost.



slika 11 – generator z alternirajočim korakom

Najboljše psevdonaključne sekvence dobimo če so LSFR vezja (primitivni polinomi) približno enakih dolžin, za katere velja $\gcd(L_1, L_2)=1$; $\gcd(L_2, L_3)=1$ in $\gcd(L_1, L_3)=1$, kjer je gcd(a,b) podana kot največji skupni količnik dveh števil oz. v našem primeru dveh dolžin LSFR vezij. V času nastanka [1] leta 1997 je bila dolžina LFSR vezij približno 128 stopenj dovolj velika, da je bila odporna na takrat znane napade.

Drugi tip generatorja z medsebojnim proženjem pa je t.i. skrčevalni generator (ang. shrinking generator), ki je bil predlagan razmeroma pozno leta 1993, vendar je zaradi svoje enostavnosti dober kandidat za psevdonaključni generator v sodobnih hitrih pretočnih algoritmih.

Skrčevalni generator je sestavljen iz dveh LFSR vezij, ki sta proženi z isto uro, vendar dejansko izhodno sekvenco določa vrednost prvega generatorja R_1. Če je ta vrednost 1 potem se trenutna vrednost drugega generatorja R_2 (generirana sekvenca) zapiše na izhod , v primeru ničle pa se trenutna vrednost R_2 zavrže. Odtod generatorju tudi ime, saj je dolžina izhodne sekvence manjše od izvornega števila bitov generatorja R_2. Za urejeno sekvenco je potrebno proizvedene bite še sinhronizirati , običajno se to stori v pomnilniku.

Skrčevalni generator ima najdaljšo periodo, če je sestavljen iz maksimalnih LFSR in če je $gdc(L_1, L_2)=1$, kjer sta L_1 in L_2 dolžini LFSR vezij. Prav tako ima tak generator veliko linearno kompleksnost in statistično zelo dobre lastnosti izhodne sekvence. Varnost takega generatorja pogojujejo zgoraj našteti parametri, še bolj pa dejstvo ali je znana zgradba takih pretočnih algoritmov, torej ali so znane povratne povezave obeh LFSR vezij ali pa so te povezave tako kot začetna stanja tajni. Če sta dolžini LFSR registrov podobni potem veljajo za varni (leta 1997) dolžini približno 64 stopenj.

Tudi v sodobnejši literaturi velja ta generator za zelo robustnega, obstaja pa veliko variant istega generatorja, ki pa imajo podobne značilnosti [3]. Znana je tudi podvrsta z enim samim generatorjem t.i. samo skrčevalni generator (ang. self-shrinking gen.), ki je najučinkovitejši generator tega tipa [5], saj zahteva njegova izvedba najmanj registrov. Slaba lastnost skrčevalnih generatorjev je nihajoča dolžina izhodne sekvence, ki jo rešujemo s posebno sinhronizacijo v pomnilniku.

Tako skrčevalni kot generatorji z alternirajočim korakom imajo odlične statistične lastnosti, ki se lahko v nekaterih primerih še izboljšajo z uporabo nelinearnih FSR vezij. Ta vezja generirajo de Bruin-ove sekvence. Najboljše rezultate dobimo s kombinacijo, kjer za kontrolni register uporabimo LFSR (primitivni polinom), za generirani register pa FSR vezje [5].

Praktični primeri uporabe

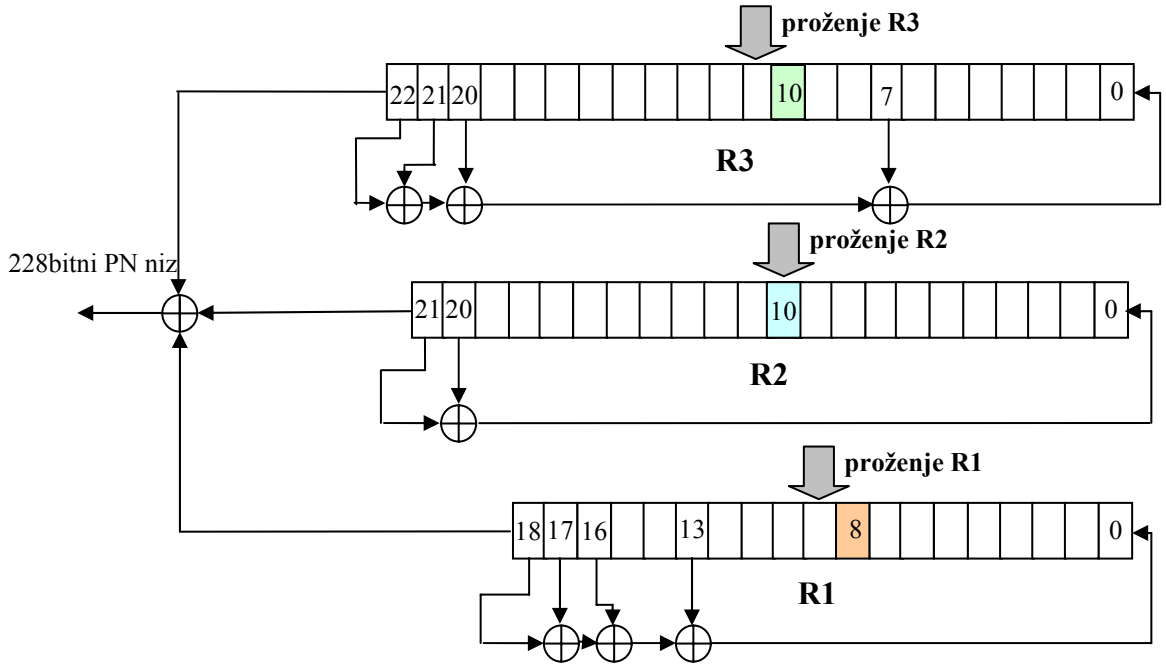
Kot primer znanega in standardiziranega pretočnega algoritma z neenakomernim proženjem bom podal algoritem A5/1, ki se uporablja v sistemu GSM. V nadaljevanju pa bom opisal algoritem, ki se šele standardizira in nakazuje moderen pristop k ustvarjanju strojnih pretočnih algoritmov.

A5/1

Pretočni postopek A5/1 so razvili leta 1989 kot standardiziran pretočni algoritem, ki se bo uporabljal za zaščito govora po brezžičnem mediju v sistemu digitalne telefonije GSM. Poleg 'močnejšega' A5/1 so razvili tudi 'šibkejši' algoritem A5/2, ki naj bi ga uporabljali v GSM sistemih izven ZDA in Evrope.

A5/1 je binarni aditivni pretočni algoritem, ki za svoje šifriranje potrebuje generator psevdonaključne sekvence narisan na sliki 12**.** Izhodni niz generatorja dobimo z XOR operacijo izhodov treh neenakomerno proženih LFSR vezij.

GSM pogovori potekajo v 228-bitnih okvirih, ki se med pogovorom izmenjujejo vsakih 4.6 milisekunde. Pri vsakem pogovoru se uporablja 64-bitni ključ (v resnici je 10 bitov 0, tako da je ključ samo 54-bitni) in 22-bitna sekvenčna številka (oz. začetna vrednost) okvira. Na ta način zagotovimo šifriranje vsakega okvira posebej s svojo enkratno kombinacijo ključa in začetne vrednosti.



slika 12 – PN generator A5/1

Proženje posameznega LFSR vezja pa določa poseben pogoj, ki je funkcija stanj na določenih bitih LFSR vezij. V našem primeru so na sliki ti biti označeni z barvami. Pogoj za proženje je

nastavljen tako, da je verjetnost proženja za vsako LFSR vezje $p = 0.75$, ob vsaki trenutku pa sta proženi vsaj dve LFSR vezji.

Začetna inicializacija se začne z nastavitvijo vseh vrednosti na 0. Nato se v povratne zanke s funkcijo XOR vnese vse bite ključa. V nadaljevanju se v povratne zanke registrov vnese še 22 bitov sekvenčne številke. Nato se alroritrm začne prožiti. Najprej se zavrže prvih 100 bitov izhodne sekvence in nato je algoritem pripravljen za delovanje. Izhod algoritma je 114 bitov za šifriranje v eno smer in drugih 114 bitov za drugo smer.

Omenjeni pretočni algoritem je izveden v večini današnjih GSM telefonov in je bil tajen do leta 1994, ko se je prvič pojavila predstavljena shema, celoten algoritem pa so dokončno razkrili z inverznim inženiringom leta 1999. Od tedaj ima A5/1 zelo omejeno kriptografsko vrednost, treba pa je poudariti, da sam algoritem že v osnovi ni bil mišljen kot maksimalno varen.

Na področju digitalne telefonije se je uveljavil UMTS pretočni postopek KASUMI, ki za zasebnost poskrbi s svojim pretočnim načinom 'f8'. KASUMI vsebuje 128-bitne ključe in 32-bitno sekvenčno številko. Po svoji zasnovi je to bločni algoritem, zato ga podrobno ne bom opisoval.

TRIVIUM

Za konec poglavja o strojnih pretočnih algoritmih (torej algoritmih zgrajenih na podlagi FSR vezij) bom podal še en zanimiv algoritem, ki je še zelo nov in v kriptografski znanosti še nepreizkušen, vendar dosedanji testi kažejo zelo dobre rezultate.

TRIVIUM je bil predlagan v okviru projekta eSTREAM, kot strojno zasnovan pretočni algoritem. Nastal je na podlagi vprašanja, kako enostaven algoritem bi se dalo izdelati, ne da bi žrtvovali njegovo hitrost, varnost in fleksibilnost.

Po lastnostih je TRIVIUM strojni sinhroni pretočni algoritem, ki uporablja 80-bitni ključ in 80-bitno začetno vrednost (IV), sestavljen pa je iz 288-stopenjskega FSR vezja z nelinearno povratno zanko. Tak generator lahko ustvari psevdonaključno sekvenco z dolžino do 2^{64} bitov.

Sam proces je kot v večini pretočnih algoritmov razdeljen v dve fazi. Prva faza je sama inializacija LFSR vezij, kjer na podlagi tajnega ključa in IV postavimo šifrirnik v določeno začetno stanje. Druga faza pa je izračunavanje izhodnega psevdonaključnega niza, ki ga nato uporabimo za klasično aditivno šifriranje. Kljub končni strojni izvedbi algoritma pa bom delovanje opisal na podlagi programskih modulov.

Inicializacija

Proces inicializacije poteka z nalaganjem skrivnega ključa $K = \{K_1,...K_{80}\}$ in znanega IV (ali sekvenčne številke) $IV = \{IV_1,...,IV_{80}\}$ v registre FSR vezja $s_1, s_2,..., s_{288}$. Celoten modul je predstavljen na sliki 13.

$(s_1, s_2, ..., s_{93}) \leftarrow (K_1, K_2, ..., K_{80}, 0, ..., 0)$

$(s_{941}, s_{95}, ..., s_{177}) \leftarrow (IV_1, IV_2, ..., IV_{80}, 0, ..., 0)$

$(s_{178}, ...s_{285,} s_{286}, s_{287}, s_{288}) \leftarrow (0, ...0, 1, 1, 1)$

For i=1 to 4*288 % začetno nastavljanje stanj

$t_1 \leftarrow s_{66} \oplus s_{91} \,\&\, s_{92} \oplus s_{93} \oplus s_{171}$

$t_2 \leftarrow s_{162} \oplus s_{175} \,\&\, s_{176} \oplus s_{177} \oplus s_{264}$ % začasne vrednosti

$t_3 \leftarrow s_{243} \oplus s_{286} \,\&\, s_{287} \oplus s_{288} \oplus s_{69}$

$(s_1, s_2, ..., s_{93}) \leftarrow (t_3, s_1, ..., s_{92})$

$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$

$(s_{178}, s_{179}, ..., s_{288}) \leftarrow (t_2, s_{178}, ..., s_{287})$

slika 13 – inicializacija algoritma TRIVIUM

Izračun psevdonaključne sekvence

Ko je TRIVIUM postavljen v potrebno stanje glede na ključ in IV, je potrebno z nelinearno povratno vezavo izračunati bit izhodne sekvence, nato pa operacijo ponavljamo, dokler ne dobimo dovolj dolge sekvence za šifriranje sporočila. Izračun izhodnega bita je predstavljen na sliki 14.

For i to N do

$t_1 \leftarrow s_{66} \oplus s_{93}$

$t_2 \leftarrow s_{162} \oplus s_{177}$

$t_1 \leftarrow s_{2436} \oplus s_{288}$

$Y_i = t_1 \oplus t_2 \oplus t_3$ % izračun izhodnega bita

$t_1 \leftarrow t_1 \oplus s_{91} \,\&\, s_{92} \oplus s_{171}$

$t_2 \leftarrow t_2 \oplus s_{175} \,\&\, s_{176} \oplus s_{264}$

$t_3 \leftarrow t_3 \oplus s_{286} \,\&\, s_{287} \oplus s_{69}$

$(s_1, s_2, ..., s_{93}) \leftarrow (t_3, s_1, ..., s_{92})$

$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$

$(s_{178}, s_{179}, ..., s_{288}) \leftarrow (t_2, s_{178}, ..., s_{287})$

slika 14 – iteracijska zanka za izhodni bit

Ker je TRIVIUM algoritem za strojno izvedbo je bitno orientiran. Za še hitrejšo izvedbo se lahko iteracije izvajajo vzporedno, kar pa bi lahko vplivalo na nelinearnost zanke (s tem bi poslabšali varnost algoritma). Da zavarujemo nelinearnost zanke, je treba predpisati, da se nobeno notranje stanje ne uporabi še vsaj 64 iteracij za tem, ko je bilo modificirano [22]. Z

vzporednim delovanjem lahko dosežemo hitrost bajt izhodne sekvence na urino periodo, kar je vsaj 32-krat hitrejše kot hitri programski algoritmi (npr. SEAL, SCREAM). Na testiranjih za projekt eSTREAM so realizirali šifrirnik, ki je omogočal šifriranje pri 18Gbit/s [26].

Algoritem TRIVIUM je mogoče izvesti tudi programsko, seveda pa je taka izvedba precej počasnejša npr. 4 cikli na bajt izhodne sekvence, kar je po hitrosti primerljivo z hitrimi programskimi algoritmi, vendar so ti za isto hitrost sposobni realizirat 256-bitno varnost, kar je bistvena prednost pred TRIVIUM-om.

PROGRAMSKI PRETOČNI POSTOPKI

Pretočni postopki na osnovi SFR vezij so se izkazali za odlično rešitev za strojne šifrirnike, saj podpirajo prav tisti del funkcionalnosti, zaradi katerih se pretočne algoritme najbolj uporablja - to pa so aplikacije, kjer se zahtevajo največje hitrosti šifriranja.

V začetku devetdesetih let so se začele pojavljati programske izvedbe pretočnih algoritmov, kjer pa se algoritmi z idejo FSR vezij niso najbolje obnesli, zato se je naglo začel razvoj drugačnih algoritmov, ki so se izkazali za uporabne pri šifriranju na višjih OSI nivojih. Programski pretočni algoritmi za svoje operacije uporabljajo sodobne mikroprocesorje, zato se mora proces generacije psevdonaključnega signala izvajati v bajtih oz. celo v besedah, ki ustrezajo konfiguraciji mikroprocesorja (za sodobne procesorje najbolje kar v 32 ali 64 bitnih besedah).

Programsko učinkovitost pretočnega algoritma merimo v procesorskih ciklih, potrebnih za izdelavo enega bajta psevdonaključne sekvence. Na koncu seminarske naloge sem podal tudi konkretno primerjavo med nekaterimi pretočnimi postopki.

V tem poglavju bom naštel nekatere izvedbe, ki imajo tudi uporabno vrednost, treba pa je poudariti da so nekateri algoritmi novi in da se bo njihova prava kriptografska vrednost izkazala šele v prihodnosti.

SEAL

Pretočni algoritem SEAL (Software-optimized Encryption ALgorithm) sta prvič objavila R. Rogaway in D. Coopersmith leta 1993. Kasneje sta avtorja objavila še dve verziji istega algoritma (SEAL2.0 in SEAL 3.0), ki omogočata večjo kriptografsko varnost. Razlike med algoritmi so v smislu programske izvedbe zelo majhne. Algoritem se je izkazal za zelo učinkovito idejo, kar pričajo številne nadaljnje raziskave in izboljšave.

SEAL je binaren aditiven pretočni algoritem, ki s pomočjo sekvenčne številke (n) in 160-bitnega skrivnega ključa (a) ustvari močno psevdonaključno sekvenco s katero nato aditivno šifriramo sporočilo. Matematično bi tako operacijo zapisali: $c = m \oplus SEAL(a, n, L)$, kjer je c šifropis, m je čistopis, L želena dolžina sporočila, n pa prej omenjena sekvenčna številka (ali IV pri drugih algoritmih). Zaradi same zgradbe algoritma tak niz ne more imeti poljubne dolžine, pač pa ima lahko samo dolžine L'=m*128bitov>L, kjer je m celo število. Podobno velja tudi za ostale programske postopke.

SEAL obsega tri module: predhodna izdelava tabel, inicializacija in glavna zanka, ki jih bom v nadaljevanju podrobneje opisal. Algoritem v predhodni fazi s pomočjo funkcije $G_a(i)$ razširi skrivni ključ v velike tabele(R[],S[],T[]), ki jih nato po inicializaciji v kasnejši glavni fazi uporablja za operacije nad 32-bitnimi besedami. Rezultat glavne faze pa je že del izhodne psevdonaključne sekvence.

Še prej pa bom podal oznake za posamezne operacije, ki bodo opisane v modulih.
'⊕' - seštevanje po modulu 2 ali XOR,
'+' -seštevanje po modulu 32 (torej bitno seštevanje dveh 32 bitnih besed brez prenosa)
'←' – prireditev vrednosti
' >>x' - rotacija registra za x mest v desno
'‖' - spajanje bitnih izrazov (npr. 32 bitno besedo Z dobimo: Z = x‖ y‖ z‖q s štirimi bajti)
'&' - boolova operacija IN
'0x' - oznaka za hexadecimalni zapis
'A, B, C, D, P, Q' - 32 bitni registri

1. modul izračun tabel S[ ], T[ ] in R[ ]

Tabele S[], T[], R[] se lahko izračunajo takoj, ko je znan skrivni 160-bitni ključ a. Te tabele dobimo iz uporabe zgoščevalne funkcije SHA-1. Funkcijo imenujemo $G_a(i)$, njen rezultat pa je 160 bitna vrednost, definirana za vsak iteracijski korak i. Nato definiramo funkcijo $F_a(i) = H^i_{i \bmod 5}$; kjer je $G_a(i) = H_0^{5j} \| H_1^{5j+1} \| H_2^{5j+2} \| H_3^{5j+3} \| H_4^{5j+4}$ pri $j = \lfloor i/5 \rfloor$; torej je tabela vrednosti $F_a(i)$ ravno tabela $G_a(i)$ brana od leve proti desni, od zgoraj navzdol. Sedaj lahko na sliki 15 definiramo vrednosti tabel s zanko:

for i od 0 do 511 T[i] ← $F_a(i)$

for j od 0 do 255 S[j] ← F_a(0x1000+j) in

for k od 0 do 4 [(L -1)/8192] -1 R[k] ← $F_a(0x2000 + k)$

slika 15 – izdelava tabel S[ ], T[ ] , R[ ]

Rezultat tega modula so torej tri tabele. Tabela T[] je velika 2kb, tabela S[] je velika 1kb, tabela R[] pa je precej manjša in je odvisna od dolžine sporočila (vsak kb čistopisa zahteva 16 bajtov v tabeli R[]). Možna je tudi izvedba z manjšimi tabelami, ki pa bi zelo vplivala na hitrost proizvedene sekvence, zato je ta možnost običajno nesprejemljiva.

2. modul: inicializacija

Inicializacija je funkcija, ki se izvede na vsakem začetku samega algoritma, njen namen pa je napolniti registre s pravilnimi vrednostmi. Vsi parametri razen vrednosti l so 32 bitne vrednosti oz. registri. Parametra n in l sta vhodna, ostali parametri so izhodni. inicializacijska koda je predstavljena na sliki 16:

$function\_inicializiraj(n, l, A, B, C, D, n_1, n_2, n_3, n_4)$

$A \leftarrow n \oplus R[4l];\ B \leftarrow (n >> 8) \oplus R[4l+1];\ C \leftarrow (n >> 16) \oplus R[4l+2]$

$D \leftarrow (n >> 24) \oplus R[4l+3];$

for j od 1 do 2

$P \leftarrow A \,\&\, 0x7fc;\ B \leftarrow B + T[P/4];\ A \leftarrow (A >> 9);$

$P \leftarrow B \,\&\, 0x7fc;\ C \leftarrow C + T[P/4];\ B \leftarrow (B >> 9);$

$P \leftarrow C \,\&\, 0x7fc;\ D \leftarrow D + T[P/4];\ C \leftarrow (C >> 9);$

$P \leftarrow D \,\&\, 0x7fc;\ A \leftarrow A + T[P/4];\ D \leftarrow (D >> 9);$

$(n_1, n_2, n_3, n_4) \leftarrow (D, B, A, C);$

$P \leftarrow A \,\&\, 0x7fc;\ B \leftarrow B + T[P/4];\ A \leftarrow (A >> 9);$

$P \leftarrow B \,\&\, 0x7fc;\ C \leftarrow C + T[P/4];\ B \leftarrow (B >> 9);$

$P \leftarrow C \,\&\, 0x7fc;\ D \leftarrow D + T[P/4];\ C \leftarrow (C >> 9);$

$P \leftarrow D \,\&\, 0x7fc;\ A \leftarrow A + T[P/4];\ D \leftarrow (D >> 9);$

slika 16 – algoritem 2.modula

Sedaj imamo naložene registre A-D in shranjene vrednosti $n_1 - n_4$.

3. modul: glavna zanka

V glavni zanki se prej izračunani parametri (tabele in vrednosti registrov) uporabijo za izdelavo izhodnega psevdonaključnega niza. Po vsaki iteraciji je niz daljši za 128 bitov, dokler ne doseže potrebne dolžine psevdonaključnega niza L . Tretji modul, ki je predstavljen na sliki 17, vsebuje dve zanki: najpomembnejša je notranja zanka, ki definira eno iteracijo, saj se v njej skriva tista nelinearna operacija, ki vpliva na kriptografsko vrednost algoritma. Prvi dve verziji algoritma SEAL sta bili identični tretji v vseh vrsticah, razen v predzadnjih dveh, ki sta se spremenili z namenom povečati odpornost na določene napade. Maksimalna dolžina niza za algoritem zapisan zgoraj je je $L_{MAX} = 2^{19}$ bitov. Če pa bi izhodne nize sestavljali za vse vrednosti sekvenčne številke n, bi bila maksimalna vrednost psevdonaključnega niza do 2^{51} bitov.

Varnost predstavljenega algoritma SEAL omogočajo naslednji gradniki: 32-bitno seštevanje (v Boolovi algebri je to nelinearna operacija), vnos komponent povezanih s sekvenčno številko(n_1, n_2, n_3, n_4) v glavno zanko in seveda SHA-1 funkcija, ki razširi ključ v veliko 'skoraj' naključno stanje tabele [6].

Največja odlika tega algoritma je relativna hitrost, saj pri zelo dolgih sekvencah potrebuje za en bajt izhodne sekvence samo 5 ciklov 32-bitnega mikroprocesorja, kar je v primeru z DES desetkrat manj. Algoritem je tudi možno enostavno prilagoditi na 64-bitne procesorje. Patent za SEAL ima IBM.

$function\_glavna(a,n,L)$
Y={ }; % prazen niz
for l = 0 to ∞
$inicializiraj(n,l,A,B,C,D,\ n_1,n_2,n_3,n_4)$
for i =1 to 64
$P \leftarrow A \,\&\, 0x7fc;\ \ B \leftarrow B + T[P/4];\ A \leftarrow (A >> 9);\ B \leftarrow A + B;$
$Q \leftarrow B \,\&\, 0x7fc;\ \ C \leftarrow C \oplus T[Q/4];\ B \leftarrow (B >> 9);\ C \leftarrow C + B;$
$P \leftarrow (P + C) \,\&\, 0x7fc;\ \ D \leftarrow D + T[P/4];\ C \leftarrow (C >> 9);\ D \leftarrow D + C;$
$Q \leftarrow (Q + D) \,\&\, 0x7fc;\ \ A \leftarrow A \oplus T[Q/4];\ D \leftarrow (D >> 9);\ A \leftarrow A + D;$
$P \leftarrow (P + A) \,\&\, 0x7fc;\ \ B \leftarrow B + T[P/4];\ A \leftarrow (A >> 9);$
$Q \leftarrow (Q + B) \,\&\, 0x7fc;\ \ C \leftarrow C \oplus T[Q/4];\ B \leftarrow (B >> 9);$
$P \leftarrow (P + C) \,\&\, 0x7fc;\ \ D \leftarrow D + T[P/4];\ C \leftarrow (C >> 9);$
$Q \leftarrow (Q + D) \,\&\, 0x7fc;\ \ A \leftarrow A \oplus T[Q/4];\ D \leftarrow (D >> 9);$
$y \leftarrow y \,||\, (B + S[4i-4]) \,||\, (C \oplus S[4i-3]) \,||\, (D + S[4i-2]) \,||\, (A \oplus S[4i-1]).$ %izhodni niz
If($|y| \geq L$ then return y and stop %konec zanke'i'
If odd(i) then $(A,B,C,D) \leftarrow (A + n_1; B + n_2; C \oplus n_1; D \oplus n_2)$
else $(A,B,C,D) \leftarrow (A + n_3; B + n_4; C \oplus n_3; D \oplus n_4)$
$l \leftarrow l + 1$

slika 17 – glavna zanka algoritma SEAL 3.0 [6]

Leta 2002 je bila predlagan izboljšan algoritem, ki je še vedno temeljil na ideji SEAL-a. Zahteve novega algoritma so bili naslednji: Hitrost kriptiranja naj bi ostala enaka, algoritem bi imel manjše tabele, proizvajal naj bi večje sekvence (kar avtomatsko izboljša varnost), uporabljal naj bi bistveno večjo sekvenčno številko. Novemu predlaganemu algoritmu je ime SCREAM [10].

PHELIX

Pretočni postopek PHELIX je bil predlagan kot eden izmed kandidatov v projektu eSTREAM leta 2004. Algoritem odlikuje kar nekaj lastnosti, zaradi katerih je bil izbran v drugo fazo projekta, ki ga bom podrobneje opisal na koncu naloge.

Najpomembnejša lastnost algoritma PHELIX je dvojna uporabnost. Do zdaj sem pretočne algoritme opisoval kot orodja, s katerimi smo lahko zagotavljali zasebnost (ang. privacy) vsebine sporočila (čistopisa). Pretočni postopek PHELIX pa poleg varnosti že v svoji notranji izvedbi omogoča tudi storitev verodostojnosti in avtorizacije (ang. authentication) sporočila, ki ga pošilja oddajnik s posebno MAC kodo (ang. MAC tag).

PHELIX deluje na enakem principu kot SEAL, torej na šifriranju sporočila, vendar tokrat ne gre samo za aditivno mešanje čistopisa s psevdonaključnim nizom, ampak je vnos čistopisa tudi del iteracijske zanke algoritma. PHELIX prav tako vsebuje javno začetno vrednost (n) in skrivni ključ (k) s katerima se ustvari močna psevdonaključna sekvenca. Skrivni ključ je tokrat lahko različne dolžine do 256 bitov (v praksi se uporablja predvsem 256 in 128-bitni ključ). Prav tako je povečana fiksna dolžina sekvenčne številke na 128 bitov. Tudi v tem

algoritmu se operacije izvajajo nad 32-bitnimi besedami. Tudi operacije so identične (seštevanje po modulu 2 in 32 ter rotacija registrov).

PHELIX je po svoji zasnovi pravi programski pretočni algoritem, ki vsebuje zelo podobne module, kot sem jih opisal za SEAL, zato bom celotno delovanje tega algoritma pojasnil na podlagi primerjav s SEAL-om. Sam opis bom tokrat namesto programske kode podal z blok shemo, ki je drugi način za opisovanje takih programskih modulov.

1. mešanje ključev

Modul mešanja ključev je zelo podoben generaciji tabel pri SEAL-u. Bistvena sprememba pa je v prihranku prostora, saj niso potrebne velike tabele, ki so bile ena večjih omejitev SEAL-a. Osnovna naloga tega modula je iz skrivnega ključa k (različnih dolžin) s pomočjo nelinearnih operacij ustvariti 256 bitni delovni ključ sestavljen iz v obliki sedmih besed $K_0 .....K_7$.

2. inicializacija

Naloga tega modula je popolnoma enaka modulu opisanem pri SEAL-u. V tem delu se v delovne registre naložijo začetni podatki, ki so funkcija vrednosti (n) in (k). Registre v i-ti iteraciji imenujemo imenujemo $Z_0^i, Z_1^i, Z_2^i, Z_3^i, Z_4^i$.

3. glavna zanka - šifriranje

Šifriranje po algoritmu PHELIX je predstavljeno na sliki 18, ki prikazuje eno iteracijsko zanko tega algoritma. V eni zanki algoritma se proizvede ena beseda (torej 32 bitov) šifropisa C_i. Zanka ima več komponent in sicer aktivne komponente $Z_0^i, Z_1^i, Z_2^i, Z_3^i, Z_4^i$, katerih vrednost spreminjamo v enem iteracijskem koraku, ter pasivne komponente $X_{i,1}, X_{i,0}, P_i$ in Z_4^{i-4}, ki jih dobimo z različnimi operacijami nad preteklimi vrednostmi oz. delovnim ključem in sekvenčno številko. Na ta način se lahko tvori sporočilo z maksimalno dolžino L=2^{64} bajtov.

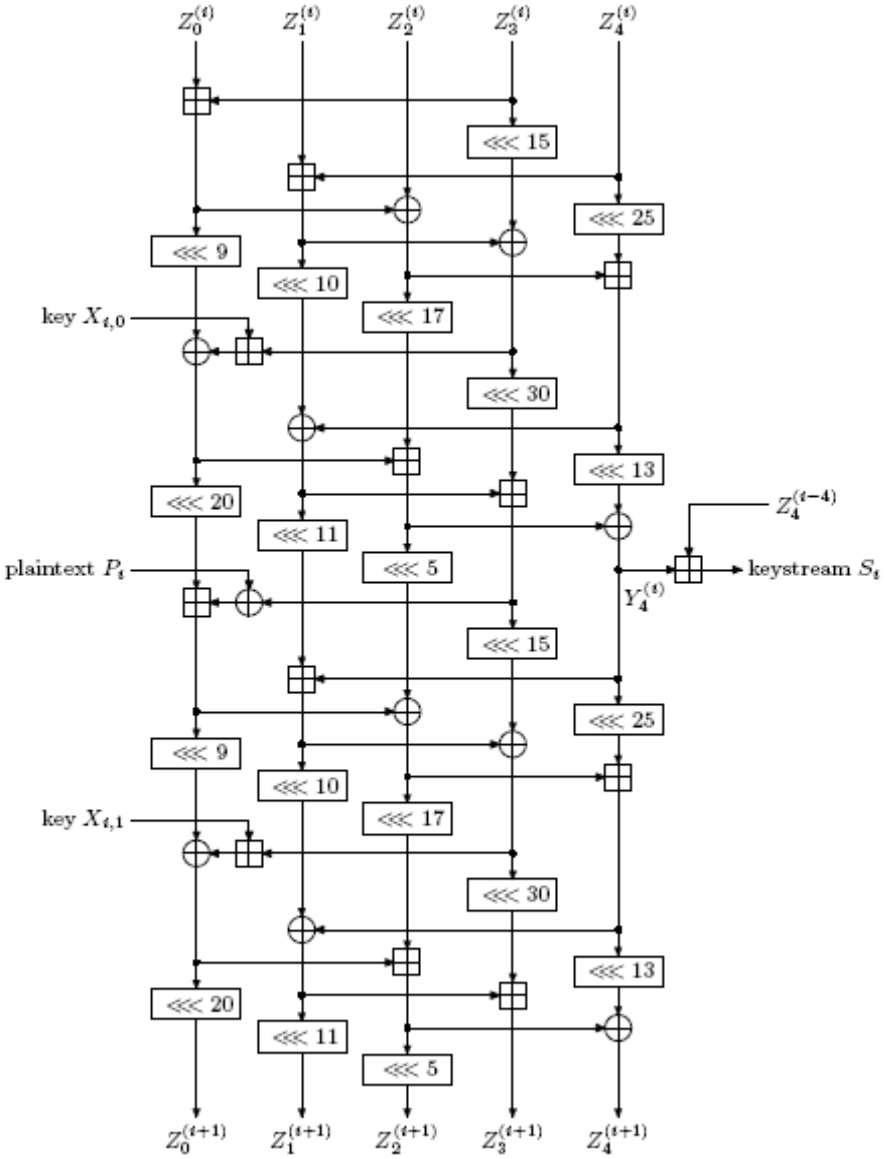
4. dodajanje MAC kode

Posebnost algoritma PHELIX je dodajanje MAC kode šifropisu. Ta se oblikuje v čisto običajen šifropis, ki je posledica preteklih stanj v šifrirniku.

Ko se šifrira zadnja beseda čistopisa se stanju Z_0^m z XOR funkcijo doda vrednosti 0x912d94f1 in vpiše na isto mesto. Namesto čistopisa se na vhod postavi beseda $P_i = l(P) \bmod 4$; kjer je l enaka številu bajtov v celotnem sporočilu. Na ta način se izvede naslednjih 8 iteracij, na izhodu pa se besede zavrže. Nato se iteracija izvede še štirikrat in izhodne besede tvorijo 128-bitno MAC kodo, ki se preprosto doda šifropisu [12].

Za samo izvedbo dobrega pretočnega šifrirnika je potrebno še nekaj pogojev, ki pa veljajo za vse pretočne algoritme:

- Vsak par (k,n) se lahko za šifriranje sporočil uporabi samo enkrat, saj s tem simulira Vernanov šifrirnik. Če se isti par uporablja za šifriranje več sporočil, je varnost pretočnega algoritma resno ogrožena, vendar je verjetnost, da bi se odkrilo skrivni ključ še vedno zelo majhna. Če bi hotelo komunicirati več ljudi z istim skrivnim ključem, potem si morajo razdeliti prostor zalog vrednosti sekvenčnih številk.
- Sprejemnik ne odda čistopisa vse dokler se ne izračuna tudi MAC koda, kar pa zahteva veliko spomina za celotno sporočilo [12].



slika 18 – iteracijska zanka algoritma PHELIX [12]

PHELIX je na dosedanjih testih dosegel dobre rezultate, saj pri dovolj dolgih sporočilih omogoča šifriranje pri 7 ciklih za proizveden bajt, kar ga uvršča zelo visoko, še posebej če upoštevamo dvojno funkcionalnost algoritma (varnost, MAC koda). Algoritem AES recimo zahteva minimalno 19 ciklov brez MAC funkcionalnosti realno pa še več.

PHELIX je sodoben šifrirnik, ki obeta zelo veliko, vendar je kriptografski znanosti povsem nov in njegova vrednost bo morala biti šele dokazana. Najlepša priložnost za to pa je konec projekta eSTREAM, kjer se je do zdaj zelo dobro obnesel

RC4

Naslednji pretočni postopek je v bitvu družina pretočnih postopkov imenovana RC4. Njihova največja prednost sta hitrost in izredna enostavnost algoritma, kar pa se je v zadnjih letih izkazalo tudi za slabost.

Za razliko od večine doslej opisanih algoritmov je RC4 zelo dobro uveljavljen pretočni algoritem, ki se uporablja v veliko standardnih aplikacijah. Če naštejem samo nekatere: TLS protokol za internetno varnost, Oracle secure SQL, aplikacije MS Windows, Lotus Notes in mnoge druge npr aplikacije v brezžičnih komunikacijah, bančne aplikacije, itd.

RC4 je predstavil Ron Rivest (eden izmed ustanoviteljev firme RSA) leta 1987. Vse do leta 1994 je bil algoritem označen kot nerešljiva komercialna skrivnost (ang. trade secret), ko je nekdo na poštno listo 'cypherpunks' poslal osnovo algoritma, za katerega se je izkazalo, da ustreza vsem lastnostim RC4. Še danes ni jasno ali je anonimnež dobil algoritem z izdajo RSA skrivnosti ali pa z inverznim inženiringom. Ker RSA ni nikoli potrdil pristnosti algoritma, ima dani pretočni algoritem oznako ARCFOUR, vendar v praksi ustreza vsem lastnostim RSA RC4.

Sam algoritem je po zasnovi podoben drugim programskim pretočnim algoritmom. To je še vedno binarni aditivni pretočni algoritem, ki uporablja skrivni ključ različnih dolžin (najpogostejše so od 40 do 256-bitne, obstaja pa tudi 2048-bitna verzija). Za svojo transformacijsko funkcijo uporablja veliko tabelo S[ ], ki se s pomočjo skrivnega ključa počasi spreminja in z nelinearnimi operacijami nad elementi tabele S[ ] tvori psevdonaključen niz, ta pa se nato na klasičen način (s operacijo XOR) uporabi za šifriranje čistopisa. Operacije znotraj RC4 se lahko izvajajo nad različno dolgimi besedami. Običajno je za to uporabljen kar bajt, torej $n = 8$. Sam algoritem temelji na dveh modulih: Mešanje tabele S[ ] s tabelo ključev in psevdonaključni generator, ki ju bom podrobneje opisal.

1. Mešanje tabele S[ ] s tabelo K[ ] – KSA

KSA (ang. Key Scheduling Algorithm) je algoritem podoben inicializaciji pri že opisanih algoritmih in ustvari začetne pogoje torej mešano tabelo S[].

V začetku je potrebno ustvariti tabelo $S[\ ]$, ki ima obliko sklada z 2^n vrednostmi dolžine n. Vhodna tabela je tabela $K[\ ]$ enake velikosti, ki jo dobimo z vlaganjem l-bajtnega ključa v tabelo $K[\ ]$. Da je tabela dovolj velika je potrebno njeno vsebino ponavljati (npr. 40 bitni oz.

5 bajtni ključ se bo v 256 bajtni tabeli ponovil 51-krat). Sam algoritem bo imel naslednjo obliko (slika 19) in je narejen za $n = 8$ oz. $2^n = 256$:

```
function _ KSA(K)
for i = 0 to 255                    % začetna tabela S[]
 S[i] ← i ;  j ← 0
for i = 0 to 255
 j ← ( j + S[i] + K[i mod l]) mod256;
 S[i] ↔ S[ j] ;                     % zamenjava vrednosti
```

slika 19 – mešanje tabele S[ ]

2. Izračun psevdonaključne sekvence

Sedaj ko imamo definirano tabelo S za vsak i, lahko nad njo izvajamo operacije in definiramo izhodni psevdonaključni niz. Celotna operacija se imenuje izračun psevdonaključne sekvence (ang. Psevdo Random Generating Algorithm - PRGA). Po vsaki iteraciji lahko na izhod damo po n bitov, v našem primeru je to število torej en bajt. Slika 20 nam prikazuje ta izračun:

Vsaka vrednost v S[] je zamenjana v 256 korakih te iteracijske zanke, kar nam omogoča stalno in kontrolirano spreminjanje transformacijske funkcije.

```
function _ PRGA(S)
 i ← 0;  j ← 0 ;
 For i 0 to ∞
 i ← (i +1) mod256;
 j ← ( j + S[i]) mod256;
 S[i] ↔ S[ j] ;
 y = S[S[i] + S[ j]] mod 256;                    % definicija
```

slika 20 – izračun izhodnega niza

Opisani modul nam že vrne psevdonaključni niz, ki ga nato uporabimo v RC4 šifriranju.

RC4 algoritem je bil, v času ko je bil razvit, izredno hiter, varen in učinkovit. Razen varnosti se mu te lastnosti prizna še danes. Kljub številnim poskusom, da bi ga odkrili, pa je do prvega pravega napada prišlo šele po razkritju teh bistvenih modulov.

Glede na svojo razširjenost je bil v preteklih letih RC4 deležen velike pozornosti in analiz. Kot največja slabost se je zaradi svoje enostavnosti izkazal KSA algoritem, ki je v določenih stanjih bil tako šibak, da se je le-to dalo uganiti že z nekaj biti skrivnega ključa [12].

Kot še večja slabost se je izkazalo prvih nekaj izhodnih bajtov algoritma, kjer je prišlo do velike korelacije s skrivnim ključem [13] saj tabela S[] še ni dovolj premešana. Zato sodobna literatura zahteva izboljšanje algoritma s tem, da se nekaj začetnih bajtov ne uporabi za šifriranje, ampak se jih zavrže že znotraj algoritma. [14] navaja potrebno število 512 bajtov zavrženih bajtov, novo izdani RFC4345 [15] pa navaja to številko precej večjo in sicer 1536 zavrženih začetnih bajtov psevdonaključne sekvence. S tem se varnost RC4 bistveno izboljša. Opisana težava nastane, ker so bili prvi biti šifropisa preveč korelirani z notranjim stanjem tabele S[ ]. Opisano težavo in rešitev lahko srečamo v vseh programskih pretočnih algoritmih.

Tudi RC4 lahko deluje po pricipu sekvenčne številke (n) (oziroma inicializacijskega vektorja IV), kjer se s sporočili prenaša tudi del skrivnega ključa. Raziskave so pokazale, da če se ta del ključa pridoda skrivnemu delu na kakšen enostaven način (spajanje ali pa XOR operacija) lahko to vodi k bistveno oslabljeni varnosti celotnega algoritma. Omenjena slabost je tudi razkrila brezžični protokol WEP, ki uporablja RC4 [13].

Kljub tem slabostim je RC4 trenutno še vedno najpogosteje uporabljen pretočni algoritem, saj napadi nanj zahtevajo razmeroma veliko procesnega dela in časa, kljub temu pa ga proizvajalci ne vgrajujejo več v novejšo varnostno infrastrukturo, ki naj bi bile varna tudi v prihodnosti.

Zaradi dobrih lastnosti in razvoja računalnikov so se pojavile tudi ideje o 32-bitnih verzijah RC4 (torej bi bile vse operacije na 32-bitnih besedah), ki naj bi bile še hitrejše in predvsem bolj varne[21], vendar do sedaj se te ideje niso uveljavile kot standard.

Ker je pretočni algoritem RC4 tako razširjen, bom v nadaljevanju opisal uporabo le-tega na različnih aplikacijah.

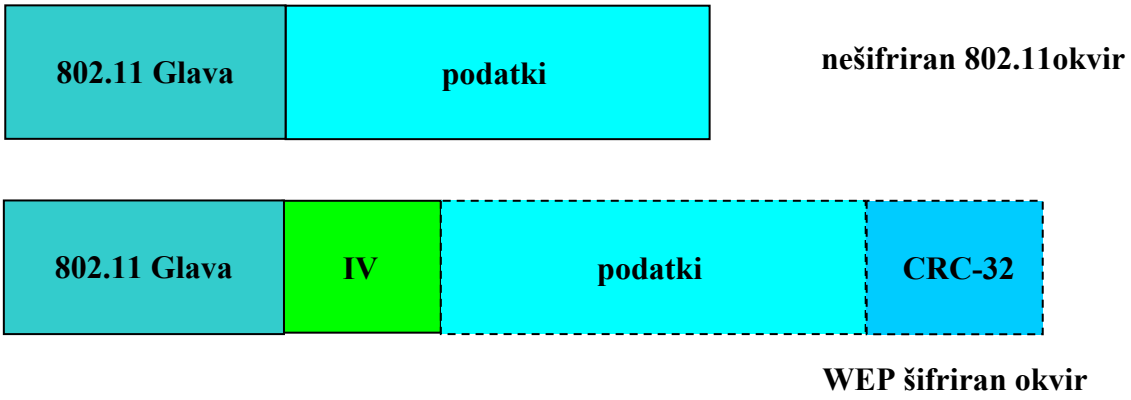
PRIMERI UPORABE RC4

WEP

WEP (Wired Equipment Privacy) je shema za zavarovanje brezžičnih omrežij (Wi-Fi) med uporabniki in dostopovnimi točkami brezžičnih lokalnih omrežij. Njen namen je bil ustvariti pogoje zasebnosti, ki jih omogočajo ožičena omrežja. WEP je bil definiran v standardu 802.11 leta 1999. Za zasebnost sporočil bi po tej shemi poskrbel algoritem RC4, za verodostojnost sporočila pa CRC-32.

Obstajajo tri verzije WEP (40-bitna, 128-bitna in 256-bitna verzija), ki upotrabljajo ključe različnih dolžin. V 802.11 je definirana samo 40-bitna verzija. V vseh verzijah različnih dolžin je ključ sestavljen iz 24-bitnega inicializacijskega vektorja (IV ali nonce) in preostalega skrivnega dela ključa.

Na sliki 21 imamo predstavljeno ovijanje običajnega WLAN okvira v varnejši WEP okvir, ki se nato oddaja med gradniki WLAN omrežij [16].



slika 21 – WEP ovijanje 802.11 okvira

Normalnemu podatkovnemu delu okvira se loči glava in doda CRC-32 koda, nato pa se vse skupaj šifrira po RC4 algoritmu. Na koncu se okviru doda še 24 bitno IV in originalno glavo ter se ga pošlje na radijski vmesnik.

Algoritem je bil kljub izboljšanim verzijam zelo kmalu proglašen za šibkega iz več razlogov:

- Prvi je bil nedefinirana uporaba skrivnih ključev in IV dela kode, ki se bi lahko zelo hitro ponavljala, kar je v kriptografiji nedopustno (enak par: skrivni ključ – IV). Tudi skrivne ključe se redko menja in tas menjava v protokolu sploh ni definirana.
- Premajhen prostor za IV del (24 bitov). Na dovolj obremenjeni mreži se lahko isti IV uporabi že po uri.
- enostavno spajanje ključa in IV, ki vodi v več možnih napadov.
- Naslednja slabost je nešifriran IV.
- Nekatere šibke IV vrednosti (iz celotnega 24-bitnega nabora jih je kar 9000), ki zelo hitro vodijo do notranjih RC4 stanj. [17]

Vse te slabosti brezžičnega standarda so narekovale imprejšnjo zamenjavo varnostnega mehanizma WEP, ki ga je nadomestil WPA. Treba je poudariti, da se slabosti WEP niso pojavile znotraj algoritma RC4, pač pa predvsem v slabi uporabi le-tega.

WPA

WPA (Wi-Fi Protected Access) so razvili izdelovalci brezžične opreme zaradi predolgega razvijanja standarda 802.11i. Njegov namen je bil čimprejšnja zamenjava varnostnega mehanizma WEP in hkrati obdržati kompatibilnost z njim.
WPA ima naslednje lastnosti, ki bistveno izboljšajo uporabo algoritma RC4.

- Mehanizem TKIP (Temporary Key Integrity Protocol), ki dinamično menja ključe posameznim uporabnikom in tudi znotraj posameznih brezžičnih sej. Mehanizem TKIP poskrbi tudi za hierarhijo ključev in mehanizem upravljanja s ključi, kar je bila največja pomanjkljivost WEP-a.
- MIC (Message Integrity Code znan tudi kot 'Michael') mehanizem za izboljšavo verodostojnosti sporočil, ki vsebuje tudi štetje okvirov kar zmanjšuje možne napade

- 48-bitni IV, ki preprečuje uporabo istega IV na isti seji (oz. z istim ključem)
- Možnost uporabe avtentikacije delovnih postaj s pomočjo aplikacijskih strežnikov RADIUS ali LDAP, ki dodatno zavarujejo vstop nepooblaščenim uporabnikom WLAN-a.[19]

Junija 2004 je IEEE postavila nov standard (802.11i), ki je zajemal specifikacije WPA in jim dodajal nekatere nove lastnosti, med njimi je bila tudi zamenjava pretočnega algoritma RC4 z novejšim bločnim algoritmom AES (shema je dobila ime WPA2).

Tudi WPA pa je imel pomankljivost in sicer pomanjkanje mehanizma za zanikanje storitve (ang. denial of service).

TLS/SSL

Protokol TLS(Transport Layer Security) je bil definiran leta 1999 v RFC2246 in je naslednik internetnih varnostnih protokolov SSL. Kot že ime pove je TLS protokol transportnega nivoja in je namenjen zagotavljanju zasebnosti in verodostojnosti sporočil v TCP/IP protokolnem skladu.

TLS deluje na pricipu klient-strežnik in je sestavljen iz dveh podslojev imenovanih 'record layer ' in 'handshaking layer'. Prvi se ukvarja s šifriranjem blokov podatkov (ang. records), drugi pa z začetnim usklajevanjem kot so avtentikacija klienta in strežnika, specifikacija šifrirnega protokola, poročanju o napakah, itd.

V specifikaciji najdemo več možnih vrst algoritmov glede na vrsto šifriranja:
- Asimetrični algoritmi: RSA, Diffie-Hellman, DSA ali Fortezza
- Simetrični algoritmi: RC2, RC4, IDEA, DES, TripleDES, AES
- Zgoščevalne funkcije: MD5, SHA

V TLS se uporablja navaden RC4 brez inicializacijskega vektorja. Skrivni ključ se ob vsaki seji izračuna znotraj 'record' podsloja na podlagi dogovorjenega 48-bitnega parametra imenovanega 'master secret'. Iz tega parametra se nato prek zgoščevalnih funkcij izračunajo vsi potrebni parametri za izračunavanje MAC kode in skrivni ključi za posamezno sejo za kar so potrebni dodatni psevdonaključni mehanizmi [20].

Ker se v TLS uporablja RC4 brez inicializacijskega vektorja, se lahko uporablja kar navadna RC4 koda, opisana v tem poglavju. V novih internetnih brskalnikih se v TLS običajno uporablja 128-bitni RC4 skrivni ključ. Primer uporabe je npr. storitev Klik NLB.

PROJEKT eSTREAM

Za zadnje poglavje seminarske naloge sem si izbral pregled projekta eSTREAM, kjer bom opisal najnovejše raziskave na področjo pretočnih algoritmov, ki bodo mogoče določile nov standardni pretočni algoritem, vsekakor pa projekt predstavlja zadnje trende na področju razvoja pretočnih postopkov.

Leta 1997 je Ameriški standardizacijski inštitut (NIST) objavil odprti projekt s katerim bi postavili nov šifrirni standard za moderne komunikacije imenovan AES. V projektu so sodelovale tako raziskovalne inštitucije kot industrija, zaključen pa je bil leta 2001. Kot končni zmagovalec je bil med drugimi izbran belgijski bločni algoritem Rijndael, ki je na ta način postal ključen za današnjo kriptografijo. Standard AES pa trenutno najbolj razširjen novejši standard.

Po ameriškem zgledu se je tudi v Evropi razvil podoben projekt imenovan NESSIE, ki je prav tako raziskoval in predvsem testiral nove šifrirne primitive. Projekt je potekal v letih 2000-2003 in kot končni rezultat podal več zmagovalcev v posameznih kategorijah, vendar je tudi tu AES ostal med zmagovalci.

Na področju pretočnih algoritmov je bil rezultat projekta zelo slab, saj niti eden izmed šestih predlaganih pretočnih algoritmov ni prestal vseh varnostnih testov [23]. Posledica tega neuspeha na področju pretočnih algoritmov pa je bila ustanovitev novega projekta imenovanega eSTREAM v okviru širšega projekta ECRYPT podrtega s strani EU.

Namen projekta eSTREAM je torej raziskava na področju pretočnih algoritmov, ki bo dala tudi nek okvir za morebiten nov standardni pretočni algoritem, ki pa znotraj projekta to še ne bo postal dokončni standard. Projekt se je začel z začetnim razpisom za pretočne algoritme leta novembra 2004 in se bo končal predvidoma januarja 2008.

Zahtevane lastnosti predlaganih algoritmov

Predlagani algoritmi so razvrščeni v dve kategoriji in sicer programski (PROFILE I) in strojni pretočni algoritmi(PROFILE II).

Zahteve za programske algoritme so predvsem velika pretočnost, vsaj 128-bitni ključ in 128-bitna začetna vrednost IV. Algoritme se ocenjuje v primerjavi z AES in drugimi predlaganimi algoritmi. Na algoritme se gleda tako s stališča izvedbe z enostavnejšimi 8-bitnimi mikroprocesorji kot s sodobnimi 32 in 64-bitnimi procesorji.

Strojni algoritmi morajo imeti vsaj 80-bitni ključ in 32-bitni IV. Namenjeni naj bi bili varnostnim karticam in enostavnejši varni strojni opremi. Bistvo pri strojnih algoritmih je najti shemo, ki bo zahtevala bistveno enostavnejšo strojno izvedbo kot je trenutna AES izvedba.

Tako prva kot druga kategorija algoritmov vsebujeta še posebno podkategorijo, kjer so skupaj zbrani algoritmi, ki že v svoji notranji strukturi vsebujejo tudi MAC funkcionalnost, kar je dodatna prednost za nadaljno obravnavo algoritma, saj s tem omogočimo dve storitvi naenkrat in prihranimo dodatno procesiranje za verifikacijo.

Algoritmi so lahko sinhroni ali asinhroni. Zahteva za vse algoritme je tudi boljša lastnost od standardiziranega AES vsaj v enem kriteriju, seveda pa je cilj najti algoritem, ki bo znatno boljši v vseh ključnih kriterijih.

Potek izbora

Konec leta leta 2004 se je objavil razpis za algoritme, ki so jih je nato zbirali do aprila 2005. Nato so začeli prvo fazo projekta v kateri so podrobno analizirali vse algoritme in določili tiste, ki so se najbolj izkazali. Prva faza se je končala februarja 2006. Izbrane algoritme, ki so se uvrstili v drugo fazo bodo podrobneje raziskali do septembra 2007, januarja 2008 pa bo objavljeno končno poročilo. Do predvidenega roka je bilo predlaganih 34 algoritmov, nekateri so bili predlagani v obeh kategorijah.

Kriteriji

Programske algoritme ocenjujejo na podlagi varnosti, enostavnosti, fleksibilnosti, zahtev trga in seveda zmogljivosti, kjer pa so kriteriji za programske in strojne pretočne algoritme različni.

Zmogljivost programskih algoritmov so opredeljevali z naslednjimi parametri:

- hitrost šifriranja bitnega pretoka
- hitrost šifriranja paketov (dolžine paketov so prilagojene IP prometu, zato meritev zajema tri vrste testov s paketi različnih dolžin in sicer 40, 576 in 1500 bajtov).
- agilnost oz. sposobnost aplikacije za hitro paralelno procesiranje večih sej – merili so torej hitrost šifriranja pod veliko sejami in stalno preklapljanje med njimi. Lastnost je seveda povezana s potrebo po pomnilniku za posamezen pretočni algoritem. Agilnost algoritma je pomembna na primer za strežniske aplikacije, ko morajo le-te podpirati veliko število TLS sej.
- Zakasnitev zaradi začetne nastavitve ključev in IV vrednosti. Ta zadnja meritev je še najmanj kritična, saj je učinkovitost nastavljanja ključev in IV vrednosti razvidna že iz hitrosti šifriranja paketov. Če algoritem zajema tudi izračun MAC vrednosti, potem je v zakasnitvi zajeta tudi ta funkcija.

Zmogljivost strojnih algoritmov so v testu [26] opredeljevali z naslednjimi parametri:

- Površina vezja
- Maksimalna hitrost urinega takta (odvisna predvsem od izvedbe algoritma)
- Možnost paralelne izvedbe algoritma (kot TRIVIUM) kar opišemo s parametrom 'radix', kjer se torej znotraj enega cikla lahko procesira več izhodnih bitov.
- Hitrost šifriranja, ki je prav gotovo najpomembnejši podatek in ga pri strojnih pretočnih algoritmih podajamo kar v bitih oz. gigabitih na sekundo.
- Relativna hitrost šifriranja glede na potrebno površino, dobimo z deljenjem prej omenjenih parametrov

Vsi strojni algoritmi so bili v tem delu izvedeni z vezji ASIC, ki tako kot FPGA omogočajo veliko gostoto vrat potrebnih za izdelavo strojnega pretočnega algoritma.

Rezultati

Na podlagi zgoraj omenjenih kriterijev za zmogljivost programskih algoritmov so bila izvedena testiranja [25]. Podani testi so bili narejeni na standardnem PC-ju (Athlon 64-3GHz) konec lanskega leta. Rezultati najbolj zanimivih algoritmov so podani v tabeli 3. Večino algoritmov sem opisal v seminarski nalogi.

Iz tabele 3 je razvidno, kakšne lastnosti imajo predlagani pretočni algoritmi glede na standardizirana in dobro uveljavljena koncepta AES in RC4.

algoritem	Key	IV	MAC	pod. tok	p 40	p 576	p 1500	p povp	več sej	zakas ključ	zakas IV
enote				ciklov / bajt						ciklov / sejo	
AES-CTR	256	128		25	31	25	25	26	26	263	12
RC4	256	0		13	334	35	21	50	19	105	/
PHELIX	256	128	128	6	31	10	9	11	7	275	726
Salsa20	256	64		8	42	9	9	11	9	28	12
TRIVIUM	80	80		4	23	5	5	6	5	43	697

Tabela 3 – testiranje programskih pretočnih algoritmov [25]

Iz tabele je tudi razvidno, kako velika je lahko razlika med hitrostjo šifriranja glede na dolžino IP paketov in prav tu je opazna vidna izboljšava predlaganih pretočnih algoritmov, glede na bločni AES.

V tabeli sodeluje tudi algoritem TRIVIUM, ki se izkazuje z veliko pretočnostjo, vendar je treba poudariti, da sam algoritem ni preveč primerljiv, saj ne vsebuje 256-bitne varnosti in je običajno izveden v strojni obliki.

Najboljše rezultate sta na testiranjih dosegla PHELIX in Salsa20, vendar je spet treba opozoriti na dodatno MAC verifikacijo sporočila, ki jo vsebuje samo PHELIX. Tudi testiranja 128-bitnih različic so dala podobne rezultate. Na [25] je podano testiranje za vseh 34 primitivov, ki so sodelovali na projektu eSTREAM.

Med testiranjem strojnih izvedb se je najbolj izkazal algoritem TRIVIUM, ki je bil daleč najhitrejši med vsemi in je omogočal šifriranje s hitrostmi do 18Gbit/s [26]. Tudi pri drugih meritvah se je najbolj izkazal TRIVIUM, ki je do sedaj najboljši kandidat med strojnimi algoritmi

Februarja letos je projekt eSTREAM prišel do konca prve faze, kjer so podali rezultate dosedanjih analiz in raziskav. Upoštevajoč vse zgoraj omenjene kriterije so določili pretočne algoritme, ki bodo šli v drugo fazo projekta, med njimi so tudi vsi zgoraj omenjeni algoritmi.

Na področju programskih algoritmov so v drugo fazo uvrstili 14 algoritmov, kjer so še posebej opozorili na sedem najboljših. Na področju strojnih algoritmov pa so izbrali 4 algoritme, ki gredo v drugo fazo.Največji poudarek druge faze projekta bo nadaljevanje kriptografskih analiz na omenjene algoritme, saj nekateri pretočni postopki še niso bili podvrženi uspešnim napadom. Tu vidim tudi največjo težavo za predlagane algoritme, saj se kriptografska vrednost algoritma preverja zelo dolgo.

ZAKLJUČEK

V seminarski nalogi sem predstavil pretočne šifrirne postopke predvsem z vidika delitve na dve veliki skupini. Ti skupini sta strojni in programski pretočni postopki. Kljub poskusom v preteklih letih bo pretočne šifrirne postopke težko dokončno zamenjati z bločnimi, še posebej na področjih hitrega šifriranja bitnih sekvenc in šifriranja z majhnimi računalniškimi viri.

Poglavje o psevdonaključnih generatorjih sem napisal iz dveh razlogov. Prvi razlog je bil predstaviti teorijo o LFSR vezjih in naključnih generatorjih, drugi pa je prikazati tehnike s katerimi lahko pojasnimo posamezne operacije v posameznih pretočnih postopkih, ki sem jih opisal kasneje. Vse te tehnike (vnos nelinearnosti v povratno vezavo, neenakomerno proženje, vnos spomina oz. stanja, izločanje prvih bitov iz šifrirne sekvence, vnos IV vrednosti, itd.) izhajajo iz te teorije in so uporabljene tudi v najnovejših pretočnih postopkih.

V nalogi sem namerno spregledal področje podrobne analize vseh teh šifrirnih postopkov, saj je za to potrebna poglobljena teoretična razlaga, ki ji nato sledijo konkretni primeri, ki jih je težko primerjati. Kot zanimivost naj napišem, da sem se med branjem srečal z vsaj 13 tipi napadov in v kriptografski znanosti glede vsakega napada potekajo burne razprave na vprašanja ali je to sploh napad na določen primitiv, ali je potrebno delo za napad realno, smo z napadom prišli do notranjega stanja oz. ključa, itd.

V nalogi nisem omenil še enega tipa pretočnih šifrirnkov in sicer bločne šifrirnike v povratnih načinih CFM in OFB. Pri tej tehniki se teoretična meja med pretočnimi in blokovnimi šifrirniki vidno zabriše. Tudi v splošnem gre razvoj tako enih kot drugih v isti smeri. Trenutno najbolj uveljavljen primer je UMTS šifrirni primitiv KASUMI, ki v načinu f8 deluje kot pretočni postopek, seveda pa gre za dokaj potratno izvedbo pretočnega šifrirnika.

Projekt eSTREAM je trenutno največji poskus vrniti pretočne algoritme na področje standardizacije šifriranja v prihodnosti. Trenutni rezultati kažejo na znatne izboljšave v primerjavi z AES standardom, še vedno pa ostane vprašanje, če je faktor 2 ali 3 pa področju zmogljivosti programskih pretočnih algoritmov dovolj dober razlog za vpeljavo novega standarda. Na področju varnosti je zaupanje ključen faktor, ki si ga je zelo težko pridobiti in AES je to uspelo. Najnovejši rezultati v zvezi tem projektom so dosegljivi na [24].

VIRI

[1] http://www.cacr.math.uwaterloo.ca/hac/about/chap6.pdf

[2] Richard E Smith INTERNET CRYPTOGRAPHY Addison, Wesley Longman inc., 1997

[3] http://citeseer.ist.psu.edu/539824.html (clock controlled generator alternating)

[4] http://www.x5.net/faqs/crypto/q86.html - (stream chipher - splono)

[5] www.isg.rhul.ac.uk/alumni/thesis/kanso_a.pdf

[6] http://citeseer.ist.psu.edu/190319.html

[7] http://citeseer.ist.psu.edu/648879.html

[8] www.gemplus.com/smart/rd/publications/pdf/HG97chis.pdf

[10] http://eprint.iacr.org/2002/019.pdf%20(scream)

[11] http://citeseer.nj.nec.com/cache/papers/cs/25729/http:zSzzSzeprint.iacr.orgzSz2002zSz019.pdf/halevi02scream.pdf

[12] http://cr.yp.to/streamciphers/phelix/desc.pdf

[13] citeseer.ist.psu.edu/fluhrer01weaknesses.html

[14] crypto.stanford.edu/~mironov/papers/**rc4**full.pdf

[15] www.ietf.org/rfc/rfc4345.txt

[16] cs.purdue.edu/homes/**...**/courses/Fall05/lectures/355_Fall05_lect13.pdf

[17] http://www.networkworld.com/research/2002/wepprimer.html

[18] http://www.weca.net/OpenSection/pdf/Wi-Fi_Protected_Access_Overview.pdf

[19] http://www.wi-fiplanet.com/tutorials/article.php/2148721

[20] http://www.faqs.org/rfcs/rfc2246.html

[21] citeseer.ist.psu.edu/723483.html

[22] http://www.ecrypt.eu.org/stream/ciphers/trivium/trivium.pdf

[23] https://www.cosic.esat.kuleuven.be/nessie/deliverables/press_release_feb27.pdf

[24] http://www.ecrypt.eu.org/stream/

[25] http://www.ecrypt.eu.org/stream/perf/amd64/

[26] http://www.ecrypt.eu.org/stream/papersdir/2006/015.pdf

[27] http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4/papers/Mantin1.zip