Pregled varnosti pri komunikaciji z GSM

Tadej Zupančič

23.6.2006

Povzetek

V zadnjih letih beležimo hiter razvoj mobilnih komunikacij, kamor spada tudi komunikacija prek GSM terminala. Tako je danes v uporabi že več kot 2.2 milijardi mobilnih telefonov v takšni ali drugačni obliki. Je sistem GSM dovolj varen za zagotavljanje tajnosti storitev? Kot odgovor na to vprašanje sledi podroben opis sistema GSM s stališča varnosti in opis algoritmov v uporabi ter njihovih slabosti.

Vsebina

1 Uvod

Glede na hitro naraščajoče število mobilnih terminalov in dejstvo, da v razvitih državah mobilni terminal uporablja velika večina prebivalcev, narašča tudi potreba po varnem komuniciranju. Nihče si namreč ne želi, da bi njegove pogovore poslušale tretje osebe, tudi če ti niso zaupne narave.

Še večji problem predstavlja kraja identitete. S tem bi se napadalec predstavljal kot drug uporabnik in uporabljal storitve v njegovem imenu in na njegov račun. Sistem GSM uporablja nekaj zaščit za preprečevanje obeh vrst napadov, postavlja pa se vprašanje: ali so učinkovite?

Začnemo z razlago sistema GSM in njegovega delovanja. SIM kartica, ki je pomemben člen pri zagotavljanju varnosti, je opisana v poglavju 3. Poglavje 4 prikazuje postopek prijave v omrežje in razlago algoritma COMP128, katerega koda se nahaja v poglavju 8. V 5. poglavju se seznanimo s kodiranjem storitev in algoritmoma Kasumi in A5, čigar koda je v poglavju 9. Zaključimo še s pregledom nekaterih napadov na sistem v poglavju 6.

2 Sistem GSM

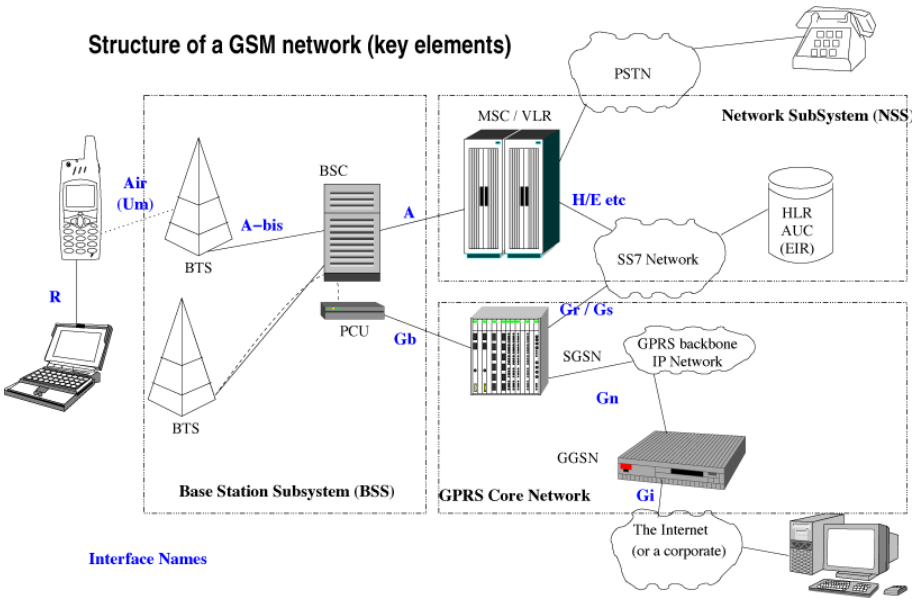
2.1 Ključne komponente sistema GSM

Komponente sistema GSM lahko razdelimo na dva dela. Na eni strani imamo uporabniški del, kamor štejemo mobilne aparate ter opremo, ki se uporablja za povezovanje z omrežjem, na drugi strani pa sistem, ki ga zagotavlja operater mobilnih komunikacij in uporabnikom omogoča storitve.

Ključna stvar za varnost komunikacije je *SIM kartica*, ki skrbi za hrambo zasebnega ključa, izvajanje avtentikacije uporabnika ter enkripcije/dekripcije. Operaterjeva stran vsebuje veliko število naprav, tukaj bomo našteli samo najpomembnejše (slika 1).

Signal pride v omrežje preko baznih postaj *BTS* (Base Transceiver Station), kontroler baznih postaj *BSC* (Base Station Controller) pa dodeljuje frekvence za posamezne postaje, skrbi za prenos komunikacije z ene bazne postaje na drugo ter deluje kot vezni člen med baznimi postajami ter preklopnim centrom. Ob prenosu podatkov v paketih se le ti prenašajo vzporedno prek večih baznih postaj hkrati, zato uporabljamo še *PCU* (Packet Control Unit), katerega glavna naloga je, da uredi pakete po zaporednih številkah.

Preklopni center *MSC* (Mobile Switching Center) deluje kot množica stikal, ki v primeru govornih komunikacij skrbijo za povezovanje med klicateljem in klicanim. Domači register *HLR* (Home Location Register) vsebuje podatke o vseh SIM karticah, ki jih je izdal operater, vključno z njeno identifikacijsko številko *IMSI*, do-



Slika 1: Slika GSM omrežja

deljeno klicno številko, omogočenih storitvah ter trenutno lokacijo uporabnika. K vsakemu preklopnemu centru je pridružen še register obiskovalcev *VLR* (Visitor Location Register), v katerega se ob prijavi v omrežje zapišejo informacije o uporabniku, ki so shranjene v domačem registru njegovega mobilnega operaterja.

Kar je v uporabniškem delu SIM kartica, je na operaterjevi strani avtentikacijski center *AUC* (Authentication Center). Ta skrbi za preverjanje identitete uporabnika ter za generiranje ključa, ki se uporablja za enkripcijo storitev. AUC hrani zasebne ključe K_i vseh SIM kartic, ki jih je izdal mobilni operater.

2.2 Potrebni postopki za komuniciranje

Operater mobilnih komunikacij si prosto izbere algoritma A3/A8 in ju sprogramira v SIM kartice in avtentikacijski center. Ta dva algoritma sta namenjena za avtentikacijo uporabnika in izračun ključa za enkripcijo storitev in naj bi bila, prav tako kot algoritem A5, ki je namenjen za enkripcijo storitev, nepoznana vsem, razen razvijalcem. Čeprav imajo mobilni operaterji proste roke pri izbiri teh dveh algoritmov, večina uporablja kar algoritem COMP128, ki je priložen specifikacijam kot vzorčni algoritem za implementacijo.

Sledi generiranje zasebnih ključev in zapis ene kopije na SIM kartico, druge pa v avtentikacijski center. Zasebna ključa se vedno nahajata samo na teh dveh lokacijah in se nikoli ne prenašata preko kateregakoli dela omrežja. Prav tako se računanje ključev vrši samo v avtentikacijskem centru in na SIM kartici.

Ko se neka oseba odloči, da bo uporabljala storitve določenega mobilnega operaterja,

si izmed ponujenih izbere klicno številko, na katero bo dosegljiva in tako postane uporabnik. Mobilni operater zapiše par (IMSI, klicna številka) v svoj domači register, poleg tega pa še seznam storitev, ki jih bo uporabnik lahko uporabljal.

2.3 Opis delovanja sistema GSM

Recimo, da se uporabnik nahaja v tujem omrežju. Poglejmo si kratek opis prijave in izračuna ključa za enkripcijo storitev.

Ko uporabnik prižge svoj mobilni terminal, ta pošlje IMSI bazni postaji, katera ga posreduje preklopnemu centru. Preklopni center opazi, da uporabnik ni v domačem omrežju, zato zahteva od domačega registra uporabnikovega domačega omrežja podatke o uporabniku, ki jih zapiše v svoj register obiskovalcev. Istočasno od avtentikacijskega centra uporabnikovega domačega omrežja dobi nalogo za avtentikacijo, pravilni odgovor na to nalogo ter ključ K_c, ki se bo uporabljal za enkripcijo storitev.

Avtentikacija uporabnika poteka na način naloga/odgovor. Preklopni center posreduje nalogo preko bazne postaje do uporabnikovega mobilnega terminala. Na SIM kartici se izvrši izračun odgovora na zastavljeno nalogo, ta odgovor pa se pošlje nazaj preko bazne postaje do preklopnega centra. Ta preveri, če se odgovor z mobilnega terminala ujema s pravilnim odgovorom, ki ga je prejel od avtentikacijskega centra in ob pozitivnem izidu potrdi identiteto uporabnika. Po uspešni avtentikaciji pošlje preklopni center ključ K_c kontrolerju baznih postaj, SIM kartica pa ga izračuna iz prejete naloge. Ta ključ se potem uporablja za enkripcijo storitev.

3 Sim kartica

3.1 Arhitektura

Sim kartica (slika 2) je običajna pametna kartica z mikroprocesorjem, ki je sposoben izvajati algoritme enkripcije/dekripcije. Za avtentikacijo in enkripcijo sto-ritev so potrebni trije algoritmi. Dva od njih (A3 in A8) si lahko operater izbere poljubno, saj se bodo izvajali samo na SIM kartici in v njegovem domačem avtentikacijskem centru. Tretji algoritem (A5), ki je namenjen enkripciji storitev, pa je standardiziran in ga v eni izmed različic uporabljajo vsi operaterji, saj je le tako zagotovljeno, da bo komunikacija v vseh omrežjih potekala brez zapletov.

Vsaka SIM kartica ima unikatno identifikacijsko številko IMSI, s katero se predstavlja operaterju in zasebni ključ K_i, ki se uporablja za avtentikacijo in enkripcijo storitev.

Glede na tip SIM kartica vsebuje še različno velik uporabniški prostor, kamor uporabnik lahko shrani imenik, kratka sporočila SMS ter prostor za dodatne storitve, ki jih omogoča mobilni operater.

Slika 2: SIM kartica

3.2 Zaščita

Uporabniški del SIM kartice je zaščiten s kodama PIN/PUK. Ti dve kodi potrebujemo tudi, če hočemo uporabljati katerekoli storitve razen klica na center za obveščanje (112). Ob vklopu mobilni terminal zahteva od nas kodo PIN. Ob trikratnem napačnem vnosu te kode, moramo vpisati kodo PUK. Po nadaljnih treh napakah pri vnosu kode PUK, se kartica zaklene, odklene nam jo lahko le naš mobilni operater. Za vklop nekaterih dodatnih storitev potrebujemo tudi kodi PIN2/PUK2, vendar te storitve niso ravno popularne.

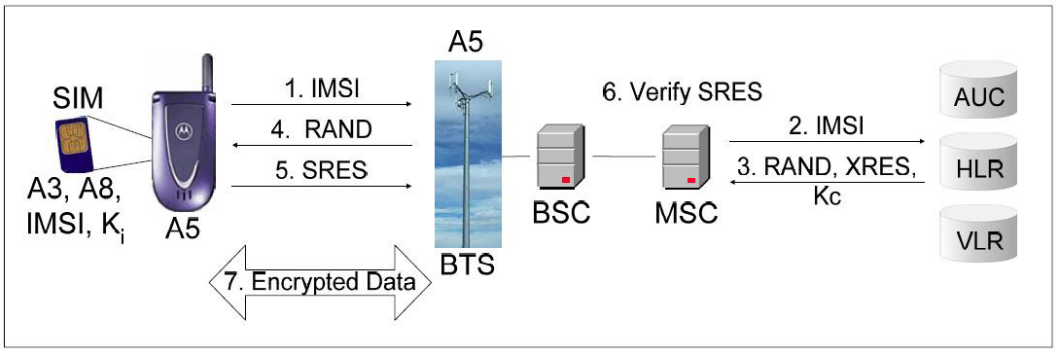
V zadnjem času se v SIM karticah pojavlja tudi zaščita pred prevelikim številom povpraševanj. S tem je kartica zaščitena pred napadalci, ki bi iz velikega števila parov (naloga, odgovor) lahko izračunali zasebni ključ K_i. Novejše SIM kartice se zaklenejo po 2^{16} nalogah.

4 Prijava v omrežje

4.1 Potek prijave

Ob vklopu mobilnega terminala se bazni postaji pošlje identifikacijska številka IMSI. Bazna postaja posreduje IMSI do preklopnega centra. Ta naredi poizvedbo v uporabnikov domači register in od njega pridobi podatke o uporabniku, od avtentikacijskega centra pa dobi trojček (RAND,SRES,K_c). RAND je naključno število, ki ga izbere avtentikacijski center, SRES pa je rezultat algoritma A3 s privatnim ključem K_i in vhodom RAND ($SRES = A3_{K_i}(RAND)$). K_c je izračunan ključ, ki se bo ob uspešni avtentikaciji uporabljal za enkripcijo storitev, avtentikacijski center pa ga izračuna z algoritmom A8 ($K_c = A8_{K_i}(RAND)$).

Mobilni terminal prejme nalogo RAND, SIM kartica izračuna rezultat RES ($RES = A3_{K_i}(RAND)$) in ga pošlje nazaj preko bazne postaje do preklopnega centra. Preklopni center preveri, ali sta SRES in RES enaka in v tem primeru je uporabnik avtenticiran.



Slika 3: Prijava v omrežje

Po uspešni avtentikaciji pošlje preklopni center ključ K_c kontrolerju baznih postaj, ki ga nato uporablja za enkripcijo storitev. SIM kartica na enak način kot avtentikacijski center izračuna K_c ($K_c = A8_{K_i}(RAND)$). Sedaj imata obe komunikacijski strani enak ključ za enkripcijo storitev in komunikacija se lahko začne.

Zaradi zagotavljanja zasebnosti uporabnikov, se tudi identifikacijska številka IMSI prenese preko omrežja samo takrat, ko je to nujno potrebno (npr. ob vklopu mobilnega terminala), kar otežuje sledenje uporabnika in morebitne poskuse prisluškovanja. Po uspešni avtentikaciji dobi mobilni terminal s ključem K_c enkriptirano začasno identifikacijsko številko TMSI (TMSI = f(IMSI)), ki se sedaj uporablja namesto IMSI. Številka TMSI se prenaša tudi med baznimi postajami in kontrolerji baznih postaj, vendar se po določenem času uporabe zamenja za novo. Nova številka TMSI se izračuna iz prejšnje TMSI.

4.2 Algoritem COMP128

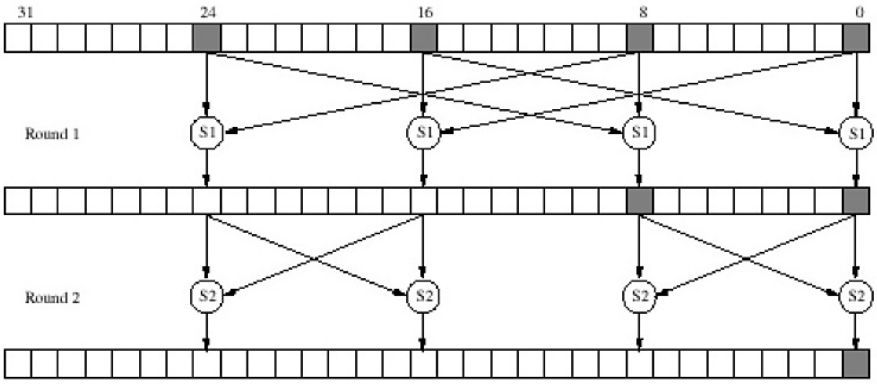
Algoritem COMP128 je bil predlagan kot okvirni algoritem za implementacijo algoritmov A3/A8, vendar običajno operaterji niso zgubljali časa s pisanjem svojih algoritmov, ampak so uporabili kar predlaganega. Algoritem ni bil javno objavljen, poznali so ga samo razvijalci in tisti, ki so dobili specifikacije. Do leta 1997 so ta algoritem dobro skrivali, takrat pa se je po pomoti pojavilo nekaj specifikacij, s pomočjo katerih so lahko približno ugotovili delovanje algoritma. Nekaj podatkov pa vseeno še ni bilo razkritih, zato so manjkajočih 4-6 vrstic kode dobili s pomočjo reverse-engineeringa.

Kot vhod algoritem sprejme 128 bitno naključno število RAND ter 128 bitni zasebni ključ K_i, vrne pa 32 bitni rezultat SRES in 64 bitni ključ K_c, čeprav večina operaterjev uporablja samo 54 bitov, ostalih 10 bitov pa postavi na 0.

Sledi algoritem v psevdokodi:

x[16-31] = RAND
for $0 < j < 8$

x[0-15] = K_i
Kompresija (5 krogov)
Ustvari bite iz bajtov
if $j < 7$ Permutacija
Kompresiraj 16 bajtni rezultat v 12 bajtnega in ga shrani v simoutput[ ]
vrni simoutput[ ]



Slika 4: Metuljčna struktura kompresijske funkcije

Kompresija poteka v obliki metuljčne strukture. Izvaja se na vsakem izmed petih nivojev na dveh enako velikih delih (npr. v krogu 0 2-16 bajtna dela, v krogu 1 4-8 bajtni deli...). Za vsak nivo i imamo tabelo T_i, ki vsebuje 2^{9-i} (8-i) bitnih vrednosti (T_0 vsebuje 512 8 bitnih vrednosti, T_4 pa 32 4 bitnih vrednosti). Na vsakem nivoju uporabimo dva vhodna bajta za izračun indeksa v tabeli. Izhodni bajt je enak vrednosti tabele pri tem indeksu. Po kompresiji imamo v vektorju x[ ] 4 bitne vrednosti. Podrobna koda algoritma je v poglavju 8.

5 Kodiranje storitev

5.1 Splošen opis

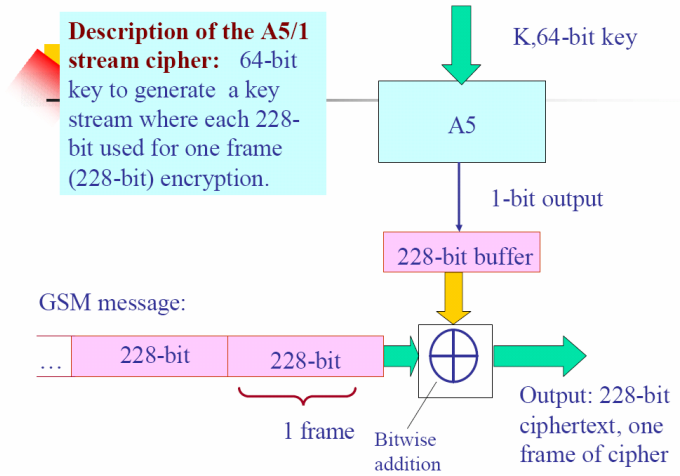
V nasprotju z algoritmi za avtentikacijo, so algoritmi za kodiranje storitev standardizirani. Pri avtentikaciji se računanje ključev vrši samo v avtentikacijskem centru domačega omrežja in na SIM kartici, kar je tudi razlog, da imajo mobilni operaterji proste roke pri izbiri teh algoritmov. Kodiranje storitev pa poteka na SIM kartici in v kontrolerju baznih postaj, zato morajo biti za uspešno komunikacijo v vseh kontrolerjih baznih postaj zapisani isti algoritmi za kodiranje storitev. V primeru, da bi različni mobilni operaterji uporabljali različne algoritme, SIM kartica enega operaterja in bazna postaja drugega operaterja ne bi znali pravilno interpretirati medsebojno prenešenih podatkov.

V GSM telefoniji je za kodiranje storitev standardiziran algoritem A5, ki pa ima več različic. Osnovna različica je imenovana A5/1, kar pomeni, da uporablja običajno

dolžino ključa. V nekaterih državah je zaradi prepovedi uvoza/izvoza kriptografskih algoritmov s (pre)dolgim ključem v uporabi različica A5/2, ki je ekvivalentna različici A5/1, s to razliko, da uporablja krajši ključ. Obstaja pa še različica A5/0, ki ne uporablja enkripcije.

5.2 Algoritem A5/1

Vsak okvir pri GSM komunikaciji vsebuje 228 bitov. Z algoritmom A5 generiramo prav tako 228 bitov dolgo besedo in jo prištejemo okvirju s funkcijo XOR. Tako dobimo okvir, ki je pripravljen za prenos preko omrežja.



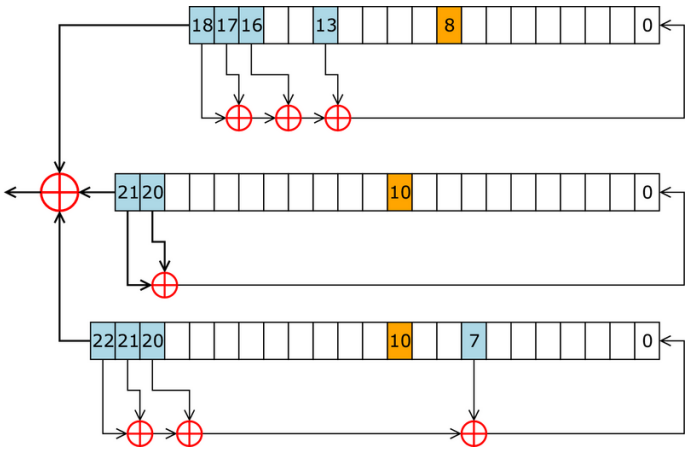
Slika 5: Potek kodiranja

5.2.1 Opis algoritma

Algoritem je sestavljen iz treh vzporedno delujočih pomičnih registrov z linearnim odzivom (LFSR) različnih dolžin (slika 6). Vsak od teh registrov ima določene bite iz katerih se s funkcijo XOR izračuna bit, ki bo prišel na mesto 0 ob pomiku bitov v levo. Izhodni bit je enak funkciji XOR vseh treh bitov, ki ob pomiku izpadejo iz registrov. V vsakem registru je en bit določen kot bit preverjanja (na sliki 6 obarvan oranžno).

5.2.2 Delovanje algoritma

Pred izračunom vsake 228 bitne besede moramo pomične registre inicializirati. Najprej v najmanj pomembni bit vsakega od teh registrov zaporedoma vstavljamo bite 64 bitnega ključa K_i in po vsakem vstavljanju po-ženemo pomik vseh registrov. Enako naredimo še z 22 bitno zaporedno številko okvirja. Po tem poženemo sistem še za 100 krogov, vendar sedaj ne prožimo vseh treh registrov naenkrat, ampak na



Slika 6: Shema pomičnih registrov z linearnim odzivom

sledeč način. Pogledamo vrednosti v bitih preverjanja v vseh treh registrih in izberemo prevladujoči bit (0 ali 1). V vsakem krogu prožimo samo tiste registre, ki imajo v trenutnem krogu bit preverjanja enak prevladujočemu bitu v trenutnem krogu.

Po izvedeni inicializaciji lahko začnemo računati besedo, ki jo potem uporabimo za kodiranje storitev. Omenjeno besedo dobimo tako, da algoritem poženemo še za 228 krogov (pri pomikih zopet gledamo prevladujoči bit) in iz vsakega kroga dobimo 1 bit celotne besede. Koda algoritma se nahaja v poglavju 9.

5.3 Algoritem A5/3 - Kasumi

Algoritem Kasumi je Feistelova šifra z osmimi krogi. Deluje na 64 bitnih podatkovnih blokih z uporabo 128 bitnega ključa. Vhodni podatkovni blok I prvo razdeli na dva 32 bitna bloka L_0 in R_0:

$$I = L_0||R_0$$

nato pa za vsak krog i $1 \leq i \leq 8$ izvaja:

$$R_i = L_{i-1},\ L_i = R_{i-1} \oplus f_i(L_{i-1}, RK_i)$$

Izhod algoritma je enak 64 bitni besedi $(L_8||R_8)$, ki jo sestavimo po koncu osmega kroga (slika 7).

5.3.1 Funkcija f_i

Funkcija $f_i()$ sprejme 32 bitni vhod I skupaj s krožnim ključem RK_i in vrne 32 bitni izhod O. Krožni ključ RK_i je sestavljen iz treh podključev (KL_i,KO_i,KI_i), sama funkcija $f_i()$ pa iz dveh podfunkcij: FL in FO. Glede na parnost zaporedne številke kroga ločimo dve obliki funkcije $f_i()$.

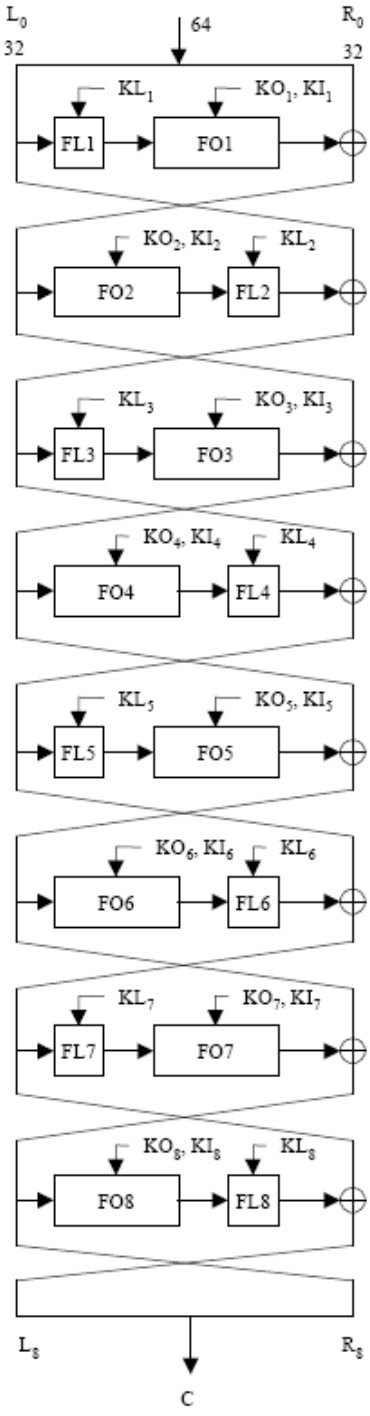


Fig. 1: KASUMI

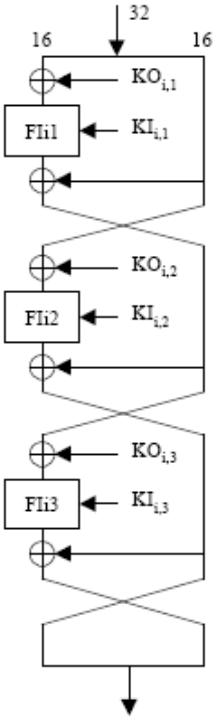


Fig.2: FO Function

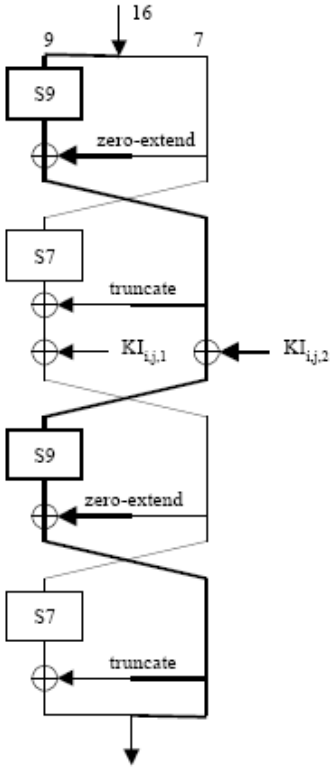


Fig.3: FI Function

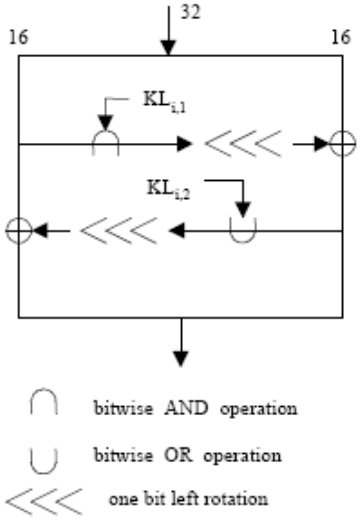


Fig.4: FL Function

Slika 7: Algoritem Kasumi

Za kroge 1,3,5 in 7 velja funkcija:

$$f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i)$$

za kroge 2,4,6 in 8 pa funkcija:

$$f_i(I, K_i) = FL(FO(I, KO_i, KI_i), KL_i)$$

Razlika med sodim in lihim krogom je samo v vrstnem redu izvajanja funkcij FO in FL.

5.3.2 Funkcija FL

Funkcija FL razdeli 32 bitni podključ KL_i na dva 16 bitna podključa, pravtako pa vhod I na dve 16 bitni besedi:

$$KL_i = KL_{i,1}||KL_{i,2},\ I = L||R$$

Izhod funkcije je enak 32 bitni besedi $(L'||R')$, ker sta L' in R' definirana kot:

$$R' = R \oplus ROL(L \cap KL_{i,1})$$

$$L' = L \oplus ROL(R' \cup KL_{i,2}),$$

kjer ROL pomeni krožno rotacijo za en bit v levo.

5.3.3 Funkcija FO

32 bitni vhod I se razdeli na levo in na desno polovico $I = L_0||R_0$, 48 bitna podključa KO_i in KI_i pa vsak na tri 16 bitne podključe:

$$KO_i = KO_{i,1}||KO_{i,2}||KO_{i,3},\ KI_i = KI_{i,1}||KI_{i,2}||KI_{i,3}$$

Za $1 \leq j \leq 3$ definiramo:

$$R_j = FI(L_{j-1} \oplus KO_{i,j}, KIi, j) \oplus R_{j-1}$$

$$L_j = R_{j-1}$$

in kot izhod vrnemo 32 bitno vrednost $(L_3||R_3)$.

5.3.4 Funkcija FI

Funkcija razdeli vhod I na dva dela različne dolžine: 9 bitni levi del L_0 in 7 bitni desni del R_0. Pravtako se ključ $KI_{i,j}$ razdeli na 7 bitni $KI_{i,j,1}$ in 9 bitni $KI_{i,j,2}$ in velja $KI_{i,j} = KI_{i,j,1}||KI_{i,j,2}$.

Funkcija uporablja dve S-škatli, S7, ki preslika 7 bitni vhod v 7 bitni izhod in S9, ki preslika 9 bitni vhod v 9 bitni izhod. Uporabljata se tudi funkciji ZE() in TR(). ZE(x) 7 bitni vhodni besedi na najbolj pomembni mesti doda dva ničelna bita, TR() pa 9 bitni besedi odvzame 2 najbolj pomembna bita in jo tako spremeni v 7 bitno besedo. Da pridemo do rezultata(16 bitne besede $(L_4||R_4)$) moramo izvesti naslednje operacije:

$$\begin{array}{ll} L_1 = R_0 & R_1 = S9[L_0] \oplus ZE(R_0) \\ L_2 = R_1 \oplus KI_{i,j,2} & R_2 = S7[L_1] \oplus TR(R_1) \oplus KI_{i,j,1} \\ L_3 = R_2 & R_3 = S9[L_2] \oplus ZE(R_2) \\ L_4 = S7[L_3] \oplus TR(R_3) & R_4 = R_3 \end{array}$$

Vsebina S-škatel, računanje ključev, izvorna koda in ostali podatki o algoritmu so opisani v [3].

6 Možni napadi

6.1 Predstavljanje kot bazna postaja

Vsak, ki ima dovolj denarja za nakup opreme, lahko postavi svojo "bazno postajo"in preko nje pošilja naloge določenemu mobilnemu terminalu. Mobilni terminal se običajno poveže na bazno postajo z najmočnejšim signalom, kar pomeni, da moramo biti dovolj blizu željenemu uporabniku, saj bi v nasprotnem primeru motili ostale uporabnike. Napad se lahko uspešno izvrši v območjih brez signala (tuneli, podzemna železnica). Po zadostnem številu prejetih odgovorov na zastavljene naloge, lahko izračunamo uporabnikov zasebni ključ.

Obstaja pa tudi možnost, da smo s svojo lažno bazno postajo samo posrednik do prave bazne postaje. V tem primeru samo posredujemo številko IMSI in odgovore od uporabnika do prave bazne postaje, od katere dobimo nalogo za SIM kartico. V tem primeru nam tudi prisluškovanje pogovorom ne predstavlja problema, saj kot bazna postaja lahko naročimo mobilnemu terminalu, naj izklopi enkripcijo, vsa komunikacija pa poteka preko naše lažne bazne postaje.

Pri tem napadu je možnih še nekaj različic. Lahko samo sledimo, katere številke kliče napadeni ali pa celo preusmerimo klice na druge številke.

6.2 Kraja klica

Ta napad je možen samo v primeru, da mobilni operater uporablja algoritem A5/0, to je algoritem brez enkripcije. Napadalec lahko izvede napad tako, da svojemu kolegu naroči, naj pokliče na napadeno mobilno številko. Napad je namenjen

uporabljanju govornih storitev na račun napadenega, če se napadeni nahaja v tujini. Mobilni operaterji zaračunavajo večino stroškov dohodnih klicev v tuje omrežje klicanemu. Zato lahko, ko uporabnik napadene številke prevzame klic, napadalec "ukrade"klic in se "brezplačno"pogovarja s svojim kolegom, napadeni pa glede tega ne more storiti nič. V primeru, da se za kodiranje storitev uporablja enkripcija, tak napad ni možen, razen če poznamo ključ K_c.

6.3 Pridobitev trojčka (RAND,SRES,K_c)

V primeru, da napadalec pridobi en trojček, se lahko dokler je trojček v veljavi predstavlja kot napadeni mobilni terminal in opravlja storitve na račun napadenega. Seveda to ne predstavlja dolgotrajnejše težave, ker se trojček slej ko prej zamenja. Tak trojček je napadalcu lahko bolj koristen za napad z lažno bazno postajo. V tem primeru se lahko z enim trojčkom vedno predstavlja mobilnemu terminalu kot prava bazna postaja in tudi enkriptira storitve s ključem K_c, ki je del trojčka.

6.4 Prisluškovanje na nekodiranem delu povezave

V GSM komunikaciji je enkriptiran samo del povezave - med mobilnim terminalom in kontrolerjem baznih postaj. Prisluškovanje je torej mogoče na poti med kontrolerjem baznih postaj in preklopnim centrom ali pa, v primeru, da gre povezava v stacionarno omrežje, tudi na delu stacionarnega omrežja do uporabnika, s katerim je vzpostavljena povezava.

6.5 Napad na preklopni center

Mobilni operater Vodafone v Grčiji je odkril, da so neznanci zamenjali firm- ware v preklopnih centrih in tako prisluškovali več kot 100 mobilnim telefonom. Preklopne centre so sprogramirali tako, da so ob uporabi določenih mobilnih telefonov vzpostavili konferenčno zvezo z enim od telefonov napadalcev, ki je pogovor napadene osebe snemal.

6.6 Kriptoanaliza algoritma A5

Leta 1997 je Golić [4] izvedel napad, ki temelji na reševanju sistema linearnih enačb in ima časovno kompleksnost $2^{40.16}$ (potrebnih rešitev sistema linearnih enačb).

Leta 2000 so Biryukov, Shamir in Wagner [5] izvedli napad v realnem času, ki temelji na napadu Golića. Za napad je potrebno izvesti predprocesiranje z 2^{48} koraki, ki izračuna okrog 300 GB podatkov. S tem napadom so lahko dobili ključ v nekaj minutah iz samo dveh sekund čistopisa. Pri tem napadu je možnih več različic z

različnim razmerjem med količino predprocesiranja, potrebnega čistopisa ter časa in spomina, potrebnega za napad.

Biham in Dunkelman sta še istega leta objavila napad, ki potrebuje $2^{20.8}$ bitov čistopisa, 32 GB prostora za podatke, 2^{38} korakov predprocesiranja in ima časovno kompleksnost $2^{39.91}$.

Leta 2003 sta Ekdahl in Johannson [6] predstavila napad na inicializacijski del algoritma, ki razbije A5 v nekaj minutah z uporabo 2-5 minut čistopisa in ne potrebuje predprocesiranja.

Maximov s sodelavci je leta 2004 ta napad še izboljšal in dosegel hitrost pod minuto pri nekaj sekundah poznanega pogovora.

Leta 2003 je Barkan [7] s sodelavci objavil množico napadov, ki uporabljajo samo tajnopis. Najlaži napad je aktiven, kar pomeni, da nekako prepričamo napadeni GSM, da uporablja algoritem A5/2 s šibkejso enkripcijo. A5/2 lahko zaradi prešibke enkripcije razbijemo z lahkoto, ključ, ki ga dobimo, pa je enak ključu, ki se uporablja pri kriptografsko močnejšem algoritmu A5/1. S pasivnim napadom na A5/1 lahko s predprocesiranjem, ki traja okrog 684 računalniških let in 50 TB prostora izračunamo ključ v približno 200 sekundah. Bolj podroben opis teh napadov je razložen v članku [7].

6.7 Napad na COMP128

Napad temelji na šibki točki kompresijske funkcije. Izhodni bajti v drugem krogu kompresije i, i+8, i+16 in i+24 so odvisni samo od njim pripadajočih vhodnih bajtov.

SIM kartici pošiljamo veliko število nalog, v katerih spreminjamo samo bajta i in i+8. Zaradi odvisnosti samo od 2 bajtov v prejšnjem krogu, se trk v drugem krogu prenese do konca in dobimo trk pri rezultatu. Po paradoksu rojstnih dni se trk zgodi najkasneje po 2^{14} nalogah. Po najdenem trku lahko ostale bajte izračunamo v največ 2^{17} korakih.

Če uporabljamo čitalec pametnih kartic, lahko ob hitrosti 6 nalog na sekundo najdemo ključ v približno šestih urah. Z dobljenim zasebnim ključem lahko kloniramo SIM kartico in uporabljamo storitve na račun napadenega. Vendar dobiti SIM kartico za 6 ur predstavlja kar velik problem, poleg tega pa imajo novejše kartice zaščito, ki po 2^{16} nalogah kartico zaklene.

7 Zaključek

Kot ponavadi se varovanje algoritmov s skrivanjem specifikacij ni pokazalo za dobro. Običajno se uporablja preverjene algoritme, ki vsebujejo čim manj pomanjkljivosti, specifikacije se javno objavi, skrit ostane samo zasebni ključ. Pri algoritmih za komunikacijo z GSM pa so ubrali drugačno taktiko. Spisali so enostavne algoritme in jih pomanjkljivo pretestirali. Da bi zaščitili omrežje pred napadi, so specifikacije skrbno skrivali. Vendar sčasoma skrite informacije, ki jih ve veliko ljudi (razvijalcev), vedno pridejo v javnost in enako se je zgodilo v tem primeru, kar je močno zmanjšalo varnost v GSM omrežju.

Vendar situacija ni tako tragična kot izgleda. Za običajne uporabnike, ki bi radi prisluškovali določeni osebi, je oprema potrebna za ta namen še predraga. Dostopna je le bogatejšim inštitucijam, kamor štejejo tudi državni organi. Zaradi tega razloga je uporabljanje mobilnih storitev za vsakdanjo uporabo še dovolj varno, za prenašanje občutljivih podatkov pa je zaradi pomanjkljivosti obstoječih algoritmov bolje poseči po kakšni drugi poti.

Slabost skrivanja specifikacij in uporabe neučinkovitih algoritmov pa je očitno izučila snovalce novejših algoritmov. Algoritmi za UMTS so namreč zasnovani na preverjenih in znanih algoritmih, kot sta Kasumi in AES, kar naj bi v prihodnosti izboljšalo varnost storitev.

8 Koda algoritma COMP128

```
/* An implementation of the GSM A3A8 algorithm.(Specifically,
   COMP128.)
 *
 * Copyright 1998, Marc Briceno, Ian Goldberg,
 * and David Wagner. All rights reserved.
 *
 * For expository purposes only.  Coded in C merely because C
 * is a much more precise, concise form of expression for these
 * purposes.  See Judge Patel if you have any problems with
 * this... Of course, it's only authentication, so it should be
 * exportable for the usual boring reasons.
 *
 *
 * This software is free for commercial and non-commercial use
 * as long as the following conditions are aheared to.
 * Copyright remains the authors' and as such any Copyright
 * notices in the code are not to be removed.
```

```
 * Redistribution and use in source and binary forms, with or
 * without modification, are permitted provided that the
 * following conditions are met:
 *
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following
 *    disclaimer.
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the
 *    following disclaimer in the documentation and/or other
 *    materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE
 * AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION)HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
 * DAMAGE.
 *
 * The license and distribution terms for any publicly available
 * version or derivative of this code cannot be changed.  i.e.
 * this code cannot simply be copied and put under another
 * distribution license [including the GNU Public License.]
 */

typedef unsigned char Byte;

#include <stdio.h> /* #define TEST */

/*
 * rand[0..15]: the challenge from the base station
 * key[0..15]: the SIM's A3/A8 long-term key Ki
 * simoutput[0..11]: what you'd get back if you fed rand and key
 * to a real SIM.
 *
 *   The GSM spec states that simoutput[0..3] is SRES,
 *   and simoutput[4..11] is Kc (the A5 session key).
```

```
 *   (See GSM 11.11, Section 8.16.  See also the leaked document
 *   referenced below.)
 *   Note that Kc is bits 74..127 of the COMP128 output, followed
 *   by 10 zeros.
 *   In other words, A5 is keyed with only 54 bits of entropy.
 *   This represents a deliberate weakening of the key used for
 *   voice privacy by a factor of over 1000.
 *
 * Verified with a Pacific Bell Schlumberger SIM.  Your mileage
 * may vary.
 *
 * Marc Briceno <marc@scard.org>,
 * Ian Goldberg <iang@cs.berkeley.edu>,
 * and David Wagner <daw@cs.berkeley.edu>
 */

void A3A8(/* in */ Byte rand[16], /* in */ Byte key[16],
    /* out */ Byte simoutput[12]);

/* The compression tables. */
static const Byte table_0[512] = {
    102,177,186,162,  2,156,112, 75, 55, 25,  8, 12,251,193,246,188,
    109,213,151, 53, 42, 79,191,115,233,242,164,223,209,148,108,161,
    252, 37,244, 47, 64,211,  6,237,185,160,139,113, 76,138, 59, 70,
    67, 26, 13,157, 63,179,221, 30,214, 36,166, 69,152,124,207,116,
    247,194, 41, 84, 71,  1, 49, 14, 95, 35,169, 21, 96, 78,215,225,
    182,243, 28, 92,201,118,  4, 74,248,128, 17, 11,146,132,245, 48,
    149, 90,120, 39, 87,230,106,232,175, 19,126,190,202,141,137,176,
    250, 27,101, 40,219,227, 58, 20, 51,178, 98,216,140, 22, 32,121,
    61,103,203, 72, 29,110, 85,212,180,204,150,183, 15, 66,172,196,
    56,197,158,  0,100, 45,153,  7,144,222,163,167, 60,135,210,231,
    174,165, 38,249,224, 34,220,229,217,208,241, 68,206,189,125,255,
    239, 54,168, 89,123,122, 73,145,117,234,143, 99,129,200,192, 82,
    104,170,136,235, 93, 81,205,173,236, 94,105, 52, 46,228,198,  5,
    57,254, 97,155,142,133,199,171,187, 50, 65,181,127,107,147,226,
    184,218,131, 33, 77, 86, 31, 44, 88, 62,238, 18, 24, 43,154, 23,
    80,159,134,111,  9,114,  3, 91, 16,130, 83, 10,195,240,253,119,
    177,102,162,186,156,  2, 75,112, 25, 55, 12,  8,193,251,188,246,
    213,109, 53,151, 79, 42,115,191,242,233,223,164,148,209,161,108,
    37,252, 47,244,211, 64,237,  6,160,185,113,139,138, 76, 70, 59,
    26, 67,157, 13,179, 63, 30,221, 36,214, 69,166,124,152,116,207,
    194,247, 84, 41,  1, 71, 14, 49, 35, 95, 21,169, 78, 96,225,215,
    243,182, 92, 28,118,201, 74,  4,128,248, 11, 17,132,146, 48,245,
    90,149, 39,120,230, 87,232,106, 19,175,190,126,141,202,176,137,
```

```
    27,250, 40,101,227,219, 20, 58,178, 51,216, 98, 22,140,121, 32,
    103, 61, 72,203,110, 29,212, 85,204,180,183,150, 66, 15,196,172,
    197, 56,  0,158, 45,100,  7,153,222,144,167,163,135, 60,231,210,
    165,174,249, 38, 34,224,229,220,208,217, 68,241,189,206,255,125,
    54,239, 89,168,122,123,145, 73,234,117, 99,143,200,129, 82,192,
    170,104,235,136, 81, 93,173,205, 94,236, 52,105,228, 46,  5,198,
    254, 57,155, 97,133,142,171,199, 50,187,181, 65,107,127,226,147,
    218,184, 33,131, 86, 77, 44, 31, 62, 88, 18,238, 43, 24, 23,154,
    159, 80,111,134,114,  9, 91,  3,130, 16, 10, 83,240,195,119,253
}, table_1[256] = {
    19, 11, 80,114, 43,  1, 69, 94, 39, 18,127,117, 97,  3, 85, 43,
    27,124, 70, 83, 47, 71, 63, 10, 47, 89, 79,  4, 14, 59, 11,  5,
    35,107,103, 68, 21, 86, 36, 91, 85,126, 32, 50,109, 94,120,  6,
    53, 79, 28, 45, 99, 95, 41, 34, 88, 68, 93, 55,110,125,105, 20,
    90, 80, 76, 96, 23, 60, 89, 64,121, 56, 14, 74,101,  8, 19, 78,
    76, 66,104, 46,111, 50, 32,  3, 39,  0, 58, 25, 92, 22, 18, 51,
    57, 65,119,116, 22,109,  7, 86, 59, 93, 62,110, 78, 99, 77, 67,
    12,113, 87, 98,102,  5, 88, 33, 38, 56, 23,  8, 75, 45, 13, 75,
    95, 63, 28, 49,123,120, 20,112, 44, 30, 15, 98,106,  2,103, 29,
    82,107, 42,124, 24, 30, 41, 16,108,100,117, 40, 73, 40,  7,114,
    82,115, 36,112, 12,102,100, 84, 92, 48, 72, 97,  9, 54, 55, 74,
    113,123, 17, 26, 53, 58,  4,  9, 69,122, 21,118, 42, 60, 27, 73,
    118,125, 34, 15, 65,115, 84, 64, 62, 81, 70,  1, 24,111,121, 83,
    104, 81, 49,127, 48,105, 31, 10,  6, 91, 87, 37, 16, 54,116,126,
    31, 38, 13,  0, 72,106, 77, 61, 26, 67, 46, 29, 96, 37, 61, 52,
    101, 17, 44,108, 71, 52, 66, 57, 33, 51, 25, 90,  2,119,122, 35
}, table_2[128] = {
    52, 50, 44,  6, 21, 49, 41, 59, 39, 51, 25, 32, 51, 47, 52, 43,
    37,  4, 40, 34, 61, 12, 28,  4, 58, 23,  8, 15, 12, 22,  9, 18,
    55, 10, 33, 35, 50,  1, 43,  3, 57, 13, 62, 14,  7, 42, 44, 59,
    62, 57, 27,  6,  8, 31, 26, 54, 41, 22, 45, 20, 39,  3, 16, 56,
    48,  2, 21, 28, 36, 42, 60, 33, 34, 18,  0, 11, 24, 10, 17, 61,
    29, 14, 45, 26, 55, 46, 11, 17, 54, 46,  9, 24, 30, 60, 32,  0,
    20, 38,  2, 30, 58, 35,  1, 16, 56, 40, 23, 48, 13, 19, 19, 27,
    31, 53, 47, 38, 63, 15, 49,  5, 37, 53, 25, 36, 63, 29,  5,  7
}, table_3[64] = {
    1,  5, 29,  6, 25,  1, 18, 23, 17, 19,  0,  9, 24, 25,  6, 31,
    28, 20, 24, 30,  4, 27,  3, 13, 15, 16, 14, 18,  4,  3,  8,  9,
    20,  0, 12, 26, 21,  8, 28,  2, 29,  2, 15,  7, 11, 22, 14, 10,
    17, 21, 12, 30, 26, 27, 16, 31, 11,  7, 13, 23, 10,  5, 22, 19
}, table_4[32] = {
    15, 12, 10,  4,  1, 14, 11,  7,  5,  0, 14,  7,  1,  2, 13,  8,
    10,  3,  4,  9,  6,  0,  3,  2,  5,  6,  8,  9, 11, 13, 15, 12
}, *table[5] = { table_0, table_1, table_2, table_3, table_4 };
```

```
/*
 * This code derived from a leaked document from the GSM standards.
 * Some missing pieces were filled in by reverse-engineering a
 * working SIM. We have verified that this is the correct COMP128
 * algorithm.
 *
 * The first page of the document identifies it as
 *  _Technical Information: GSM System Security Study_.
 *  10-1617-01, 10th June 1988.
 * The bottom of the title page is marked
 *  Racal Research Ltd.
 *  Worton Drive, Worton Grange Industrial Estate,
 *  Reading, Berks. RG2 0SB, England.
 *  Telephone: Reading (0734) 868601   Telex: 847152
 * The relevant bits are in Part I, Section 20 (pages 66--67).
 * Enjoy!
 *
 * Note: There are three typos in the spec (discovered by
 * reverse-engineering).
 * First, "z = (2 * x[n] + x[n]) mod 2^(9-j)" should clearly read
 * "z = (2 * x[m] + x[n]) mod 2^(9-j)".
 * Second, the "k" loop in the "Form bits from bytes" section is
 * severely botched: the k index should run only from 0 to 3, and
 * clearly the range on "the (8-k)th bit of byte j" is also off
 * (should be 0..7, not 1..8, to be consistent with the subsequent
 * section). Third, SRES is taken from the first 8 nibbles of x[],
 * not the last 8 as claimed in the document.  (And the document
 * doesn't specify how Kc is derived, but that was also easily
 * discovered with reverse engineering.) All of these typos have
 * been corrected in the following code.
 */

void A3A8(/* in */ Byte rand[16], /* in */ Byte key[16],
    /* out */ Byte simoutput[12])
{
    Byte x[32], bit[128];
    int i, j, k, l, m, n, y, z, next_bit;

    /* ( Load RAND into last 16 bytes of input ) */
    for (i=16; i<32; i++)
        x[i] = rand[i-16];

    /* ( Loop eight times ) */
```

```
for (i=1; i<9; i++) {
    /* ( Load key into first 16 bytes of input ) */
    for (j=0; j<16; j++)
        x[j] = key[j];
    /* ( Perform substitutions ) */
    for (j=0; j<5; j++)
        for (k=0; k<(1<<j); k++)
            for (l=0; l<(1<<(4-j)); l++) {
                m = l + k*(1<<(5-j));
                n = m + (1<<(4-j));
                y = (x[m]+2*x[n]) % (1<<(9-j));
                z = (2*x[m]+x[n]) % (1<<(9-j));
                x[m] = table[j][y];
                x[n] = table[j][z];
            }
    /* ( Form bits from bytes ) */
    for (j=0; j<32; j++)
        for (k=0; k<4; k++)
            bit[4*j+k] = (x[j]>>(3-k)) & 1;
    /* ( Permutation but not on the last loop ) */
    if (i < 8)
        for (j=0; j<16; j++) {
            x[j+16] = 0;
            for (k=0; k<8; k++) {
                next_bit = ((8*j + k)*17) % 128;
                x[j+16] |= bit[next_bit] << (7-k);
            }
        }
}

/*
 * ( At this stage the vector x[] consists of 32 nibbles.
 *   The first 8 of these are taken as the output SRES. )
 */

/* The remainder of the code is not given explicitly in the
 * standard, but was derived by reverse-engineering.
 */

for (i=0; i<4; i++)
    simoutput[i] = (x[2*i]<<4) | x[2*i+1];
for (i=0; i<6; i++)
    simoutput[4+i] = (x[2*i+18]<<6) | (x[2*i+18+1]<<2)
            | (x[2*i+18+2]>>2);
```

```
    simoutput[4+6] = (x[2*6+18]<<6) | (x[2*6+18+1]<<2);
    simoutput[4+7] = 0;
}



#ifdef TEST int hextoint(char x) {
    x = toupper(x);
    if (x >= 'A' && x <= 'F')
        return x-'A'+10;
    else if (x >= '0' && x <= '9')
        return x-'0';
    fprintf(stderr, "bad input.\n");
    exit(1);
}


int main(int argc, char **argv) {
    Byte rand[16], key [16], simoutput[12];
    int i;

    if (argc != 3 || strlen(argv[1]) != 34 || strlen(argv[2]) != 34
            || strncmp(argv[1], "0x", 2) != 0
            || strncmp(argv[2], "0x", 2) != 0) {
        fprintf(stderr, "Usage: %s 0x<key> 0x<rand>\n", argv[0]);
        exit(1);
    }

    for (i=0; i<16; i++)
        key[i] = (hextoint(argv[1][2*i+2])<<4)
            | hextoint(argv[1][2*i+3]);
    for (i=0; i<16; i++)
        rand[i] = (hextoint(argv[2][2*i+2])<<4)
             | hextoint(argv[2][2*i+3]);
    A3A8(key, rand, simoutput);
    printf("simoutput: ");
    for (i=0; i<12; i++)
        printf("%02X", simoutput[i]);
    printf("\n");
    return 0;
} #endif
```

9 Koda algoritma A5

```
/* A pedagogical implementation of the GSM A5/1 and A5/2 "voice
 * privacy" encryption algorithms.
 *
 * Copyright (C) 1998-1999: Marc Briceno, Ian Goldberg,
 * and David Wagner
 *
 * The source code below is optimized for instructional value
 * and clarity. Performance will be terrible, but that's not
 * the point.
 *
 * This software may be export-controlled by US law.
 *
 * This software is free for commercial and non-commercial use
 * as long as the following conditions are adhered to.
 * Copyright remains the authors' and as such any Copyright
 * notices in the code are not to be removed.
 * Redistribution and use in source and binary forms, with or
 * without modification, are permitted provided that the following
 * conditions are met:
 *
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above
 *    copyright notice, this list of conditions and the following
 *    disclaimer in the documentation and/or other materials provided
 *    with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
 * OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
 * USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
 * DAMAGE.
 *
 * The license and distribution terms for any publicly available
 * version or derivative of this code cannot be changed.  i.e. this
```

```
* code cannot simply be copied and put under another distribution
* license [including the GNU Public License].
*
* Background: The Global System for Mobile communications is the
* most widely deployed digital cellular telephony system in the
* world. GSM makes use of four core cryptographic algorithms, none
* of which has been published by the GSM MOU.  This failure to
* subject the algorithms to public review is all the more puzzling
* given that over 215 million GSM subscribers are expected to rely
* on the claimed security of the system.
*
* The four core GSM cryptographic algorithms are:
* A3              authentication algorithm
* A5/1 "stronger" over-the-air voice-privacy algorithm
* A5/2  "weaker"  over-the-air voice-privacy algorithm
* A8              voice-privacy key generation algorithm
*
* In April of 1998, our group showed that COMP128, the algorithm
* used by the overwhelming majority of GSM providers for both A3
* and A8 functionality is fatally flawed and allows for cloning of
* GSM mobile phones.
*
* Furthermore, we demonstrated that all A8 implementations we could
* locate, including the few that did not use COMP128 for key
* generation, had been deliberately weakened by reducing the
* keyspace from 64 bits to 54 bits. The remaining 10 bits are simply
* set to zero!
*
* See http://www.scard.org/gsm for additional information.
*
* [May 1999]
* One question so far unanswered is if A5/1, the "stronger" of the
* two widely deployed voice-privacy algorithm is at least as strong
* as the key. Meaning: "Does A5/1 have a work factor of at least
* 54 bits"? Absent a publicly available A5/1 reference implementation,
* this question could not be answered. We hope that our reference
* implementation below, which has been verified against official A5/1
* test vectors, will provide the cryptographic community with the base
* on which to construct the answer to this important question.
*
* Initial indications about the strength of A5/1 are not encouraging.
* A variant of A5, while not A5/1 itself, has been estimated to have a
* work factor of well below 54 bits. See http://jya.com/crack-a5.htm
* for background information and references.
```

```
 *
 * With COMP128 broken and A5/1 published below, we will now turn our
 * attention to A5/2.
 *
 * [August 1999]
 * 19th Annual International Cryptology Conference - Crypto'99
 * Santa Barbara, California
 *
 * A5/2 has been added to the previously published A5/1 source. Our
 * implementation has been verified against official test vectors.
 *
 * This means that our group has now reverse engineered the entire set
 * of cryptographic algorithms used in the overwhelming majority of GSM
 * installations, including all the over-the-air "voice privacy"
 * algorithms.
 *
 * The "voice privacy" algorithm A5/2 proved especially weak. Which
 * perhaps should come as no surprise, since even GSM MOU members have
 * admitted that A5/2 was designed with heavy input by intelligence
 * agencies to ensure
 * breakability. Just how insecure is A5/2?  It can be broken in real
 * time with a work factor of a mere 16 bits. GSM might just as well use
 * no "voice privacy" algorithm at all.
 *
 * We announced the break of A5/2 at the Crypto'99 Rump Session.
 * Details will be published in a scientific paper following soon.
 *
 *
 * -- Marc Briceno      <marc@scard.org>
 *    Voice:            +1 (925) 798-4042
 *
 */
```

```
#include <stdio.h>


/* Masks for the shift registers */
#define R1MASK  0x07FFFF /* 19 bits, numbered 0..18 */
#define R2MASK  0x3FFFFF /* 22 bits, numbered 0..21 */
#define R3MASK  0x7FFFFF /* 23 bits, numbered 0..22 */
```

```
#ifdef A5_2
    #define R4MASK  0x01FFFF /* 17 bits, numbered 0..16 */
#endif /* A5_2 */


#ifndef A5_2
/* Middle bit of each of the three shift registers, for
clock control */
    #define R1MID   0x000100 /* bit 8 */
    #define R2MID   0x000400 /* bit 10 */
    #define R3MID   0x000400 /* bit 10 */
#else /*A5_2 */
/* A bit of R4 that controls each of the shift registers */
    #define R4TAP1  0x000400 /* bit 10 */
    #define R4TAP2  0x000008 /* bit 3 */
    #define R4TAP3  0x000080 /* bit 7 */
#endif /* A5_2 */


/* Feedback taps, for clocking the shift registers.
 * These correspond to the primitive polynomials
 * x^19 + x^5 + x^2 + x + 1, x^22 + x + 1,
 * x^23 + x^15 + x^2 + x + 1, and x^17 + x^5 + 1. */


#define R1TAPS  0x072000 /* bits 18,17,16,13 */
#define R2TAPS 0x300000 /* bits 21,20 */
#define R3TAPS  0x700080 /* bits 22,21,20,7 */
#ifdef A5_2
    #define R4TAPS  0x010800 /* bits 16,11 */
#endif /* A5_2 */


typedef unsigned char byte;
typedef unsigned long word;
typedef word bit;


/* Calculate the parity of a 32-bit word, i.e. the sum of its bits
modulo 2 */ bit parity(word x) {
    x ^= x>>16;
    x ^= x>>8;
    x ^= x>>4;
```

```
    x ^= x>>2;
    x ^= x>>1;
    return x&1;
}


/* Clock one shift register.  For A5/2, when the last bit of the
 * frame is loaded in, one particular bit of each register is forced
 * to '1'; that bit is passed in as the last argument. */
#ifndef A5_2
    word clockone(word reg, word mask, word taps) {
#else/*A5_2 */
    word clockone(word reg, word mask, word taps, wordloaded_bit){
#endif /* A5_2 */
    word t = reg & taps;
    reg = (reg << 1) & mask;
    reg |= parity(t);
#ifdef A5_2
    reg |= loaded_bit;
#endif /* A5_2 */
    return reg;
}


/* The three shift registers.  They're in global variables to make
 * the code easier to understand.
 * A better implementation would not use global variables. */
word R1, R2, R3;
#ifdef A5_2
    word R4;
#endif /* A5_2 */


/* Return 1 if at least two of the parameter words are non-zero.*/
bit majority(word w1, word w2, word w3) {
    int sum = (w1 != 0) + (w2 != 0) + (w3 != 0);
    if (sum >= 2)
            return 1;
    else
            return 0;
}


/* Clock two or three of R1,R2,R3, with clock control
```

```
 * according to their middle bits.
 * Specifically, we clock Ri whenever Ri's middle bit
 * agrees with the majority value of the three middle bits.  For
 * A5/2, use particular bits of R4 instead of the middle bits.
 * Also, for A5/2, always clock R4.
 * If allP == 1, clock all three of R1,R2,R3, ignoring their middle
 * bits. This is only used for key setup.  If loaded == 1, then this
 * is the last bit of the frame number, and if we're doing A5/2, we
 * have to set a particular bit in each of the four registers. */
void clock(int allP, int loaded) {
#ifndef A5_2
    bit maj = majority(R1&R1MID, R2&R2MID, R3&R3MID);
    if (allP || (((R1&R1MID)!=0) == maj))
            R1 = clockone(R1, R1MASK, R1TAPS);
    if (allP || (((R2&R2MID)!=0) == maj))
            R2 = clockone(R2, R2MASK, R2TAPS);
    if (allP || (((R3&R3MID)!=0) == maj))
            R3 = clockone(R3, R3MASK, R3TAPS);
#else /* A5_2 */
    bit maj = majority(R4&R4TAP1, R4&R4TAP2, R4&R4TAP3);
    if (allP || (((R4&R4TAP1)!=0) == maj))
            R1 = clockone(R1, R1MASK, R1TAPS, loaded<<15);
    if (allP || (((R4&R4TAP2)!=0) == maj))
            R2 = clockone(R2, R2MASK, R2TAPS, loaded<<16);
    if (allP || (((R4&R4TAP3)!=0) == maj))
            R3 = clockone(R3, R3MASK, R3TAPS, loaded<<18);
    R4 = clockone(R4, R4MASK, R4TAPS, loaded<<10);
#endif /* A5_2 */ }
/* Generate an output bit from the current state.
 * You grab a bit from each register via the output generation
 * taps; then you XOR the resulting three bits.  For A5/2, in
 * addition to the top bit of each of R1,R2,R3, also XOR in a
 * majority function of three particular bits of the register
 * (one of them complemented) to make it non-linear.  Also, for
 * A5/2, delay the output by one clock cycle for some reason. */
bit getbit() {
        bit topbits = (((R1 >> 18) ^ (R2 >> 21) ^ (R3 >> 22)) & 0x01);
#ifndef A5_2
        return topbits;
#else /* A5_2 */
        static bit delaybit = 0;
        bit nowbit = delaybit;
        delaybit = (
            topbits
```

```
            ^ majority(R1&0x8000, (~R1)&0x4000, R1&0x1000)
            ^ majority((~R2)&0x10000, R2&0x2000, R2&0x200)
            ^ majority(R3&0x40000, R3&0x10000, (~R3)&0x2000)
            );
        return nowbit;
#endif /* A5_2 */ }


/* Do the A5 key setup.  This routine accepts a 64-bit key and
 * a 22-bit frame number. */
void keysetup(byte key[8], word frame) {
        int i;
        bit keybit, framebit;


        /* Zero out the shift registers. */
        R1 = R2 = R3 = 0;
#ifdef A5_2
        R4 = 0;
#endif /* A5_2 */


    /* Load the key into the shift registers,
     * LSB of first byte of key array first,
     * clocking each register once for every
     * key bit loaded.  (The usual clock
     * control rule is temporarily disabled.) */
    for (i=0; i<64; i++) {
            clock(1,0); /* always clock */
            keybit = (key[i/8] >> (i&7)) & 1;
            /* The i-th bit of the key */
            R1 ^= keybit; R2 ^= keybit; R3 ^= keybit;
#ifdef A5_2
            R4 ^= keybit;
#endif /* A5_2 */
    }


    /* Load the frame number into the shift registers, LSB
     * first, clocking each register once for every key bit
     * loaded. (The usual clock control rule is still
     * disabled.) For A5/2, signal when the last bit is being
     * clocked in. */
    for (i=0; i<22; i++) {
```

```
            clock(1,i==21); /* always clock */
            framebit = (frame >> i) & 1;
            /* The i-th bit of the frame # */
            R1 ^= framebit; R2 ^= framebit; R3 ^= framebit;
#ifdef A5_2
            R4 ^= framebit;
#endif /* A5_2 */
    }


    /* Run the shift registers for 100 clocks
     * to mix the keying material and frame number
     * together with output generation disabled,
     * so that there is sufficient avalanche.
     * We re-enable the majority-based clock control
     * rule from now on. */
    for (i=0; i<100; i++) {
            clock(0,0);
    }



    /* For A5/2, we have to load the delayed output bit.  This
     * does _not_ change the state of the registers.  For A5/1,
     * this is a no-op. */
    getbit();


    /* Now the key is properly set up. */
}


/* Generate output.  We generate 228 bits of
 * keystream output.  The first 114 bits is for
 * the A->B frame; the next 114 bits is for the
 * B->A frame.  You allocate a 15-byte buffer
 * for each direction, and this function fills
 * it in. */
void run(byte AtoBkeystream[], byte BtoAkeystream[]) {
    int i;

    /* Zero out the output buffers. */
    for (i=0; i<=113/8; i++)
            AtoBkeystream[i] = BtoAkeystream[i] = 0;
```

```
    /* Generate 114 bits of keystream for the
     * A->B direction.  Store it, MSB first. */
    for (i=0; i<114; i++) {
            clock(0,0);
            AtoBkeystream[i/8] |= getbit() << (7-(i&7));
    }


    /* Generate 114 bits of keystream for the
     * B->A direction.  Store it, MSB first. */
    for (i=0; i<114; i++) {
            clock(0,0);
            BtoAkeystream[i/8] |= getbit() << (7-(i&7));
    }
}


/* Test the code by comparing it against
 * a known-good test vector. */

void test() {

#ifndef A5_2
    byte key[8] = {0x12, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
    word frame = 0x134;
    byte goodAtoB[15] = { 0x53, 0x4E, 0xAA, 0x58, 0x2F, 0xE8, 0x15,
                    0x1A, 0xB6, 0xE1, 0x85, 0x5A, 0x72, 0x8C, 0x00 };
    byte goodBtoA[15] = { 0x24, 0xFD, 0x35, 0xA3, 0x5D, 0x5F, 0xB6,
                    0x52, 0x6D, 0x32, 0xF9, 0x06, 0xDF, 0x1A, 0xC0 };
#else /* A5_2 */
    byte key[8] = {0x00, 0xfc, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
    word frame = 0x21;
    byte goodAtoB[15] = { 0xf4, 0x51, 0x2c, 0xac, 0x13, 0x59, 0x37,
                   0x64, 0x46, 0x0b, 0x72, 0x2d, 0xad, 0xd5, 0x00 };
    byte goodBtoA[15] = { 0x48, 0x00, 0xd4, 0x32, 0x8e, 0x16, 0xa1,
                    0x4d, 0xcd, 0x7b, 0x97, 0x22, 0x26, 0x51, 0x00 };
#endif /* A5_2 */
    byte AtoB[15], BtoA[15];
    int i, failed=0;


    keysetup(key, frame);
```

```
    run(AtoB, BtoA);


    /* Compare against the test vector. */
    for (i=0; i<15; i++)
            if (AtoB[i] != goodAtoB[i])
                    failed = 1;
    for (i=0; i<15; i++)
            if (BtoA[i] != goodBtoA[i])
                    failed = 1;


    /* Print some debugging output. */
    printf("key: 0x");
    for (i=0; i<8; i++)
            printf("%02X", key[i]);
    printf("\n");
    printf("frame number: 0x%06X\n", (unsigned int)frame);
    printf("known good output:\n");
    printf(" A->B: 0x");
    for (i=0; i<15; i++)
            printf("%02X", goodAtoB[i]);
    printf("  B->A: 0x");
    for (i=0; i<15; i++)
            printf("%02X", goodBtoA[i]);
    printf("\n");
    printf("observed output:\n");
    printf(" A->B: 0x");
    for (i=0; i<15; i++)
            printf("%02X", AtoB[i]);
    printf("  B->A: 0x");
    for (i=0; i<15; i++)
            printf("%02X", BtoA[i]);
    printf("\n");

    if (!failed) {
            printf("Self-check succeeded: everything looks ok.\n");
            exit(0);
    } else {
            /* Problems!  The test vectors didn't compare*/
            printf("\nI don't know why this broke;");
            printf(" contact the authors.\n");
    }
}
```

```
int main(void) {
    test();
    return 0;
}
```

Literatura

[1] Barkan, Biham, Keller: Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication, Israel, 2006.

[2] Billy Brumley, A3/A8 & COMP128, Special Course on Cryptology, Helsinki University of Technology, 2004

[3] 3GPP, Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 2: KASUMI specification, 1999

[4] Jovan Dj. Golic, Cryptanalysis of Alleged A5 Stream Cipher, EUROCRYPT 1997, pp239–255

[5] Alex Biryukov, Adi Shamir and David Wagner, Real Time Cryptanalysis of A5/1 on a PC, FSE 2000, pp1–18

[6] Patrik Ekdahl and Thomas Johansson: Another attack on A5/1. IEEE Transactions on Information Theory 49(1), pp284–289, 2003

[7] Elad Barkan, Eli Biham and Nathan Keller, Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication, CRYPTO 2003, pp600–616

[8] Chris J. Mitchell, The Security of the GSM Air Interface Protocol, Royal Holloway, University of London, 2001

[9] Enciklopedija Wikipedia, http://www.wikipedia.org