Napad na KeeLoq s pomočjo meritve električne napetosti

Luka Goljevšček

1 Kratek opis šifre KeeLoq in njenega delovanja

Keeloq je simetrična bločna šifra, ki uporablja 64 bitni tajni ključ in zašifrira 32 bitov čistopisa v 32 bitov tajnopisa. Je patentirana, toda njeno delovanje je sedaj dobro dokumentirano. Šifra uporablja dva pomična registra, register K dolg 64 bitov za tajni ključ, ter register L dolg 32 bitov za sporočilo; na koncu je v njem tajnopis oz. čistopis, odvisno ali smo na strani oddajnika oz. sprejemnika. Posamezen korak enkripcije poteka tako, da vzame 2 specifična bita sporočila, NLF (ta vzame 5 specifičnih bitov sporočila, vrne en bit) ter 1 specifičen bit iz ključa, med katerimi se izvede operacija XOR. Nato se sporočilo (register L) zamakne v desno (odpade bit čisto na desni), na izpraznjeno mesto čisto na levi pa pride rezultat prejšnjih XOR operacij. Nato se še ključ samo zarotira v desno (cikličen zamik v desno, tj. vsebina bita čisto na desni gre na mesto čisto na levi). To se ponovi 528-krat. Odšifriranje poteka podobno, le da zamikamo registre v levo (oz. ključ rotiramo) in uporabljamo isto NLF funkcijo z malo drugačnimi parametri. Delovanje je v nadaljevanju podrobno opisano, NLF funkcija v 1.1, šifriranje v 1.2, dešifriranje pa v 1.3.

1.1 NLF funkcija

Funkcija NLF (nonlinear feedback function) vzame 5 bitov sporočila, izbira bitov je tako za šifriranje kot odšifriranje določena, več o drugih možnih izbirah v dokazu pravilnosti delovanja šifre. Zaenkrat samo definicija:

$$\mathrm{NLF}(\mathrm{x}_4, \mathrm{x}_3, \mathrm{x}_2, \mathrm{x}_1, \mathrm{x}_0) = \mathrm{x}_0 \oplus \mathrm{x}_1 \oplus \mathrm{x}_0\mathrm{x}_1 \oplus \mathrm{x}_1\mathrm{x}_2 \oplus \mathrm{x}_2\mathrm{x}_3 \oplus \mathrm{x}_0\mathrm{x}_3 \oplus \mathrm{x}_0\mathrm{x}_4 \oplus \mathrm{x}_2\mathrm{x}_4 \oplus$$

$$\oplus x_0x_1x_4 \oplus x_0x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4$$

Pri tem množenje (tj. x_0x_1 pomeni $x_0 \wedge x_1$), operacija $x_0 \oplus x_1$ pa x_0 XOR x_1 na logičnih vrednostih spremenljivk (spremenljivka je dejansko posamezen bit sporočila, vzamemo njihove logične vrednosti in naredimo logične operacije na njih).

NLF lahko predstavimo tudi malo drugače: na 5 parametrov lahko gledamo kot na petmestno število zapisano v dvojiškem sistemu (tj. od $00000_2 = 0$ do $11111_2 = 31$). Izkaže se, da je vrednost NLF(a, b, c, d, e) ravno enaka kot "(*abcde*)-ti bit"šestnajstiškega $3A5C742E_{16}$. Do rezultata pridemo s Karnaugh-ovo preslikavo (Karnaugh map, K-map), treba je le zapisati vse možne logične vrednosti spremenljivk $x_4, \ldots, x_0$, (kar pomeni

tabelo s 32 vrsticami), ter za vsako vrstico pripadajočo vrednost NLF$(x_4, x_3, x_2, x_1, x_0)$. Tako dobimo 32 mestno dvojiško število, ki ga zapišemo v šestnajstiško. To lastnost uporabljajo predvsem softwerske implementacije Keeloq-a (sprejemniki).

Primer: $3A5C742E_{16} = 00111010010111000111010000101110_2$. Če želimo izračunati NLF$(0,0,0,0,1) = 1$, vzamemo število $00001_2 = 1_{10}$, rezultat pa je vrednost prvega bita z desne (najbolj desni bit ima indeks 0). Podobno NLF$(1,1,1,1,1) = 0$, $11111_2 = 31_{10}$, vzamemo najbolj levi (tj. 31.) bit.

1.2 Šifriranje

Algoritem:

Vhod: čistopis(32b), ključ(64b)
Izhod: tajnopis(32b)

1. shrani ključ v register ključa $K = (k_{63}, k_{62}, \ldots k_1, k_0)$,
2. shrani čistopis v register stanja $L = (l_{31}, l_{30}, \ldots, l_0)$,
3. for $i = 0, 1, 2, \ldots, 527$ do:
   (a) $\varphi = NLF(l_{31}, l_{26}, l_{20}, l_9, l_1) \oplus l_{16} \oplus l_0 \oplus k_0$,
   (b) $L = (\varphi, l_{31}, l_{30}, \ldots, l_1)$ (zamikamo register L v desno, v najbolj levi bit pride φ, najbolj desni bit odpade),
   (c) $K = (k_0, k_{63}, k_{62}, \ldots, k_1)$ (ključ ciklično zamikamo v desno, v najbolj levi bit pride prejšnji najbolj desni bit),
4. Vrni L.

Jedro šifriranja je for zanka, ta se ponovi 528-krat, na vsakem koraku zanke se izračuna nov bit φ, stran pa odpade en "star" bit. Ključ se samo ciklično zamika, načeloma bi lahko na i-tem koraku vzeli kar $j = i \bmod 64$ -ti bit registra K (tj. k_j), saj se ključ po 64 korakih zanke ponovi.

1.3 Odšifriranje

Odšifriranje je zelo podobno šifriranju, le da registra zamika v drugo smer ter da so indeksi pri računanju φ različni. Opazimo pa, da so indeksi v NLF ravno za 1 manjši od indeksov pri šifriranju. Nekaj opažanj o "izbirah" indeksov oz. možnih modifikacijah šifre ob zahtevi simetričnosti je opisanih po dokazu pravilnosti šifre.

Algoritem:

Vhod: tajnopis(32b), ključ(64b)
Izhod: čistopis(32b)

1. shrani ključ v register ključa $K = (k_{63}, k_{62}, \dots k_1, k_0)$
2. shrani tajnopis v register stanja $L = (l_{31}, l_{30}, \dots, l_0)$
3. for $i = 0, 1, \dots, 527$ do:
   (a) $\varphi = NLF(l_{30}, l_{25}, l_{19}, l_8, l_0) \oplus l_{31} \oplus l_{15} \oplus k_{15}$
   (b) $L = (l_{30}, l_{29}, \dots, l_0, \varphi)$ (shift v levo, izgubimo najpomembnejši bit, na najmanj pomemben bit pride φ)
   (c) $K = (k_{62}, k_{61}, \dots, k_0, k_{63})$ (ključ zarotiramo v levo)
4. Vrni L

1.4 Pravilnost delovanja

Pravilnost delovanja algoritmov šifriranja in odšifriranja bomo podali v obliki izreka.

Izrek: Če označimo šifriranje (z algoritmom iz 1.2) čistopisa x s ključem k z $E(x, k)$ in odšifriranje (z algoritmom iz 1.3) le-tega z $D(x, k)$, potem velja $D(E(x, k), k) = x$ za poljuben 32 bitni čistopis x in poljuben 64 bitni ključ k.

Dokaz. Zaradi lažjega razumevanja pozabimo pomičen register stanja in ga nadomestimo s tabelo dolžine 560 (indeksirana od $559, \dots, 0$), s tem, da je na mestih $31, \dots, 0$ originalno sporočilo - vsebina $559, \dots, 528$ pa je tajnopis, ki bi bil zapisan v registru L po 528 korakih. Na vsakem koraku for zanke pa izračunamo nov bit ter ga postavimo na $(i + 32)$-ti indeks. Bit ključa, ki se uporabi, računamo kar po modulu 64 – na ta način "odpravimo" oba pomična registra.

Register L predstavimo kot tabelo, v 528 korakih napolnimo indekse $32, \dots, 559$ od desne proti levi. Tajnopis je zapisan v zadnjih 32-bitih (ti so skrajno levo).

L_{559}	...	L_{527}	...	L_{31}	L_{30}	...	L_0

Pri tej predstavitvi je treba poudariti, da na i-tem koraku šifriranja "vidimo" le bite med $(i + 31, \dots, i)$, pri odšifriranju pa $(559 - i, \dots, 527 - i)$. Naj L_k označuje bit na k-tem indeksu v zgornji tabeli (tj. L_0 = najmanj pomemben bit / skrajno desni). Poleg tega je ključ konstanten.

Vzemimo sedaj poljuben 64 bitni ključ k in 32 bitni čistopis x ter z algoritmom enkripcije napolnimo zgornjo tabelo L. Sedaj vzamemo zadnjih 32 bitov tabele L (tj., tajnopis, $L(559-528)$), ter ga skopiramo v enako tabelo $T(559-528)$ (vsebine ostalih indeksov še ne poznamo) in na tabeli T uporabljamo algoritem dešifriranja (ta polni tabelo od leve proti desni).

T_{559}	...	T_{528}	T_{527}	...

Izračunamo $T(527)$ z enim korakom odšifriranja:

$$T(527) = \varphi_d = NLF(L'(30), L'(25), L'(19), L'(8), L'(0)) \oplus L'(15) \oplus L'(31) \oplus k_{15}.$$

L' je le oznaka skrajno levih bitov v tabeli T zaradi boljše preglednosti algoritma. $L'(31) = T(559) = L(559)$ oz. $L'(j) = L(528+j)$. Od tod tudi pravilnost, da odšifriranje "vidi"na vsakem koraku le 32-bitov, saj poznamo le 32 bitov T-ja, ostalih pa ne potrebujemo.

$$T(527) = NLF(L(558), L(553), L(547), L(536), L(528)) \oplus L(543) \oplus L(559) \oplus k_{15}. \quad (1)$$

Poglejmo sedaj, kako smo v enkripciji izračunali zadnji bit $L(559)$:

$$L(559) = \varphi_e = NLF(Z'(31), Z'(26), Z'(20), Z'(9), Z'(1)) \oplus Z'(16) \oplus Z'(0) \oplus k'_0.$$

Spet je treba razmisliti, kaj so Z' v zadnjem koraku šifriranja. V interpretaciji s tabelo je očitno, da so to ravno biti $L(558-527)$ in zato $Z'(j) = L(527+j)$, k'_0 pa je ravno $k_{527 \bmod 64} = k_{15}$. Vemo, da je v zadnjem koraku enkripcije $i = 527$, torej se uporabi $527 \bmod 64 = 15$. bit ključa. Zato:

$$L(559) = NLF(L(558), L(553), L(547), L(536), L(528)) \oplus L(543) \oplus L(527) \oplus k_{15}. \quad (2)$$

Opazimo, da so argumenti NLF v (1) in (2) enaki, vstavimo (2) v (1) in upoštevamo $A \oplus A = 0$.

$$T(527) = NLF \oplus L(543) \oplus L(559) \oplus k_{15} =$$

$$= NLF \oplus L(543) \oplus (NLF \oplus L(543) \oplus L(527) \oplus k_{15}) \oplus k_{15} =$$
$$= NLF \oplus NLF \oplus L(543) \oplus L(543) \oplus k_{15} \oplus k_{15} \oplus L(527) =$$
$$= 0 \oplus 0 \oplus 0 \oplus L(527).$$

Upoštevamo še, da je 0 enota za $\oplus$ (tj. $0 \oplus A = A$) in dobimo

$$T(527) = L(527). \quad (3)$$

Na tem mestu smo pokazali, da je prvi korak odšifriranja ravno inverz zadnjega koraka šifriranja. Načeloma bi bilo treba narediti to za poljuben par indeksov oblike $i, 527-i$ (tj: $527-i$-ti korak dešifriranja je ravno inverz i-temu koraku šifriranja), ampak zgornji postopek lahko "induktivno" ponovimo 528-krat. Ob upoštevanju (3) lahko pokažemo $T(526) = L(526)$ in nadaljujemo naprej do $T(0) = L(0)$, kar smo ravno hoteli dokazati. □

1.5 Možne modifikacije KeeLoq-a

Iz dokaza pravilnosti delovanja opazimo naslednje stvari:

1. NLF je pravzaprav "poljuben" v tem smislu, da je pravzaprav vseeno kakšno funkcijo uporabimo (iz stališča pravilnosti delovanja šifriranje/odšifriranje, seveda pa ne iz stališča varnosti, algebrski napadi). Pomembno je le, da za odšifriranje "zmanjšamo indekse bitov" za ena in da šifriranje ne uporabi bita z indeksom 0 (sicer bi moralo odšifriranje uporabiti 33 bitno okno). Kakršnokoli že funkcijo uporabimo na tako opisan način, bo le-ta na sebi uporabila ekskluzivni ali (XOR) v pripadajočem koraku šifriranja/odšifriranja in zato odpadla stran.

2. Algoritem vzame dva bita iz čistopisa (vzeti več istih bitov je nesmiselno zaradi XOR), važno je le, da odšifriranje vzame 31-bit, šifriranje pa ničti bit, za ostale pa moramo samo v algoritmu odšifriranja zmanjšati indekse parametrov za ena in se na ta način krajšajo (spet XOR sami na sebi). Načeloma bi lahko na tak način izbralii več bitov (KeeLoq vzame dva, en je pomemben, drug je pravzaprav odveč s stališča pravilnosti šifriranja/odšifriranja) za računanje naslednjega.

3. Algoritem uporabi le en bit ključa. Važno je le, da pri odšifriranju jemljemo $(p + 16)$-ti bit ključa (če v šifriranju uporabi p-ti bit, originalno $p = 0$), to pa zaradi 528 mod 64. Če bi spremenili število ponovitev zanke v šifriranjui na m, bi morali vzeti m mod 64-i indeks ključa. Podobno, če bi podaljšali ključ na j-bitov, bi morali vzeti m mod j-ti indeks ključa v odšifriranju.

2 Hipotetični napad z uporabo analize električne napetosti

Sam napad z analizo električne napetosti (angl. Power analysis, v bistvu gre za napad z uporabo t.i. stranskega kanala, iz samega delovanja kriptografske naprave dobimo nek podatek o ključu) predvideva fizičen dostop do nekega dela kriptografskega sistema. Glede na to, do katerega dela kriptografskega sistema imamo dostop, je potem možnih več scenarijev, omejil se bom na tistega, ki se zdi najenostavnejši (vendar morda ni najbolj realističen, zelo odvisno od strojne implementacije – zahteva lahko zelo drago opremo (focused ion beam), s katero lahko dostopamo do posameznih komponent čipov, ampak lahko merimo porabo na vsakem gradniku čipa).

Sam hipotetičen napad bo temeljil na dejstvu, da CMOS vezja (tipični gradniki) porabijo več (izmerjena napetost bo višja), ko pride do prehoda logičnih vrednosti (tj. $0 \to 1$ ali pa $1 \to 0$, kot pa če do prehoda ne pride, tj. $1 \to 1$ ali $0 \to 0$)[2]. Poleg tega je različna tudi poraba pri zapisu logičnih vrednosti v register (zapis 1 porabi več kot zapis 0). Ker se razlike pojavljajo na CMOS gradnikih, se jih da izmeriti (vprašanje je le, kako kvalitetno opremo potrebujemo), obstajajo tudi vezja drugih vrst, kjer ne prihaja do teh razlik, vendar se ne proizvajajo množično.

V nadaljevanju bom predpostavljal naslednje: izmerimo spremembe električne napetosti med enim pošiljanjem/sprejemanjem sporočila (tj. 528 ciklov KeeLoq-a, 1 šifriranje oz. odšifriranje), dobimo sled (angl. power trace),ki predstavlja graf $(t, V(t))$, iz nje lahko ločimo med posameznimi cikli in "lahko vidimo" katera logična vrednost se je zapisala v register stanja na koncu vsakega cikla.

2.1 O meritvah

Recimo, da bi radi znali interpretirati izmerjeno sled za čisto določen kriptografski sistem, ki uporablja KeeLoq. Uporabili bi kar "osnovno idejo" DPA, recimo da bi podtaknili 1000 naključnih sporočil kriptografski napravi v kateri poznamo (oz. lahko nastavimo) ključ, v vsakem bitu sporočila je zapisana 0 z verjetnostjo $\frac{1}{2}$. Vzeli bi samo prvi $\frac{1}{528}$ del sledi (kar ustreza koraku zanke), izračunali povprečno sled ter nato primerjali vmesne rezultate, ki se zapišejo v register stanja in njihove sledi s povprečno sledjo. Če opazimo konsistentno razliko (tj., ko se zapisuje 1 v register stanja oz. ko je rezultat XOR 1), potem lahko skoraj direktno preberemo ključ.

2.2 Še nekaj uporabnih dejstev o Keeloq

Za funkcijo NLF opazimo naslednje: $\mathrm{NLF}(0,0,0,0,0) = 0$, prav tako $\mathrm{NLF}(1,1,1,1,1) = 0$. Iz samega delovanja šifriranja(in dešifriranja) je potem jasno, da če kot sporočilo podtaknemo 32 bitov ničel, je $\varphi = \mathrm{NLF}(0,0,0,0,0) \oplus 0 \oplus 0 \oplus k_0 = 0 \oplus k_0 = k_j$ in se v register stanja zapiše ravno uporabljeni bit ključa. Prav tako, če za sporočilo podtaknemo 32 bitov samih enk $\varphi = \mathrm{NLF}(1,1,1,1,1) \oplus 1 \oplus 1 \oplus k_j = 0 \oplus 1 \oplus 1 \oplus k_j = 0 \oplus k_j = k_j$. Opazimo še naslednji dejstvi:

$$\mathrm{NLF}(0,0,0,0,k) = k \qquad \text{in} \qquad \mathrm{NLF}(0,0,0,x,y) = x \vee y.$$

Rezultate se preveri z neposrednim računom. Najbrž je bolj uporabno uporabljati kar sporočila samih ničel, saj so pri njem tudi vsi vmesni rezultati XOR operacij 0 in lahko upamo na bolj razvidne rezultate iz sledi.

Torej, če znamo podtakniti sporočilo (lahko ga podtaknemo kot čistopis ali pa kot tajnopis) samih ničel, bo na prvem koraku KeeLoq v register stanja zapisal prvi uporabljen bit ključa. Če znamo iz sledi napetosti le-to prebrati, je nadaljevanje enostavno, vendar bi se zaradi enostavnosti omejili na določen scenarij. Namreč, če podtikamo 32 bitov ničel kot tajnopis, potem sprejemni del kriptografske naprave na njem izvede dekodiranje, pri tem se uporabljajo biti ključa od bita na naslovu 15 po nazaj (in obratno, če podtikamo čistopis oddajniku, se uporabljajo biti od naslova 0 naprej)

2.3 Napad s fizičnim dostopom do sprejemnika

Predpostavke: imamo fizičen dostop do sprejemnika sporočil (realno je to možno, če je to sprejemnik za zapornico na parkirišču ipd.), na njem merimo električno napetost. Imamo

kopijo oddajnika, ne poznamo tajnega ključa, lahko pa pošiljamo sporočila poljubne oblike (ki jih prejemnik interpretira kot tajnopis).

Cilj napada: radi bi pridobili tajni ključ (64 bitov)

Sprejemniku pošljemo 32 bitno sporočilo samih ničel, ta ga bo interpretiral kot tajnopis in na njem izvedel dešifriranje(spodaj le jedro zanke).

1. $\varphi = NLF(l_{30}, l_{25}, l_{19}, l_8, l_0) \oplus l_{31} \oplus l_{15} \oplus k_{15}$,
2. $L = (l_{30}, l_{29}, \ldots, l_0, \varphi)$ (zamik v levo, izgubimo najpomembnejši bit, na najmanj pomemben bit pride φ),
3. $K = (k_{62}, k_{61}, \ldots, k_0, k_{63})$ (ključ ciklično zamaknemo v levo).

Po prvem koraku bo v registru stanja zapisano: $L = (0, \ldots, 0, k_{15})$. Predpostavka je bila, da znamo izmeriti, kaj k_{15} je. Če je njegova logična vrednost enaka 0, potem je v 2. koraku dešifriranja $L = (0, \ldots, k_{15} = 0, k_{14})$. Če pa je $k_{15} = 1$, potem je:

$$\varphi = NLF(0, 0, 0, 0, 1) \oplus 0 \oplus 0 \oplus k_{14} = 1 \oplus k_{14} = \neg k_{14}.$$

V registru stanja je tako po dveh korakih

$$L = (0, \ldots, k_{15} = 0, k_{14}) \qquad \text{ali} \qquad L = (0, \ldots, k_{15} = 1, \neg k_{14})$$

.
Na tem mestu bi bilo dobro razmisliti o splošnem koraku in algoritmu, saj v nasprotnem primeru (tj. obravnavamo vsak primer ločeno) dobimo 2^{32} možnosti (za vsak bit registra stanj 2 možnosti) kar je mnogo preveč. Vendar pa, iz meritev in enostavnega računa dejansko poznamo vsebino registra stanj na vsakem koraku.

Algoritem:

Ob predpostavki, da znamo za vsak korak izmeriti, kaj se zapiše v register stanja in da poznamo začetni register stanja (tj. podtaknjen tajnopis samih ničel) vrne tajni ključ.
Vhod: sled (power trace)
Izhod: tajni ključ (64 bitov)

1. $L = (0, \ldots, 0)$ (vsebina registra stanja),
2. $K = (0, \ldots, 0)$ (sem bomo zapisali ključ),
3. for $i = 0, 1, \ldots, 63$ do
   (a) $\psi = NLF(L_{30}, L_{25}, L_{19}, L_8, L_0) \oplus L_{31} \oplus L_{15}$,
   (b) določi k_d (iz sledi izmerimo, ali smo v register stanja zapisali 0 ali 1),
   (c) $L := (L_{30}, L_{29}, \ldots, L_0, k_d)$ (popravimo register stanja za naslednji korak),
   (d) iz ψ in k_d določimo logično vrednost uporabljenega bita ključa. Dejansko je treba rešiti bitno enačbo $k_d = \psi \oplus k_b$. Na srečo je ta enačba enolično rešljiva(logične enačbe z $\wedge, \vee$ niso!), rešitev je kar $k_b = k_d \oplus \psi$,
   (e) $K_{64-i+15+64 \bmod 64} = k_b$ (na pravo mesto zapišemo določeni bit ključa),
4. Vrni K.

2.4 Napad z dostopom do oddajnika, ki hrani tajni ključ

Predpostavke: imamo fizični dostop do oddajnika sporočil, na njem merimo električno napetost. V oddajniku je shranjen 64b dolg tajni ključ, lahko pa sami določamo vsebino čistopisa, ki bi ga radi zašifrirali. Na vsakem koraku šifriranja znamo izmeriti (določiti) logično vrednost, ki se zapiše v register stanja.

Cilj napada: pridobitev tajnega ključa

Ideja napada je skoraj enaka kot prej, spet oddajniku podtaknemo 32b samih ničel in izrabljamo dejstvo, da znamo iz podatka, kaj se zapiše v register stanja določiti logično vrednost uporabljenega bita ključa.

Algoritem:

Vhod: sled
Izhod: tajni ključ(64b)

1. $L = (0, \ldots, 0)$ (podtaknjen čistopis),
2. $K = (0, \ldots, 0)$ (sem bomo zapisali znane bite ključa),
3. for $i = 0, \ldots, 63$ do
   (a) $\psi = NLF(L_{31}, L_{26}, L_{20}, L_9, L_1) \oplus L_{16} \oplus L_0$,
   (b) določi k_e, tj. bit, ki se je zapisal v register stanja,
   (c) $L = (k_e, L_{31}, \ldots, L_1)$ (zashiftaj register stanja),
   (d) $k_b = \psi \oplus k_e$ (izračunaj kateri bit ključa je bil uporabljen),
   (e) $K_i = k_b$ (bit ključa na pravo mesto),
4. Vrni K.

2.5 Težave opisanih napadov

Glavna težava (poleg meritve) se pravzaprav skriva v dejstvu, da si s samim tajnim ključem ne moremo prav dosti pomagati. KeeLoq v praksi pogosto uporablja t.i. code hopping, kjer je le majhen del sporočila namenjen nekemu ukazu. Sporočilo je dolgo 32 bitov, od tega jih 10 (ali 12) predstavlja diskriminatorno vrednost, naslednjih 18 (ali 16) služi sinhronizacijskemu števcu, zadnji 4 pa so dejanski ukaz sprejemniku. Nekoliko bi si lahko pomagali s prestrezanjem sporočil med pravim oddajnikom in sprejemnikom, saj ko enkrat poznamo ključ lahko odšifriramo tajnopise in nato "poskušamo uganit" (če poznamo dovolj parov tajnopis/čistopis in njihov vrstni red, ne ugibamo več) kateri biti predstavljajo diskriminatorno število, kaj je števec in kaj ukaz ter nato povečati števec in z našim oddajnikom zašifrirati "pravilno" sporočilo.

Sporočilo poslano sprejemniku se odšifrira s pomočjo tajnega ključa. Ukaz oddajnika se upošteva le, če se diskriminatorni vrednosti ujemata in če sinhronizacijski števec pade v interval veljavnih vrednosti. Dejansko so biti sinhronizacijskega števca razdeljeni na tri "intervale". Če se poslani števec in zadnji shranjen števec razlikujeta za manj kot 16 vrednosti, potem se ukaz izvrši. Če se razlikujeta za več, potem se preveri drug interval (do 2^{15} vrednosti), ukaz se izvrši le, če sta dve zaporedni sinhronizacijski vrednosti znotraj teh vrednosti (uporabnik uporabi oddajnik dvakrat in se tega niti ne zaveda). Ostala sporočila so avtomatsko zavrnena, zato da ne bi prihajalo do t.i. replay napadov (shranimo sporočilo in ga ponovno uporabimo).

3 DPA napad na KeeLoq

3.1 Motivacija

Članek z naslovom Physical Cryptanalysis of KeeLoq Code Hopping Applications[2] trdi, da lahko z modificirano varianto DPA napada pridobimo tajni ključ oddajnika že z 10 do 30 sledmi porabe. Klasičen DPA ponavadi porabi nekaj tisoč sledi. Zakaj?

3.2 Modeli električne porabe

Kot porabo definiramo padec električne napetosti med izvajanjem operacije v vezju, ki nas zanima. V splošnem vezje ne izvaja bitnih operacij, ampak ima "večbitno" arhitekturo (tipičen namizni računalnik ima 32 bitno ali 64 bitno arhitekturo), razni mikrokontrolerji pa 8 bitno, hardverske implementacije pa tudi manj. Izkaže se, da se da pri mnogo hardwerskih in softwerskih implementacijah dobro modelirati električno porabo "večbitnega" vezja oz. njegovih delov z uporabo Hammingovih uteži oz. Hammingovih razdalj.

Definicija: Hammingovo težo (oznaka HW) niza bitov definiramo kot število bitov z logično vrednostjo ena.

Definicija: Hammingova razdalja (oznaka HD) dveh nizov bitov iste dolžine je enaka številu mest, kjer se logični vrednosti posameznih bitov v nizih razlikujeta.

Primera:

- 8 bitni niz 00011111 ima Hammingovo težo 5, tj. $\mathrm{HW}(00011111) = 5$.

- Hammingova razdalja med nizi: $\mathrm{HD}(10101010, 00000000) = 4$.

Trditev: Hammingova razdalja nizov s_1 in s_2 je enaka Hammingovi teži niza $s_1 \oplus s_2$. Matematično zapisano $\mathrm{HD}(s_1, s_2) = \mathrm{HW}(s_1 \oplus s_2)$.

Dokaz: s_1, s_2 poljubna niza bitov dolžine k. Naj bo i poljuben indeks v nizu $i \in \{0, 1, \ldots, k-1\}$. Potem zavzame $(s_1 \oplus s_2)_i$ logično vrednost 1 natanko takrat, ko imata $(s_1)_i$ in $(s_2)_i$ različni logični vrednosti. Odtod takoj sledi, da bit na indeksu i prispeva 1 v $\mathrm{HW}(s_1 \oplus s_2)$ natanko takrat, ko se bita na indeksu i razlikujeta, kar je točno takrat ko prispevata 1 v $\mathrm{HD}(s_1, s_2)$. (trditev je očitna).

Dejstvo, da porabo vezja lahko dobro modeliramo z modelom Hammingovih uteži oz. razdalj, lahko razumemo (oz. napadalec izrabi) tako, da bo poraba vezja v nekem trenutku dejansko izdajala Hammingove teže oz. razdalje in s tem neke vmesne vrednosti šifriranja oz. odšifriranja in s tem tudi podatek o uporabljenem ključu.

Poglejmo nekaj zelo enostavnih scenarijev, kako bi lahko uporabili modela Hammingovih uteži oz. razdalj za kriptografski napad. Zaradi enostavnosti se omejimo na 8 bitne ključe in čistopise, ideje lahko posplošimo, včasih jih pa niti ni treba. Npr. AES razdeli 128 bitni ključ na 16 8 bitnih"podključev". Natančna uporaba HD modela pri napadu na KeeLoq je opisana v 3.3.

- Če znamo izmeriti Hammingovo težo ključa ($\mathrm{HW}(K) = k$), to močno zmanjša prostor ključev, namesto $2^8 = 256$ imamo le še $\binom{8}{k} \leq \binom{8}{4} = 70$ možnosti. Vseeno to ni preveč verjeten scenarij, saj načeloma lahko izmerimo le Hammingovo razdaljo med dvema ključema (razen, ko se prvič ključ zapiše v register, taka meritev pa je zelo verjetno slaba, saj imajo vezja ob začetku delovanja več šuma).

- Če znamo izmeriti Hammingovo razdaljo med nekim znanim nizom P in neznanim ključem K, na podoben način kot zgoraj zmanjšamo prostor ključev. Uporabimo dejstvo $\mathrm{HD}(P, K) = \mathrm{HW}(P \oplus K) = k$ in hitro preverimo možne ključe, saj se mora ključ razlikovati od P v natanko k mestih, na $8 - k$ mestih pa je enak. "Različna mesta"lahko razporedimo v niz dolžine 8 na ravno $\binom{8}{k}$ načinov.

V praksi se izkaže, da so HD modeli boljši (v smislu, da bolje opišejo dejansko porabo vezja) in zelo dobro opišejo porabo podatkovnih vodil (bus) in registrov. Problem HD modela je, da je odvisen od dveh vrednosti in moramo eno vrednost poznati, da ga lahko uspešno uporabimo – vseeno to ni težava, če poznamo čistopis ali tajnopis in napadamo prvi korak šifriranja ali odšifriranja. Če tega ne moremo, nam še vedno ostane HW model, ki temelji na predpostavki, da je poraba sorazmerna s številom enk ali ničel v nizu.

Obstajajo tudi drugi modeli porabe, vendar sta zgoraj omenjena dva najbolj splošna in zahtevata najmanj predpostavk o vezjih.

3.3 DPA napad na KeeLoq

DPA napad temelji na ugibanju vmesne vrednosti, ki je odvisna od znanega niza (del čistopisa, tajnopisa) in neznanega niza (del ključa) in jo je lahko predvideti. Kot je že v opisu šifre KeeLoq opisano, implementacija potrebuje 64 bitni in 32 bitni register, nekaj operacij XOR in NLF. Izkaže se, da je poraba registrov dosti večja od porabe operacij (NLF, nekaj XOR), ki je zanemarljiva.

Porabo registrov modeliramo z modelom Hammingovih razdalj, osredotočimo se na register stanja (32 bitni register L). Med izvajanjem šifriranja ali odšifriranja se register ključa samo rotira in so vse Hammingove razdalje tega registra enake (dva zaporedna ključa sta vedno zamaknjena le za eno mesto, ta zamik se ohranja), kar pomeni konstantno porabo v teoretičnem modelu. Porabo registra stanja L pa model opiše takole

$$P_h^{(i)} = \mathrm{HD}(L^{(i)}, L^{(i-1)}) = \mathrm{HW}(L^{(i)} \oplus L^{(i-1)}),$$

kjer je $P_h^{(i)}$ teoretična poraba v i-tem koraku zanke v KeeLoq, $L^{(i)}$ pa vsebina registra stanja v i-tem koraku.

Kot je že prej omenjeno, je potrebno za uspešno uporabo HD modela poznati vsebino registra na prejšnjem koraku. Ponudita se dva scenarija napada na oddajnik(hardverska implementacija šifre), bodisi napadamo prvi korak šifriranja (poznamo čistopis) ali pa zadnji korak šifriranja (poznamo tajnopis). Bolj naraven je napad na zadnji korak šifriranja, saj je lažje določiti tajnopis (čistopisa načeloma ne poznamo).

Po zadnjem koraku šifriranja je v registru $L = L^{(528)} = (L_{31}^{(528)}, L_{30}^{(528)}, \ldots, L_0^{(528)})$, na predzadnjem koraku pa $(L_{31}^{(527)}, L_{30}^{(527)}, \ldots, L_0^{(527)})$. Od tega je 31 bitov enakih, natančneje:

$$L^{(528)} = (L_{31}^{(528)}, L_{30}^{(528)}, \ldots, L_0^{(528)}) = (L_{31}^{(528)}, L_{31}^{(527)}, \ldots, L_1^{(527)}).$$

Vrednost v registru na predzadnjem koraku je:

$$L^{(527)} = (L_{31}^{(527)}, \ldots, L_0^{(527)}) = (L_{30}^{(528)}, L_{29}^{(528)}, \ldots, L_0^{(528)}, L_0^{(527)}).$$

Zveza med njima pa je podana z:

$$L_{31}^{(528)} = \mathrm{NLF}(L_{31}^{(527)}, L_{26}^{(527)}, L_{20}^{(527)}, L_9^{(527)}, L_1^{(527)}) \oplus L_{16}^{(527)} \oplus L_0^{(527)} \oplus k_0^{(527)}.$$

Zadnjo enačbo lahko enolično rešimo na L_0^{527} kot enačbo, ki je odvisna le od vrednosti $L^{(528)}$ in bita ključa. (Za reševanje enačb z $\oplus$ le upoštevamo, da je vsaka spremenljivka "sebi nasproten" element za $\oplus$, tj. prištevanje spremenljivke k enačbi pomeni, da spremenljivka zamenja stran v enačbi – hkrati pa se ohranja enoličnost).

$$L_0^{(527)} = \mathrm{NLF}(L_{30}^{(528)}, L_{25}^{(528)}, L_{19}^{(528)}, L_8^{(528)}, L_0^{(528)}) \oplus L_{15}^{(528)} \oplus L_{31}^{(528)} \oplus k_0^{(527)}.$$

Če povzamem, poznavanje bita L_0^{527} pri znanem tajnopisu $L^{(528)}$ pomeni, da lahko izračunamo bit ključa:

$$k_0^{(527)} = \mathrm{NLF}(L_{30}^{(528)}, L_{25}^{(528)}, L_{19}^{(528)}, L_8^{(528)}, L_0^{(528)}) \oplus L_{15}^{(528)} \oplus L_{31}^{(528)} \oplus L_0^{(527)}.$$

Kaj lahko na tem mestu povemo o Hammingovih razdaljah med tajnopisom $L^{(528)}$ in predzanjo vrednostjo $L^{(527)}$ v registru stanj: v resnici poznamo 31 od 32-ih bitov v predzadnjem registru, HD pa je po bitih aditivna funkcija, torej

$$\mathrm{HD}(L^{(528)}, L^{(527)}) = \mathrm{HD}(L_{31}^{(528)}, L_{31}^{(527)}) + \ldots + \mathrm{HD}(L_1^{(528)}, L_1^{(527)}) + \mathrm{HD}(L_0^{(528)}, L_0^{(527)}).$$

V tej vsoti prvih 31 členov lahko izračunamo na podlagi znanega tajnopisa, saj:

$$\mathrm{HD}(L^{(528)}, L^{(527)}) = \sum_{i=1}^{31} \mathrm{HD}(L_i^{(528)}, L_{i+1}^{(528)}) + \mathrm{HD}(L_0^{(528)}, L_0^{(527)})$$

Spomnimo se modela porabe. Ta pravi:

$$P_h^{(528)} = \mathrm{HD}(L^{(528)}, L^{(527)}) = \sum_{i=1}^{31} \mathrm{HD}(L_i^{(528)}, L_{i+1}^{(528)}) + \mathrm{HD}(L_0^{(528)}, L_0^{(527)}).$$

Klasičen DPA napad bi na podlagi zgornje enačbe razvil preroka in poskušal pridobiti ključ bit za bitom. Vendar mogoče to ni najboljši način, saj je hipotetična poraba v modelu odvisna le od enega bita ključa (linearna odvisnost) in bi potrebovali mnogo sledi.

Radi bi modificirali preroka tako, da bi uporabil več bitov ključa naenkrat (ugibamo nekaj (m) zaporednih bitov) in upamo, da bodo razlike med hipotetičnimi porabami pri pravem ključu in napačnem ključu večje – to pa bi pomenilo, da potrebujemo manj sledi za določitev pravih vrednosti bitov. Za začetek poglejmo $m = 2$ oz. prerok temelji na $k_0^{(527)}, k_0^{(526)}$. Prvi korak je isti kot prej:

$$\mathrm{HD}(L^{(528)}, L^{(527)}) = \sum_{i=1}^{31} \mathrm{HD}(L_i^{(528)}, L_{i-1}^{(528)}) + \mathrm{HD}(L_0^{(528)}, L_0^{(527)}).$$

Drugi korak pa že temelji tudi na bitu, odvisnemu od $L_0^{(527)}$, vseeno pa še vedno poznamo 30 bitov, saj se v dveh korakih le dva spremenita:

$$\mathrm{HD}(L^{(527)}, L^{(526)}) = \sum_{i=2}^{31} \mathrm{HD}(L_{i-1}^{(528)}, L_{i-2}^{(528)}) + \mathrm{HD}(L_1^{(527)}, L_1^{(526)}) + \mathrm{HD}(L_0^{(527)}, L_0^{(526)}).$$

Poskušamo izraziti zgornje s členi iz čistopisa:

$$\mathrm{HD}(L^{(527)}, L^{(526)}) = \sum_{i=2}^{31} \mathrm{HD}(L_{i-1}^{(528)}, L_{i-2}^{(528)}) + \mathrm{HD}(L_0^{(528)}, L_0^{(527)}) + \mathrm{HD}(L_0^{(527)}, L_0^{(526)}).$$

Zanimiva sta le zadnja dva člena v zgornji vsoti:

$$\mathrm{HD}(L_0^{(528)}, L_0^{(527)}) = \mathrm{HD}(L_0^{(528)}, \mathrm{NLF}(L_{30}^{(528)}, L_{25}^{(528)}, L_{19}^{(528)}, L_8^{(528)}, L_0^{(528)}) \oplus$$
$$\oplus L_{15}^{(528)} \oplus L_{31}^{(528)} \oplus k_0^{(527)}),$$

$$\mathrm{HD}(L_0^{(527)}, L_0^{(526)}) = \mathrm{HD}(L_0^{(527)}, \mathrm{NLF}(L_{29}^{(528)}, L_{24}^{(528)}, L_{18}^{(528)}, L_7^{(528)}, L_0^{(527)}) \oplus$$
$$\oplus L_{14}^{(528)} \oplus L_{30}^{(528)} \oplus k_0^{(526)}).$$

V prvem členu nastopa $k_0^{(527)}$ le linearno, vendar se to spremeni v drugem členu, kjer $L_0^{(527)}$ prejšnjega koraka vstopi kot parameter nelinearne funkcije. To se pri klasičnem DPA napadu ne bi zgodilo, saj dela prerok na posameznih bitih ključa in se zato neznani

bit ključa nikoli ne uporabi nelinearno (ko se, je "že poznan"). Vseeno so ti sistemi enačb preveliki, da bi lahko prišli do analitičnih zaključkov, zato smo raje porabo modelirali.

Zgled: Zgled je iz modela porabe. Ključi so zamaknjeni v desno (uporablja se bit na indeksu 0, nato zamik desno), prerok ugiba samo skrajno desni bit (klasičen DPA). Na 30 tekstih (30 naključnih 32 bitnih števil zašifriranih s ključem (izpisan v predzadnji vrstici)). Po vrsticah so zapisani kvadrati razlik med porabo pri pravem ključu in i-tim ključem, ki ga generira prerok.V tem primeru le 0 in 1. V zadnji vrsti je zapisan najboljši trenutno znani kandidat za ključ.

Če je ugibana vrednost bita ključa enaka 0, potem se kvadrati razlik Hammingovih razdalj med pravo in hipotetično (bit = 0) porabo na vseh 30 tekstih razlikujejo za samo 4. Če za ugibani bit postavimo 1, potem je jasno razlika enaka 0. V zadnji vrsti je "trenutno" najboljši znan ključ.

```
4
0
10011100 11011000 00010011 11110100 01000010 11000000 01101101 01000001

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
```

Isto ponovimo na 300 naključnih tekstih, se rezultati nekoliko izboljšajo, vendar se je treba zavedati, da bi jih bilo treba koreniti in povprečiti za neko resno statistiko in primerjavo z sledjo.

```
144
0
10011100 11011000 00010011 11110100 01000010 11000000 01101101 01000001

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
```

3.3.1 Modifikacija preroka

Sprememba preroka je ta, da prerok ugiba več ($npr.m$) bitov ključa hkrati. To pomeni, da na vsakem koraku generiramo 2^m hipotetičnih ključev (smiselna omejitev števila m je 8), z njimi izvedemo m korakov odšifriranja in primerjamo porabe modela z izmerjenimi porabami in na podlagi razlik oz. koreliranosti izberemo m najbolj verjetnih ključev (ta parameter lahko spreminjamo, ampak za majhne dolžine prerokov je m kar primeren, več v nadaljevanju, pri algoritmu). Resno težavo namreč predstavljajo hipotetični ključi, ki se razlikujejo le v najbolj desnem bitu - vendar se napačnost izbire "podobnega" hipotetičnega ključa takoj pokaže ob uporabi naslednjih hipotetičnih bitov - porabe ne morejo biti več podobne pravi porabi in zato lahko take ključe hitro zavržemo. To je prikazano v nadaljevanju.

Zgled: Isti ključ kot prej, prerok ugiba 4 bite in meri porabo na 30 tekstih(na koncu so pravi ključ in štirje najbolj verjetni ključi) – poraba s pravim ključem je v 13. vrsti (kar pomeni 12. ključ oz. 1100 v dvojiškem):

```
4 16 9 16
4 16 9 64
4 16 9 81
4 16 9 49
4 0 1 16
4 0 1 16
4 0 81 81
4 0 81 361
0 0 0 16
0 0 0 36
0 0 64 81
0 0 64 289
0 0 0 0
0 0 0 100
0 0 64 169
0 0 64 169

11100110 11000000 10011111 10100010 00010110 00000011 01101010 00001100

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001100
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001001
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000100
```

Podobno isto kot prej, le da se meri razlike na 300 tekstih. Ta del je vključen predvsem, ker se lepo vidi, da se najbolj pravilni ključi razlikujejo v najmaj pomembnih bitih(porabe so zaporedoma po vrsticah).

```
676 841 1600 1681 / 676 841 1600 7569 / 676 841 1764 5041
676 841 1764 3721 / 676 1 841 1225    / 676 1 841 5041
676 1 9 441       / 676 1 9 441       / 0 484 289 576
0 484 289 3364    / 0 484 1369 196    / 0 484 1369 144
0 0 0 0           / 0 0 0 2304        / 0 0 576 2916
0 0 576 1764

11100110 11000000 10011111 10100010 00010110 00000011 01101010 00001100

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001100
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001101
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000110
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000111
```

Nato postopek nadaljujemo tako, da vzamemo 4 bite iz najbolj verjetnih ključev in ugibamo naslednje 4 (za naslednje korake je potem treba preživele ključe in znane bite še sestaviti skupaj) – tako je pravi ključ 15., kar pomeni 14 oz. 1110. Skupaj z biti od prej bi tako bili znani biti 1100 1110.

```
0 0 0 0 16 1 4 1     /  0 0 0 0 16 1 4 9     / 0 0 0 0 16 1 0 0
0 0 0 0 16 1 0 4     /  0 0 0 0 16 25 1 1    / 0 0 0 0 16 25 1 9
0 0 0 0 16 25 25 64  /  0 0 0 0 16 25 25 16  / 0 0 0 0 0 4 25 25
0 0 0 0 0 4 25 9     /  0 0 0 0 0 4 9 0      / 0 0 0 0 0 4 9 16
0 0 0 0 0 0 16 1     /  0 0 0 0 0 0 16 25    / 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4

01101100 00001001 11111010 00100001 01100000 00110110 10100000 11001110

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001110
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001111
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000010
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001010
```

Če pa vzamemo drugo najboljši ključ, vidimo, da ostaja napaka na porabi na že znanih bitih pa tudi najboljši ključi po modelu niso niti približno podobni pravim. (Spet je tu težava kratek prerok, načeloma za dolžino preroka 4 izberemo $\frac{1}{4}$ ključev, za dolžino 8 pa le še $\frac{1}{32}$ ključev.)

```
0 0 0 16 25 9 0 9     /  0 0 0 16 25 9 0 1     /  0 0 0 16 25 9 4 4
0 0 0 16 25 9 4 16    /  0 0 0 16 25 25 1 1    /  0 0 0 16 25 25 1 9
0 0 0 16 25 25 25 64  /  0 0 0 16 25 25 25 16  /  0 0 0 16 49 16 1 1
0 0 0 16 49 16 1 9    /  0 0 0 16 49 16 9 36   /  0 0 0 16 49 16 9 4
0 0 0 16 49 64 16 49  /  0 0 0 16 49 64 16 9   /  0 0 0 16 49 64 64 64
0 0 0 16 49 64 64 100

01101100 00001001 11111010 00100001 01100000 00110110 10100000 11001110

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000010
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000100
```

Za zglede sem uporabljal preroka, ki ugiba 4 bite, v praksi bi uporabil preroka, ki ugiba 8 bitov, vendar je zgled s takim prerokom praktično nemogoč, saj generira $2^8 = 256$ ključev, kar pa je bistveno preveč za zgled.

3.3.2 Algoritem in njegova računska zahtevnost

Algoritem, prerok generira 8 bitne ključe:

Vhod: vsota porab nekaj 10 šifriranj ter pripadajoči tajnopisi Ct
Izhod: tajni ključ(64b)

1. While
   (a) generiraj 2^8 ključev, z njimi izvedeš 8 korakov dešifriranja na tekstih, izračunaj porabe v modelu
   (b) izmed 2^8 ključev izbereš 8 tistih, ki so najbolj korelirani z izmerjeno porabo
   (c) za vsakega od zgornjih ključev naračunaj naslednjih 2^8 nadaljevanj ključa in izberi najboljših 8 izmed vseh $8 * 256 = 2^{11}$
   (d) to ponavljaj, dokler ne dobiš ključev dolžine 64b
   (e) za vse 64b ključe koreliraj porabo po modelu s pravo porabo na vseh 528 korakih in izberi najboljšega
2. vrni ključ K

Idealno bi bilo (iz stališča enostavnosti), če bi lahko na vsakem koraku razširili prostor ključev z vsemi hipotetičnimi in nato s temi ključi testirali porabo na 528 korakih (zakaj je to dobro, glej zgled naprej), vendar bi na ta način dobili $8^8 = 2^{24}$ ključev, kar pa je že "veliko- poleg tega pa narobe uganjeni biti ključa pomenijo tudi napake v tekstu in

zato pride v naslednjem koraku zanke s takim ključem do velikih sprememb v porabi in se da napačne ključe hitro določiti.

Časovna zahtevnost je pregled 2^{14} ključev, kar je enostavno za namizni računalnik.

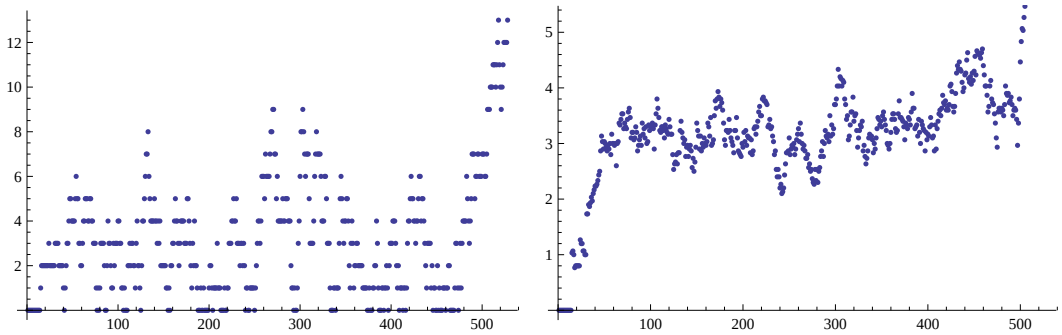
Zgled: Nadaljevanje zgleda iz 3.2, ugibamo 4 bite, vendar na prvem koraku vzamemo napačen ključ (vzamemo 1101 namesto 1100). (najprej naredimo odšifriranje z biti 1101 1110, poraba je na prvih štirih korakih ista kot v zgledu na začetku s hipotetičnim ključem 1101 (naslednji ključ od pravega)). Naslednje 4 bite zaradi enostavnosti zgleda nastavimo kar na pravilne ter nato še štiri bite z oraclom. Vidimo, da niti ena razlika porabe v modelu ni blizu 0.

```
0 0 0 100 196 324 361 361 324 121 36 81
0 0 0 100 196 324 361 361 324 121 36 49
0 0 0 100 196 324 361 361 324 121 64 25
0 0 0 100 196 324 361 361 324 121 64 225
0 0 0 100 196 324 361 361 324 361 121 121
0 0 0 100 196 324 361 361 324 361 121 225
0 0 0 100 196 324 361 361 324 361 361 361
0 0 0 100 196 324 361 361 324 361 361 529
0 0 0 100 196 324 361 361 324 289 144 289
0 0 0 100 196 324 361 361 324 289 144 121
0 0 0 100 196 324 361 361 324 289 196 81
0 0 0 100 196 324 361 361 324 289 196 529
0 0 0 100 196 324 361 361 324 169 25 9
0 0 0 100 196 324 361 361 324 169 25 121
0 0 0 100 196 324 361 361 324 169 169 225
0 0 0 100 196 324 361 361 324 169 169 225
11000000 10011111 10100010 00010110 00000011 01101010 00001100 11100110

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001100
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000010
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Nauk teh zgledov je, da se za napačno izbrane ključe v naslednjih korakih hitro pokaže, da so zares napačni. Torej "mora" modificirani DPA napad hitro delovati.

Zgled: Zakaj bi bilo "idealno" testirati kar najbolj verjetnih 2^{24} ključev. Zgled diferenc porabe v HD modelu za samo en napačen bit ključa na 528 korakih dešifriranja. Graf na levi kaže razliko med porabo s pravim ključem in ključem, kjer je bit na mestu 0 zamenjan. Graf na desni kaže povprečno razliko na 30 zaporednih tekstih – čistopise inkrementiramo na indeksu 5(code hopping, vseeno so tajnopisi neodvisni), nato zašifriramo ter na zašifriranih tekstih odšifriramo z danimi ključi. Ta zgled tudi pokaže, kako med vsemi generiranimi ključi (tistimi, ki "preživijo" algoritem do konca) ločimo pravega od napačnih.



3.3.3 Zaključek:

Motivacija je bila, zakaj lahko z varianto DPA napada na hardwersko implementirani KeeLoq hitro (v smislu malo sledi porabe) pridobimo tajni ključ – če seveda najdemo komponento vezja, ki izdaja Hammingovo razdaljo registra L. Razlog se skriva v sami enostavnosti šifre, kjer lahko razmeroma enostavno sestavimo preroka, ki ugiba več bitov ključa naenkrat, poleg tega pa že zelo hitro (recimo v naslednjih dveh ciklih ugibanja bitov, za $m = 8$ čez 16 bitov) lahko zavržemo večino predhodno narobe določenih ključev, kot se to vidi iz simulacij – enostavno ni nobene korelacije več med porabo s pravim in napačnim ključem.

Omeniti velja še naslednje: zaradi same sheme delovanja (angl. code hopping) in če ujamemo nekaj zaporednih tajnopisov, pravzaprav vemo, kdaj imamo pravi ključ. Zaporedni čistopisi se povečujejo za ena na petem bitu oz. razlika med dvema zaporednima čistopisoma v desetiškem sistemu je ravno 32 (0x20). Verjetno bi se dalo dokaj enostavno pokazati, da je v tej obliki to NP-poln problem (DPA pa nam pomaga izbrati rešitev).

4 Posledice uspešnega napada na Keeloq oz. kako klonirati oddajnik brez fizičnega dostopa

Ta razdelek predvsem povzema po članku Physical Cryptanalysis of KeeLoq Code Hopping Applications[3].

V razdelku je natančneje opisana shema kriptosistema ter kako so avtorji zgornjega članka izpeljali določene napade. Glavne ideje pa so zajete.

4.1 Shema kriptostistema

V praksi imamo en sprejemnik ter več oddajnikov. Poleg tega mora obstajati možnost dodajanja novih oddajnikov v sistem, radi pa bi tudi zagotovili, da oddajnik ne bi slučajno odprl nekega naključnega sprejemnika (pri brezžičnih napravah je to dosti bolj

pomembno, kot pri klasičnih ključavnicah).

Oddajnik vsebuje ključ proizvajalca (angl. manufacturer key) k_M, na podlagi tega ključa, ter serijskih številk oddajnika in sprejemnika izpelje ključ naprave (angl. device key) k_{Dev}, ki ga nato uporabljata specifičen oddajnik in sprejemnik (torej sprejemnik vodi seznam k_{Dev}).

Samo sporočilo, ki ga oddajnik pošlje sprejemniku je dolgo 32 bitov, od tega je navadno 10(12) bitov diskriminatorne vrednosti, 18(16) bitov števca ter 4 biti za ukaz. Sprejemnik dešifrira sporočilo s k_{Dev}, ter preveri diskriminatorno vrednost - če ta ni prava ne naredi ničesar. Nato preveri vrednost števca, če se vrednosti poslanega števca in v sprejemniku shranjenega števca razlikujeta dovolj malo (16 vrednosti ponavadi), izvrši ukaz. Sicer mora oddajnik še enkrat poslati sporočilo, če se to od prejšnjega razlikuje za dovolj malo, se ukaz izvrši. Oddajnik za vsako oddajanje poveča števec, prav tako sprejemnik vodi število uporabljenih ključev in že uporabljene ignorira (varnost pred replay napadi).

4.2 Kloniranje oddajnika

Pri tem napadu ne poznamo in ne dostopamo do sprejemnika, prav tako ne poznamo k_M. Za uspešen napad potrebujemo fizičen dostop do "veljavnega"oddajnika, in od 10 do 30 sledi (power traces). Poznati moramo tudi vsebino poslanih sporočil (tajnopise). Z DPA napadom pridobimo k_{Dev} in ko enkrat poznamo pravi ključ oddajnika, lahko tudi odšifriramo poslana sporočila. Če poznamo (znamo ločiti) med diskriminatornimi biti, števcem in ukazi, lahko nov (prazen) oddajnik napolnimo s pravimi vrednostmi (tj. nastavimo diskriminatorno vrednost ter števec). Tako sprejemnik ne loči več med tako kloniranim oddajnikom ter ostalimi oddajniki.

4.3 Sheme izpeljave ključev (angl. key derivation schemes)

V tem podrazdelku je opisana izpeljava ključev naprave k_{Dev} iz ključa proizvajalca. Te so pomembne za razumevanje napadov brez fizičnega dostopa.

V tipičnem KeeLoq sistemu sta uporabljena dva tipa ključev. Ključ oddajnika (device key) (k_{Dev}), ta je unikaten za vsak oddajnik in ga poznata oddajnik in sprejemnik. Ključ oddajnika se nastavi v "azi učenja" (angl. learning phase). Drugi tip ključa je ključ proizvajalca (k_M), ta je shranjen samo v sprejemniku - **in naj bi bil enak v vseh sprejemnikih danega proizvajalca** , njegova vloga je predvsem izpeljava ključev oddajnika. Oba tipa ključev sta dolga 64 bitov.

Obstaja več shem izpeljave ključev, opisal bom le najbolj enostavni dve. Ta izpeljava se izvede v sprejemniku.

1. Prva shema vzame serijsko številko oddajnika (p) - načeloma je to kar diskriminatorna vrednost, ter serijsko številko sprejemnika (q), nato izračuna $F_1(p)$ ter $F_2(q)$, ta dva rezultata sta dolga 32bitov. Sicer sta F_1 in F_2 načeloma neznani funkciji, vendar gre največkrat za enostavno dopolnitev do dolžine 32bitov (angl. padding). Nato se na vsakem rezultatu ločeno uporabi algoritem dešifriranja, ki uporabi kot ključ k_M ter oba rezultata združi (zaporedno doda) in dobimo 64 bitni rezultat, ki ga nato oddajnik in sprejemnik uporabljata kot k_{Dev}.

2. Ta shema deluje še bolj enostavno, zaporedno združi $F(p)$ in $F(q)$ v 64 bitov in to XOR-a s k_M.

3. Obstajajo tudi drugačne sheme, ki uporabljajo "naključno" seme (angl. seed), ter namesto na serijskih številkah uporabijo F_1 in F_2 na tem semenu in naprej podobno kot prejšnji dve. Mora pa priti do izmenjave semena na nek način, najverjetneje bi se to ugotovilo tako, da v fazi učenja pride do neke izmenjave od sprejemnika do oddajnika.

4.4 Pridobitev ključa proizvajalca k_M

Za uspešno pridobitev ključa proizvajalca k_M je treba "uganiti" katero shemo pridobivanja ključa kriptosistem uporablja. Tu predpostavimo, da imamo dostop do celotnega kriptosistema (oddajnika in sprejemnika).

V zgoraj opisanem primeru sheme (2) je postopek popolnoma enostaven, če poznamo ključ oddajnika (k_{Dev}). Spomnimo se: $k_{Dev} = k_M \oplus (F(p), F(q))$. Spomnimo se, da so enačbe z logičnim XOR enolično rešljive in tako $k_M = k_{Dev} \oplus (F(p), F(q))$.

Malo težje je, če se uporablja shema (1), ki zaporedoma dvakrat izvede dešifriranje na nizu 32 bitov, kjer za ključ uporabi kar ključ proizvajalca na $F(p)$ in nato na $F(q)$. Vendar, če ponovimo fazo učenja dovoljkrat, lahko z DPA napadom direktno pridobimo ključ proizvajalca k_M.

4.5 Kloniranje oddajnika brez fizičnega dostopa do kriptosistema

Predpostavka je, da poznamo k_M in shemo izpeljave ključa ter možnost, da zajamemo (tj. samo preberemo) dva tajnopisa neznanega oddajnika (iste vrste) sprejemniku, ki bi ga radi napadli. Označimo ta dva tajnopisa s c_1, c_2.

Spet ločimo možnosti glede na shemo pridobivanja ključa oddajnika:
Če je ključ oddajnika pridobljen kot $k_{Dev} = k_M \oplus (F(p), F(q))$ in ker odposlana sporočila vsebujejo serijske številke, lahko kar ponovimo proces izpeljave ključev in dobimo ključ oddajnika. Nato z njim odšifriramo eno od odposlanih sporočil in v klonu nastavimo

števec. Časovna zahtevnost takega napada je največ dve dekripciji KeeLoq-a.

V primeru sheme s semenom je postopek bolj zapleten in odvisen od velikosti semena, vendar v primeru semen velikosti 2^{32} dobimo pravo vrednost semena v manj kot treh urah z današnjim namiznim računalnikom. Dejansko gre kar za napad s surovo močjo, napadalec na dveh prestreženih sporočilih uporabi algoritem odšifriranja, za ključ uporabi k_M in to ponovi za vse možne vrednosti semena.

$$k_{Dev}^{(i)} = KeyDerivation(k_M, seed^{(i)})$$

$$(Counter_1^{(i)}, Disc_1^{(i)}) = D(c_1, k_{Dev}^{(i)})$$

$$(Counter_2^{(i)}, Disc_2^{(i)}) = D(c_2, k_{Dev}^{(i)})$$

Če imata tako dekodirani sporočili isto diskriminatorno vrednost ($Disc$) in sta števca blizu ($Counter$), tj. se razlikujeta za manj kot 16 (to je odvisno od tega, koliko sporočil je bilo poslanih med c_1 in c_2), potem smo zelo verjetno določili pravi k_Dev.
Obstajajo tudi večja semena, velikosti 2^{48} in 2^{60}. Manjša od teh dveh se tudi da poiskati z močnejšimi računalniki, večja pa so trenutno neizračunljiva s "prosto dostopnimi" računalniki.

Literatura

[1] G.V. Bard, (2009), *Algebraic Cryptanalysis*, Springer

[2] S. Mangard, E. Oswald (2007), T. Popp, *Power Analysis Attacks, Revealing the Secrets of Smart Cards*, Springer

[3] T. Eisenbarth in drugi (2008), *Physical Cryptanalysis of KeeLoq Code Hopping Applications*, IACR Eprint archive

[4] S. Indesteege in drugi (2008), *A Practical Attack on KeeLoq*, Advances in Cryptology - EUROCRYPT 2008