Математички факултет
Универзитет у Београду

Мастер рад

Тема: Криптоанализа алгоритма Oryx

СТУДЕНТ:
Дејан Капларевић

МЕНТОР:
проф. др Миодраг Живковић

ЧЛАНОВИ КОМИСИЈЕ:
проф. др Предраг Јаничић
проф. др Саша Малков

Београд, 2017.

Садржај

Глава 1

Увод

Тема рада је алгоритам шифровања Oryx и приказ једног могућег напада — спецификација алгоритама и програмске реализације у програмском језику C.

Oryx представља проточну шифру развијену од стране TIA (Telecommunications Industry Association), као део општег система безбедности комуникације мобилних телефона у Северној Америци.

У поглављу 2 пре преласка на објашњење и спецификацију алгоритма, следи неколико речи о безбедности мобилних телефона уопште, и начину на који се Oryx користи у пракси. Уводе се појмови који су неопходни за разумевање самог алгоритма шифровања и напада на алгоритам.

У поглављу 3, пре спецификације алгоритма Oryx, дефинише се линеарни померачки регистар (ЛПР) и наводе се неопходни појмови и чињенице о коначним пољима.

У поглављу 4 приказује се напад када криптоаналитичар има на располагању одређени број бајтова низа кључа, односно шифрат и одговарајући отворени текст. Описује се напад када је познато 25 бајтова низа кључа за шифровање, а наводе се и експериментални резултати о успешности напада када је познато мање од 25 бајтова низа кључа. Описани напад када се зна 25 бајтова низа кључа је програмски реализован на језику C. Предложена је модификација алгоритма за шифровање тако да буде отпоран на описани напад, али која би била осам пута спорија од оригиналног алгоритма. Размотрена је могућност напада када је на располагању само шифрат.

Глава 2

Заштита у систему GSM

Потражња за мобилним комуникационим системима је драматично порасла у последњих неколико година. Комуникација се врши преко радио везе, па је веома лако неприметно прислушкивање, тако да су криптографски алгоритми од велике важности да обезбеде заштиту приватности.

Прва генерација мобилних уређаја су били аналогни [1], и као такви веома ретко су користили алгоритме шифровања; чак и у случајевима да су коришћени неки видови шифровања, то је обезбеђивало веома низак ниво безбедности. Током последњих година су се појавили дигитални мобилни комуникациони системи, као што је Global Systems Mobile (GSM) међународни стандард развијен у Европи и Telecommunications Industry Association (TIA) стандарад развијен у Северној Америци.
GSM је најраспрострањенији међународни стандард за дигиталну комуникацију. Намењен је пре свега за пренос гласа и кратких текстуалних порука.

2.1 Алгоритми за шифровање

Када корисник жели да оствари комуникацију преко мобилног телефона, прво се врши аутентификација. Када је аутентификација извршена успешно, врши се пренос података шифрованих низом кључа, који се генерише једним од алгоритама:

- За GSM, користи се алгоритам A5 (верзије A5/1 и A5/2)
- За TIA, користи се алгоритам Oryx

Алгоритми Oryx и A5 (A5/1) користе линеране померачке регистре. За алгоритам Oryx су значајна три 32-битна линеарна померачка ре-

гистра, што значи да је за почетно стање потребно 96 бита. Када су регистри иницијализовани, сваким кораком алгоритма, два регистра изврше један корак, а трећи регистар изврши један или два корака, и на излазу се добија осам бита који представљају један бајт низа кључа.

За алгоритам A5/1 су такође значајна три линеарна померачка регистра, дужине 19, 22 и 23 бита, одакле следи да је за почетно стање потребно 64 бита. Када су регистри иницијализовани, првих 100 корака алгоритма не даје излаз, а затим следећих 228 корака алгоритма се понавља, и сваким кораком се на излазу добија по један бит. Приликом једног корака алгоритма се не гарантује да ће сваки од три регистра начинити корак.

У наставку се описује поступак аутентификације и шифровања. [1]

2.2 Успостављање везе

Нека је Алиса GSM корисник који жели да обави сигуран позив на свом GSM телефону. У њеном телефону се налази Subscriber Identity Module, познатији као SIM картица. SIM картица садржи приватни кључ Ki који једнозначно идентификује Алисин телефон. Тај кључ се такође налази и у приватној бази података у базној станици GSM. На Алисиној SIM картици се налази и хеш функција позната као A3.

Када Алисин телефон успоставља конекцију са GSM мрежом, телефон захтева 128-битни случајни број RAND од базне станице. Алисин телефон тада израчунава SRES = A3(RAND, Ki) и шаље га базној станици. Затим базна станица обавља исти прорачун и пореди резултат са вредношћу добијеном од Алисе. Уколико се бројеви слажу, Алиса је идентификована и може се наставити са комуникацијом. У противном Алиса добија поруку о грешци идентификације и веза се прекида.

Након аутентификације, Алисин телефон генерише 64-битни кључ Kc, који је израчунат користећи другу хеш функцију A8 са истим бројем RAND коришћеним у аутентификацији, Kc = A8(RAND, Ki).
Добијених 64 бита се уписују као почетно стање три регистра алгоритма A5, и затим се сав пренос података шифрује алгоритмом A5.
Базна станица такође обавља исте прорачуне и користећи кључ Kc и алгоритам A5 може да дешифрује податке које Алиса шаље.

Подаци који се размењују шаљу се у блоковима од по 228 бита, при чему се за сваки блок отвореног тескта, алгоритмом A5, генерише 228

[1] Како у расположивој литератури није пронађено објашњење о успостављању везе коришћењем Oryx-a, показано је како се то ради у системима који користе алгоритам A5

бита кључа. На битове отвореног текста и кључа примењује се операција сабирања по модулу 2, а затим се шифрована порука шаље кроз мрежу. На другој страни, базна станица ради исти поступак, и од примљеног шифрата и генерисаног кључа добија отворени текст који је Алиса послала.

2.3 GSM архитектура

Public Land Mobile Network (PLMN) је бежични комуникациони систем који обезбеђује услуге мобилних корисника. GSM мрежа је најпопуларнији пример PLMN-а.
GSM-PLMN мрежа је састављена из четири ентитета:

- Mobile Station (MS) је најчешће мобилни телефон, општије, било који уређај опремљен адекватном антеном и SIM картицом може се посматрати као мобилна станица.

- Base Station Subsystem (BSS) састоји се од мреже радио релеја који се називају Base Transceiver Station (BTS) и контролера названих Base Station Controllers (BSC). BSC се могу посматрати као интелигенција BTS-а, и обично имају од 10 до 100 BTS-а под својом контролом. BTS прима сигнале од мобилних станица и шаље их даље ка BSC. BTS се може посматрати као радио интерфејс GSM система.

- Network Switching Subsystem (NSS) је задужен за усмеравање говорних позива између два GSM корисника. Састоји се од специјализованих прекидача названих Mobile Switching Centers (MSC), који под својом контролом имају BSC. Сваки MSC је повезан са базом Visitor Location Register (VLR), чија је улога управљање информацијама корисника који су у оквиру домена који покрива одговарајући MSC. Ова база је повезана са централном базом Home Location Register (HLR) у оквиру које се налази Authentication Center (AuC) који је задужен за идентификацију корисника (допунске информације се налазе на корисниковој SIM картици). И на крају специфична MSC названа Gateway MSC (GMSC) која се налази на улазу у Public Switched Telephone Network (PSTN).

- Operation And Maintenance Center (OMC) је задужен за праћење исправности рада свих GSM система и преузимање одговарајућих мера за одржавање када је то потребно.

2.3.1 Сигурносни механизми

Главни сигурносни механизми обезбеђени од стране GSM-а налазе се лоцирани на BSC-у.

Под GSM сигурносним механизмима подразумевају се три класе заштите:

- Subscriber identity protection - у циљу заштите приватности, избегава се пренос идентитета корисника као отворени текст кроз радио везу. Примарни циљ је да се заштите информације о томе који корисник користи који мрежни ресурс. Секундарни циљ је да се спречи локализација и праћење одређене MS.

- Network access control - главна функционалност SIM картице је безбедно чување и управљање поверљивим информацијама како би се омогућило да GSM може тачно да идентификује корисника. Када се додаје нови уређај на мрежу, исти тајни кључ Ki се додаје у AuC и на SIM картицу новог уређаја. Сав сигурносни механизам се базира на кључу Ki, који се никада не шаље кроз мрежу.

- Radio communication encryption - подразумева шифровање свих порука које се шаљу између мобилних уређаја и BTS-а.

2.3.2 Могући напади

Описани безбедоносни механизми на GSM мрежи обезбеђују веома висок ниво заштите. Међутим како не постоји систем са 100% поузданости, увек се могу искористити неке мане система за могући напад.
Једна од мана која се може искористити за могући напад је клонирање SIM картице [6]. Ако нападач успе да клонира SIM картицу, а затим се претвара да је нови мобилни уређај, мрежа ће детектовати два мобилна уређаја са истим идентификаторима у исто време и угасиће оба, чиме се спречава крађа идентитета. Међутим, такви напади могу проћи неопажено ако је нападач заинтересован само за прислушкивање на радио комуникацијама. Заиста, уљез има приступ тајном кључу Ki, прима случајни број RAND и стога може генерисати Kc за пасивно дешифровање порука. Једно од решења које је предложено за ублажавање таквог напада се заснива на томе да се у SIM картицу додају одговарајући материјали као мера заштите од клонирања.

Још један од могућих напада се заснива на фалсификовању BTS-а и BSC-а. Ако нападач обезбеди да његова BTS има јачи сигнал него било

који други легитимни BTS у околини, мобилни уређај неће доводити под сумњу његову легитимност. Иако лажни BTS не може генерисати Kc, а тиме не може ни да дешифрује поруке од уређаја, он може једноставно да искључи шифровање, стога слободно може да прислушкује комуникацију која се преноси преко BTS-а.

Глава 3

Алгоритам Oryx

У овом поглављу уводе се неопходни појмови, као што су коначна поља [3], линеарни померачки регистар [3, 4], а затим се даје спецификација алгоритма Oryx [1, 2].

3.1 Коначна поља

Ако је p прост број, нека се са F_p означава поље са p елемената $\{0, 1, ..., p-1\}$ са операцијама $+$, $-$, $\cdot$, при чему се рачуна по модулу броја p. За елементе $\alpha \neq 0$ важи $\text{nzd}(\alpha, p) = 1$, па се може одредити α^{-1}. Због тога се може делити било којим елементом различитим од нуле.
У скупу $F_p^* = \{1, 2, ..., p-1\}$ се могу користити операције $\cdot$, $/$.
Неки елеменат $x \in F_p^*$ је генератор ако и само ако је његов ред (број различитих елемената у скупу $\{x, x^2, x^3, ...\}$) једнак $p-1$.
Специјално за $p = 2$ је $F_2 = \{0, 1\}$. Очигледно је $-1 = 1$, па је одузимање исто што и сабирање.

Нека је $F_2[x]$ скуп полинома са коефицијентима из $F_2 = \{0, 1\}$. Број полинома степена $\leq n$ је 2^{n+1}. То су полиноми $a_n x^n + ... + a_1 x + a_0$, $a_i \in \{0, 1\}$. Полиноми се множе и сабирају на уобичајени начин, при чему се са коефицијентима рачуна по модулу 2.
На пример:

$$(x^2 + x + 1)(x^2 + x) = x^3 + x^2 + x + x^4 + x^3 + x^2 = x^4 + x$$

Неки полином је несводљив над пољем ако се не може раставити у производ полинома (нижег степена) са коефицијентима из тог поља.

Над пољем F_2 полином $x^2 + 1 = (x+1)^2$ није несводљив, док полином $x^2 + x + 1$ није дељив ни једним полиномом првог степена па је несводљив (то је једини квадратни несводљиви полином).

Уопште, несводљиви полиноми добијају се од списка свих полинома прецртавањем умножака несводљивих полинома, слично као што се прости бројеви добијају применом Ератостеновог сита.

Нека се посматрају полиноми $F_2[x]$ и њихови остаци по модулу несводљивог полинома $x^3 + x + 1$. Очигледно је да се добијају полиноми мањег степена и важи $x^3 + x + 1 \equiv 0$ тј. $x^3 \equiv x + 1$. Према томе на скупу $F_2[x]/(x^3 + x + 1) = \{0, 1, x, x + 1, x^2, x^2 + 1, x^2 + x, x^2 + x + 1\}$ дефинисане су уобичајене операције + , - , $\cdot$. Ово поље означава се са F_8 јер има 8 елемената.

Уопште, ако је $p(x)$ несводљив полином степена d, онда је скуп $F_2[x]/(p(x))$ поље (остаци се рачунају по модулу полинома $p(x)$), које се означава са F_{2^d}; то поље има 2^d елемената. Ово поље састоји се од свих полинома степена $\leq d - 1$. $F^*_{2^d}$ је скуп од не-нула елемената поља F_{2^d}.
Полином $g(x) \in F^*_{2^d}$ је генератор ако и само ако је његов ред (број различитих елемената у скупу $\{g(x), g(x)^2, g(x)^3, ...\}$) једнак броју елемената у скупу $F^*_{2^d}$.
Полином x је генератор за F^*_8 из претходног примера:
$g = x$, x^2, $x^3 = x + 1$, $x^4 = x^2 + x$, $x^5 = x^4 \cdot x = x^3 + x^2 = x^2 + x + 1$, $x^6 = x^3 + x^2 + x = x^2 + 1$, $x^7 = x^3 + x = 1$.
Кажемо да је полином $g(x)$ примитиван ако је несводљив и x је генератор $F^*_{2^n} = F_2[x]/(g(x))^*$.

3.2 Линеарни померачки регистри

Линеарни померачки регистар (ЛПР) дужине n, је коначни аутомат са почетним стањем $(s_0, s_1, ..., s_{n-1})$, при чему $s_i \in \{0, 1\}$ за $i = 0, ..., n-1$. Ако се ЛПР налази у стању $(s_k, s_{k+1}, ..., s_{k+n-1})$, у наредном кораку прелази у стање $(s_{k+1}, s_{k+2}, ..., f(s_k, s_{k+1}, ..., s_{k+n-1}))$, и на излазу даје бит s_k. Функција која даје последњи бит наредног стања

$$f(s_0, ..., s_{n-1}) = b_0 s_0 \oplus ... \oplus b_{n-1} s_{n-1}$$

је одређена са n бита $b_0, ..., b_{n-1} \in \{0, 1\}$, $b_0 \neq 0$.

Начин рада ЛПР-а може се описати на следећи начин:
Нека су $c_0, c_1, ..., c_{n-1}$ ћелије регистра R. Када регистар изврши један корак, садржај ћелије c_{i+1} постаје садржај ћелије c_i $(i = n - 2, n - 3, ..., 0)$, излазни бит је садржај ћелије c_0, а нова вредност за садржај ћелије c_{n-1} се израчунава на основу израза

$$f(\overline{c_0}, \overline{c_1}, ..., \overline{c_{n-1}}) = b_0 \overline{c_0} \oplus b_1 \overline{c_1} \oplus ... \oplus b_{n-1} \overline{c_{n-1}}$$

где је $\overline{c_i}$ садржај ћелије c_i, а b_i су константе из скупа {0,1}.
За ЛПР дужине n постоји 2^n могућих почетних стања. За два стања се каже да су у истој орбити ако се из једног стања после неколико корака долази у друго стање.
Стање $(0, 0, ..., 0)$ има орбиту величине 1, тако да је најбоља могућа вредност за дужину периода $2^n - 1$.

Важно питање је када ЛПР има само две орбите, чије су величине 1 и $2^n - 1$? Нека је $g(x) = b_0 + b_1x^1 + b_2x^2 + ... + b_{n-1}x^{n-1} + x^n$, полином којим се описује ЛПР.

Теорема 1. Максимална орбита (величине $2^n - 1$) добија се ако и само ако је полином $g(x)$ примитиван по модулу 2 [4].

Доказ. Нека је полином $g(x) = b_0 + b_1x^1 + ... + b_{n-1}x^{n-1} + x^n$ несводљив, што значи да је $F_2[x]/g(x)$ поље.
Функција преласка у ново стање може се описати матрицом:

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_0 & b_1 & b_2 & \dots & b_{n-1} \end{bmatrix}$$

Очигледно је да важи: $s_{k+1} = A \cdot s_k$, где су s_k, s_{k+1} узастопна стања.

Нека је са $\Psi : F_2[x]/g(x)^* \to M_{n^2}$ дефинисано једно пресликавање [5] којим сваком полиному $p(x)$ додељујемо $p(A)$ (лако се види да је Ψ хомоморфизам и "1-1")

$$\Psi(p(x)) = p(A)$$

па важи:

$$\Psi(x) = A$$

$$\Psi(x^k) = A^k$$

Због тога су ред полинома x и матрице A исти (кажемо да је l ред матрице A ако важи $A^l = I$, где је I јединична матрица).

Са друге стране, ако за неко $k > 0$ важи да је $A^{k+1} = A$, следи да је:

$$A^{k+1} \cdot s_0 = A^k \cdot s_1 = ... = A^1 \cdot s_k = s_{k+1}$$

и

$$A^{k+1} \cdot s_0 = A \cdot s_0 = s_1$$

Одатле следи да је $s_{k+1} = s_1$, што значи да су редови матрице A и величина орбите ЛПР-а једнаки k. Може се закључити да су ред полинома x и величина орбите исти.

Ако је полином x генератор мултипликативне групе поља $F_2[x]/g(x)$, онда је његов ред $2^n - 1$, па следи да је величина орбите $2^n - 1$, што је максимална могућа величина орбите.

Са друге стране ако је величина орбите управо $2^n - 1$, следи да полином x има ред $2^n - 1$, а то је број елемената у коначном скупу $F_2[x]/g(x)^*$, што значи да овај полином може генерисати све елементе скупа $F_2[x]/g(x)^*$, па је он генератор.

□

Ако је број $2^n - 1$ прост, онда је довољно да $g(x)$ буде несводљив.

3.3 Спецификација алгоритма Oryx

Алгоритам Oryx се састоји од четири битне компоненте: три 32-битна ЛПР регистра (индекс излазног бита је 31), који се означавају са A, B и K, и табела L чије су вредности пермутација бројева од 0 до 255.

Полином за регистар K је:

$$k(x) = x^{32} + x^{28} + x^{19} + x^{18} + x^{16} + x^{14} + x^{11} + x^{10} + x^9 + x^6 + x^5 + x + 1$$

Кораци регистра A одређени су једним од два полинома:

$$a_1(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$$

$$a_2(x) = x^{32}+x^{27}+x^{26}+x^{25}+x^{24}+x^{23}+x^{22}+x^{17}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^7+x^2+x+1$$

Полином регистра B је:

$$b(x) = x^{32} + x^{31} + x^{21} + x^{20} + x^{16} + x^{15} + x^6 + x^3 + x + 1$$

Табела пермутације L је фиксна и иницијализује се на почетку; њен садржај приказан је у Табели 1 [2].

Кораци алгоритма Oryx су описани у наставку:

- K регистар извршава један корак, полиномом $k(x)$.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	ed	$3e$	$0d$	20	$a9$	$c3$	36	75	$4c$	$2c$	57	$a3$	00	ae	31	$0f$
1	19	$4d$	44	$a0$	11	56	18	66	09	69	$6e$	$3d$	25	$9c$	db	$3f$
2	65	58	$1a$	$6d$	ff	$d7$	46	$b3$	$b1$	$2b$	78	cf	be	26	42	$2f$
3	$d8$	$d4$	$8e$	48	05	$b9$	34	43	de	68	$5a$	aa	$9d$	bd	84	$a2$
4	$3c$	50	ce	$8b$	$c5$	$d0$	$a5$	77	$1f$	12	$6b$	$c2$	$b5$	$e6$	ab	54
5	81	22	$9f$	bb	$5c$	$a8$	dc	ec	$2d$	$1e$	ee	$d6$	$6c$	$5f$	$9a$	fd
6	$c8$	$d5$	94	fc	$0c$	$1c$	96	$4f$	$f9$	51	da	$9b$	df	$e1$	47	37
7	$d1$	eb	af	$f7$	$a4$	03	$f0$	$c7$	60	$e4$	$f4$	$b4$	85	$f6$	62	04
8	71	87	ea	17	99	$1d$	$3a$	15	52	$0a$	07	35	$e0$	70	$b6$	fa
9	cb	$b0$	86	$a6$	92	fb	98	55	06	$4b$	$5d$	$4a$	45	83	bf	16
a	$7c$	10	95	28	38	82	$f3$	$6a$	$f8$	fe	79	39	27	$2a$	$5e$	$e7$
b	59	$b8$	$1b$	ca	$8d$	$d3$	$7b$	30	33	90	$d2$	$d9$	ac	76	$8f$	$5b$
c	$a7$	$0e$	63	$c4$	$b2$	$e9$	97	91	53	$7a$	$0b$	41	08	$c1$	$8c$	$7d$
d	88	24	$f5$	$f2$	01	72	$e8$	80	49	13	23	$9e$	$c6$	14	73	ad
e	$8a$	29	ef	$e5$	67	61	ba	$e2$	$7e$	89	64	02	$c0$	21	$6f$	$f1$
f	dd	$b7$	$c9$	$e3$	cd	$3b$	93	$2e$	40	bc	$4e$	$a1$	cc	74	32	$7f$

Табела 3.1: Вредности табеле L

- Регистар A извршава један корак, користећи један од два полинома, у зависности од садржаја регистра K. Ако је 29-ти бит регистра K постављен на 0, користи се полином $a_1(x)$, иначе се користи полином $a_2(x)$.

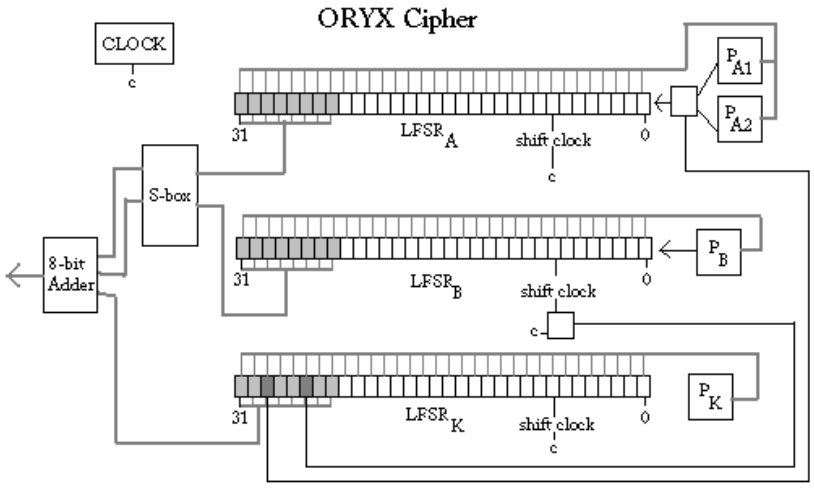
- Регистар B извршава један или два корака што такође зависи од садржаја регистра K. Ако је 26-ти бит регистра K постављен на 0, регистар B изврши један корак, иначе изврши два корака, полиномом $b(x)$.

- Највиших 8 бита (24 - 31) садржаја регистара K, A и B, се комбинују и формирају бајт кључа на основу израза:

$$bajtKljuca = \{High8_K + L[High8_A] + L[High8_B]\} \pmod{256} \quad (3.1)$$

  где се са $High8_R$ означава осмобитни број из највишег бајта регистра R.

Слика 1 илуструје кораке овог алгоритма.



Слика 3.1: Шематска представа алгоритма Oryx

Глава 4

Напад

Како је уобичајена претпоставка да нападач зна алгоритам, оно што му је потребно да би реконструисао низ кључа су почетни садржаји регистара K, A и B. Како ова три регистра имају по 32 бита, број могућих комбинација за почетно стање је 2^{96}.
Ако постоји могућност да нападач дође до неког дела низа кључа генерисаног алгоритмом Orux, тада се број комбинација почетних стања може знатно смањити. Један од таквих напада [1] приказан је у наставку.

4.1 Напад са познатим паром (Отворени текст, Шифрат)

Нека су нападачу познати парови бајтова (отворени текст, шифрат). Прецизније, познати су му парови бајтова (OT_i, ST_i) за $i \in \{1, 2, ..., n\}$, за неко n. Тада се бајтови низа кључа могу добити на следећи начин:

$$bajtKljuca_i = OT_i \oplus ST_i$$

Показује се да за $n = 25$, нападач на ефикасан начин може доћи до почетног стања регистара K, A и B.
Нека је $bajtKljuca(t)$ бајт низа кључа добијен у кораку t, $t = 1, 2, ...$ Даље, нека су виших 8 бита (24 - 31) регистара A, B и K, добијених у кораку t, сукцесивно, означени са $High8_A(t)$, $High8_B(t)$ и $High8_K(t)$.
Пре првог корака виших 8 бита (24 - 31) регистара A, B и K су сукцесивно означени са $High8_A(0)$, $High8_B(0)$ и $High8_K(0)$.

За $t = 1$ регистар A изврши један корак, при чему се добија $High8_A(1)$, регистар B изврши један или два корака при чему се добија $High8_B(1)$, и регистар K изврши један корак, при чему се добија $High8_K(1)$.

Следи да је бајт низа кључа добијен после првог корака Orux-a

$$bajtKljuca(1) = (L[High8_A(1)] + L[High8_B(1)] + High8_K(1)) \pmod{256}$$

Уопште важи да је:

$$bajtKljuca(t) = (L[High8_A(t)] + L[High8_B(t)] + High8_K(t)) \pmod{256}$$

За сваку претпоставку (од 2^{16} могућих претпоставки) у вези са вредношћу $High8_A(1)$ и $High8_B(1)$, вредност $High8_K(1)$ се може израчунати на основу израза:

$$High8_K(1) = (bajtKljuca(1) - L[High8_A(1) - L[High8_B(1)]) \pmod{256}$$

За $t = 2$, $High8_A(2)$ и $High8_K(2)$ ће имати по један непознати нови бит, а $High8_B(2)$ непознат један или два бита, у зависности да ли је регистар B извршио један или два корака, што зависи од бита на позицији 26 регистра K (познати бит).

За проширење $High8_A(2)$ новим битом постоје две могућности, за проширење $High8_K(2)$ новим битом такође две модућности, а за $High8_B(2)$ број могућности је два или четири, што даје укупан број могућности: $2 \cdot 2 \cdot 2 = 8$ или $2 \cdot 2 \cdot 4 = 16$, у просеку 12 могућности.

Даље се покушава са свим могућим стањима регистара (прецизније са свим могућим проширењима регистра за један или два бита), и по изразу (3.1), добијену вредност поредити са $bajtKljuca(2)$. Ако после испробавања свих могућих стања, није пронађено слагање, значи да је претпоставка за $High8_A(1)$ и $High8_B(1)$ погрешна и покушава се са другим вредностима. У супротном, ако постоји слагање, покушава се корак даље, и на тај начин се испитује свака грана у стаблу могућих варијанти и стања регистара. Тиме се за сваку од 2^{16} претпоставки формира једно стабло варијанти које се обилази алгоритмом претраге (backtracking).

Уопште, ако се стигне до неког корака t, покушава се са проширивањем непознатим битом регистара A и K, и са једним или два бита регистра B, и за свако проширивање регистара проверава да ли постоји слагање са $bajtKljuca(t + 1)$.
Ако постоји преклапање, прелази се на следећи корак, док у супротном, ако за сва могућа проширења није пронађено преклапање, враћа се корак назад и понавља процедура. Ако су после понављања процедуре испитане све могућности и налази се у корену стабла варијанти, онда се прелази на наредну могућу варијанту за 16 највиших битова регистара A и B.

После понављања описаног поступка 24 пута, што је очекивана висина стабла варијанти (у првом кораку напада, што је нулти корак у корену стабла варијанти, иницијализује се осам највиших бита регистара

A, B и K. У свакој следећој итерацији садржаји регистара A и K се проширују са по једним новим битом, прецизније, иницијализује се бит на позицији $24 - i$. Садржај регистра B се проширује са једним или са два нова бита у свакој итерацији), добија се почетно стање, тачније, стања регистара A, B и K после првог корака алгоритма, на основу којих се Огух-ом добија низ кључа. Свака итерација захтева по један бајт низа кључа, и један бајт је искоришћен за одређивање виших 8 бита (виших 8 бита регистара после првог корака). Одатле, са познатих 25 бајта кључа, иницијална стања регистара ће бити успешно реконструисана.

4.2 Перформансе напада

На почетку за сваки одабир $High8_A$ и $High8_B$, користи се $bajtKljuca(1)$ за одређивање $High8_K$. То значи да ако је познат само један бајт низа кључа, постоји $2^8 \cdot 2^8 = 65536$ могућности за највиших осам бита регистара A, B и K који дају $bajtKljuca(1)$. Ако су позната прва два бајта кључа, тада за сваку од 65536 могућности добијених у првом кораку, регистри A, B и K се могу проширити на 12 начина у просеку.

Свако исправно проширење се мора поклопити са 7 бита шифтованог садржаја $High8_K$. У просеку само један у 128 случајева из проширења "преживи", под претпоставком да се модел поређења бајтова посматра као случајан. Пошто је L пермутација, разумно је да се вредност обрачунава као случајан избор из скупа {0, 1, ..., 255}. Суштина је да се користе само два бајта низа кључа, одакле следи да очекивани број преосталих претпоставки од 65536 постаје:

$$\frac{12 \cdot 65536}{128} = 6144$$

.

Ако се настави са нападом укључујући сада и трећи бајт низа кључа, очекивани број преживелих могућности је:

$$\frac{12 \cdot 6144}{128} = 576$$

Укључујући и четврти бајт низа кључа, очекивани број преживелих могућности је:

$$\frac{12 \cdot 576}{128} = 54$$

Настављајући даље са петим бајтом низа кључа добија се

$$\frac{12 \cdot 54}{128} = 5.06$$

Ако се сада укључи и шести бајт низа кључа добија се

$$\frac{12 \cdot 5}{128} < 1$$

Дакле све претпоставке, осим оне праве, после највише 6 корака се одбацују [2].

Минимална дужина низа кључа потребна да би напад био успешан је 25 бајта, при чему први бајт даје осам бита за сваки од три 32-битна регистра, док сваки наредни бајт открива по један бит иницијалног стања ова три регистра.

У случају да је познато мање од 25 бајтова низа кључа, може се применити описани поступак над познатим бајтовима кључа, а за преостали део применити исцрпну претргу (покушавати са свим могућностима). Очигледно, да што је више бајтова кључа познато, смањује се број могућности за исцрпну претрагу и тиме ће напад бити успешнији.

Перформансе напада оцењиване су и експериментално [1], за низ бајтова дужине $N = 25$, 26 и 27. За сваку од ових дужина примењен је описани напад 1000 пута, при чему је за иницијализацију регистара A, B и K коришћен неки од генератора псеудо-случајних бројева.

У табели 2 су показани резултати овог експермента, при чему је за $N = 25$ напад у великој већини случаја успешан. У неким случајевима се дешава да број реконструисаних стања регистара добијених описаним нападом може бити и већи од 1, тако да нападач не може идентификовати право иницијално стање. У оваквим случајевима повећањем броја познатих бајтова кључа се елиминишу погрешна стања.

N	25	26	27
% Успеха	99.7	99.9	100.0

Табела 4.1: Проценат успеха у зависности од броја бајтова кључа

4.3 Повећање сигурности шифровања

Највећи проблем Oryx-a је у томе што се у свакој итерацији генерише по један бајт кључа. Ово доприноси ефикасности шифровања, док, са друге стране нападачу открива превише о унутрашњем стању.

Поставља се питање да ли се Oryx може учинити сигурнијим?
Ако се уместо сваког бајта на излазу, у једном кораку, алгоритам измени тако да излаз буде по један бит, може се значајно допринети сигурности.

На пример, нека се сваки бит рачуна на следећи начин :

$$bitKljuca_i = s(K) \oplus s(L[High8_A]) \oplus s(L[High8_B])$$

где s означава бит највеће тежине регистра или бајта [2].

Оваква идеја би онемогућила наведени напад, чак и са знатно већим бројем познатих бајтова кључа, у свакој итерацији број кандидата за разматрање би брзо растао, уместо да се тај број смањује.
Са друге стране оваква модификација алгоритма би процес шифровања успорила осам пута, што сигурно није практично за његову примену.

4.4 Напад када је познат само шифрат

У неким случајевима, напад са познатим паром (отворени текст, шифрат), може се проширити на напад са познатим шифратом[1] ако су позната нека знања о статистикама језика на коме је писан отворени текст. Ако је језик на коме је писан отворени текст енглески, може се искористити идеја да се почне са идентификовањем стрингова (речи или фраза), дужине неколико карактера, за које постоји највећа вероватноћа да се појаве у отвореном тексту. Неки примери стрингова, дужине $N = 7$, који се најчешће појављују су "login:␣ " и ".␣␣The␣". Покушава се са идентификовањем стринга највеће вероватноће на свакој позицији у шифрату. У сваком покушају примењује се описани напад, при чему ако описани алгоритам (примењен на низ бајтова кључа дужине N) ни по једној грани разматраних могућности не дође до краја, сматра се да се стринг не налази на датој позицији у отвореном тексту, у супротном се може претпоставити да је дати стринг идентификован у шифрату. Може се показати да је вероватноћа грешке описаног поступка веома мала, и да ако су идентификована два или три стринга у шифрату, уз коришћење познатих статистика језика на коме је писан отворени текст, на ефикасан начин се могу реконструисати садржаји регистара алгоритма Oryx.

4.5 Програмска реализација и резултати

У Прилогу 1 је програмска реализација шифровања (дешифровања) алгоритмом Oryx. Програм отвара датотеку "ulaz" у којој се налази порука коју је потребно шифровати, чита карактер по карактер, и за сваки прочитани карактер извршава један корак Oryx-a. Добијени бајт кључа се уписује у датотеку "keys.txt", док се шифровани карактер уписује у датотеку "izlaz". Програм на стандардни излаз исписује стања регистара

K, A и B, редом, после првог корака Oryx-a.

Дешифровање се врши аналогно, при чему сада као први аргумент командне линије програма треба навести датотеку са шифрованом поруком (датотека "izlaz"), а као други аргумент назив датотеке у коју ће се уписати дешифрована порука.

У Прилогу 2 је програмска реализација напада. Програм отвара датотеку "ulaz" (отворени текст) и датотеку "izlaz" (шифрат), чита првих 25 карактера, и применом операције $\oplus$ добија првих 25 бајтова кључа, које затим даље користи за откривање иницијалних стања регистара, тачније стања после првог корака Oryx-a, и добијено решење исписује на стандардни излаз.

Програми су преведени командама:

```
gcc -Wall -pedantic -ansi -o oryx oryx.c
gcc -Wall -pedantic -ansi -o attac attac.c
```

Садржај датотеке "ulaz":

```
Primer poruke koja se sifruje Oryx-om
```

После покретања ./oryx излаз је:

```
beb7fab2 56dfa6b1 cffed274
```

Садржај датотеке "keys.txt":

```
0x17 0x49 0xFFFFFFD4 0xFFFFFFEF 0x4B 0xFFFFFFA6 0x7E 0x25
0xFFFFFF87 0x1B 0x79 0xFFFFFFD9 0x78 0x26 0x5 0x6B 0x3D
0xFFFFFF84 0xFFFFFF93 0xFFFFFFF3 0xFFFFFF82 0x2B
0xFFFFFFC0 0x73 0x57 0xFFFFFFE5 0xFFFFFFEF 0xFFFFFFA4
0x1F 0x4 0xFFFFFFF5 0x5D 0x72 0x13 0x23 0x9 0x1 0xFFFFFF88
```

После покретања ./attac излаз је:

```
beb7fab2 56dfa6b1 cffed274
```

Глава 5

Закључак

Анализом алгоритма Oryx, показано је како се може извршити један напад, знатно ефикаснији од напада грубом силом. Напад се заснива на претпоставци да је нападачу познат мали број шифрата и одговарајућих бајтова отвореног текста.
У програмској реализацији напада се види да је довољно 25 бајтова кључа да би напад био успешан, прецизније, експериментални резултати показују да је проценат успеха са познатих 25 бајтова кључа 99%, а са 27 бајтова тај процента је 100%.

Даљи кораци рада би могли да буду у вези са разрадом напада када је познат само шифрат, при чему је потребно дефинисати скуп стрингова највеће вероватноће да се појаве у отвореном тексту. Такође је потребно проучити и статистике језика на коме је писан отворени текст.

Библиографија

[1] D. Wagner, L. Simpson, E. Dawson, J. Kelsey, W. Millan , and B. Schneier, *Cryptanalysis of ORYX*, SAC '98 Proceedings of the Selected Areas in Cryptography, Springer-Verlag, 1998. str. 296-305

[2] Mark Stamp, Richard M. Low, *APPLLIED CRYPTANALYSIS, Breaking Ciphers in the Real World*, A JOHN WILEY & SONS, INC., PUBLICATION 2007.

[3] М. Живковић, *Криптографија,* скрипта, 2012.

[4] Solomon W. Golomb, *SHIFT REGISTER SEQUENCES*, 3. izdanje, World Scientific, 2017.

[5] Mark Goresky, Andrew Klapper, *ALGEBRAIC SHIFT REGISTER SEQUENCES*, Cambridge University Press, 2012.

[6] Hakima Chaouchi, Maryline Laurent-Maknavicius, *Wireless and Mobile Network Security*, Wiley, 2009.

Глава 6

Прилог 1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Rigistri */
5 unsigned int K = 0xDF5BFD59;
6 unsigned int A = 0xAB6FD358;
7 unsigned int B = 0xF3FFB49D;
8
9 /* Tabela permutacije */
10 unsigned char L[] = {
11 0xed, 0x3e, 0x0d, 0x20, 0xa9, 0xc3, 0x36, 0x75, 0x4c,
12 0x2c, 0x57, 0xa3, 0x00, 0xae, 0x31, 0x0f, 0x19, 0x4d,
13 0x44, 0xa0, 0x11, 0x56, 0x18, 0x66, 0x09, 0x69, 0x6e,
14 0x3d, 0x25, 0x9c, 0xdb, 0x3f, 0x65, 0x58, 0x1a, 0x6d,
15 0xff, 0xd7, 0x46, 0xb3, 0xb1, 0x2b, 0x78, 0xcf, 0xbe,
16 0x26, 0x42, 0x2f, 0xd8, 0xd4, 0x8e, 0x48, 0x05, 0xb9,
17 0x34, 0x43, 0xde, 0x68, 0x5a, 0xaa, 0x9d, 0xbd, 0x84,
18 0xa2, 0x3c, 0x50, 0xce, 0x8b, 0xc5, 0xd0, 0xa5, 0x77,
19 0x1f, 0x12, 0x6b, 0xc2, 0xb5, 0xe6, 0xab, 0x54, 0x81,
20 0x22, 0x9f, 0xbb, 0x5c, 0xa8, 0xdc, 0xec, 0x2d, 0x1e,
21 0xee, 0xd6, 0x6c, 0x5f, 0x9a, 0xfd, 0xc8, 0xd5, 0x94,
22 0xfc, 0x0c, 0x1c, 0x96, 0x4f, 0xf9, 0x51, 0xda, 0x9b,
23 0xdf, 0xe1, 0x47, 0x37, 0xd1, 0xeb, 0xaf, 0xf7, 0xa4,
24 0x03, 0xf0, 0xc7, 0x60, 0xe4, 0xf4, 0xb4, 0x85, 0xf6,
25 0x62, 0x04, 0x71, 0x87, 0xea, 0x17, 0x99, 0x1d, 0x3a,
26 0x15, 0x52, 0x0a, 0x07, 0x35, 0xe0, 0x70, 0xb6, 0xfa,
27 0xcb, 0xb0, 0x86, 0xa6, 0x92, 0xfb, 0x98, 0x55, 0x06,
```

```
28 0x4b, 0x5d, 0x4a, 0x45, 0x83, 0xbf, 0x16, 0x7c, 0x10,
29 0x95, 0x28, 0x38, 0x82, 0xf3, 0x6a, 0xf8, 0xfe, 0x79,
30 0x39, 0x27, 0x2a, 0x5e, 0xe7, 0x59, 0xb8, 0x1b, 0xca,
31 0x8d, 0xd3, 0x7b, 0x30, 0x33, 0x90, 0xd2, 0xd9, 0xac,
32 0x76, 0x8f, 0x5b, 0xa7, 0x0e, 0x63, 0xc4, 0xb2, 0xe9,
33 0x97, 0x91, 0x53, 0x7a, 0x0b, 0x41, 0x08, 0xc1, 0x8c,
34 0x7d, 0x88, 0x24, 0xf5, 0xf2, 0x01, 0x72, 0xe8, 0x80,
35 0x49, 0x13, 0x23, 0x9e, 0xc6, 0x14, 0x73, 0xad, 0x8a,
36 0x29, 0xef, 0xe5, 0x67, 0x61, 0xba, 0xe2, 0x7e, 0x89,
37 0x64, 0x02, 0xc0, 0x21, 0x6f, 0xf1, 0xdd, 0xb7, 0xc9,
38 0xe3, 0xcd, 0x3b, 0x93, 0x2e, 0x40, 0xbc, 0x4e, 0xa1,
39 0xcc, 0x74, 0x32, 0x7f
40 };
41
42 /* Polinomi predstavljeni maskama */
43 unsigned int mk = 0x100D4E63;
44 unsigned int ma1 = 0x04C11DB7;
45 unsigned int ma2 = 0x0FC22F87;
46 unsigned int mb = 0x8031804B;
47
48 FILE *keys;
49
50 unsigned char hight_byte(unsigned int);
51 int number_of_ones(int);
52 unsigned int shift_K(unsigned int);
53 unsigned int shift_A(unsigned int, unsigned int);
54 unsigned int shift_B(unsigned int, unsigned int);
55 char step();
56 void encode_file(char* in, char* out);
57 char encode_one_char(char);
58
59 int
60 main(int argc, char** argv)
61 {
62   /* Fajl u koji se upisuje niz kljuceva  */
63   keys = fopen("keys.txt", "w");
64   if(keys == NULL) {
65         fprintf(stderr,"error fopen(): Neuspelo
            otvoranje datoteke keys.txt za pisanje.\n");
66         exit(EXIT_FAILURE);
67   }
```

```
68
69    /* Prvi arg komandne linije je fajl teksta koji se
         sifruje, a drugi arg je fajl sa sifrovanim
         tekstom */
70    if(argc == 3) {
71      encode_file(argv[1], argv[2]);
72    } else {
73      encode_file("ulaz", "izlaz");
74    }
75
76    fclose(keys);
77    return 0;
78 }
79
80 /* Vraca se visih 8 bita broja n */
81 unsigned char
82 high_byte(unsigned int n)
83 {
84    return (n >> 24) & 0xFF;
85 }
86
87 /* Broj jedinica u binarnom zapisu broja */
88 int
89 number_of_ones(int n)
90 {
91    int count = 0;
92    while(n > 0) {
93      count++;
94      n = n & (n - 1);
95    }
96    return count;
97 }
98
99 /* Siftuje se registar K */
100 unsigned int
101 shift_K(unsigned int K)
102 {
103    unsigned int y;
104
105    y = number_of_ones(K & mk) % 2  == 1 ? 1 : 0;
106    K = (K << 1) | y;
```

```
107
108   return K & 0xFFFFFFFF;
109 }
110
111 /* Siftuje se registar A */
112 unsigned int
113 shift_A(unsigned int A, unsigned int K)
114 {
115   unsigned int y;
116   unsigned int a_29 = 0x20000000 & K; /* 29-ti bit */
117
118   if(a_29 == 0) {
119     y = number_of_ones(A & ma1) % 2  == 1 ? 1 : 0;
120   } else {
121     y = number_of_ones(A & ma2) % 2  == 1 ? 1 : 0;
122   }
123
124   A = (A << 1) | y;
125   return A & 0xFFFFFFFF;
126 }
127
128 /* Siftuje se registar B */
129 unsigned int
130 shift_B(unsigned int B, unsigned int K)
131 {
132   unsigned int y;
133   unsigned int b_26 = 0x04000000 & K; /* 26-ti bit */
134
135   y = number_of_ones(B & mb) % 2  == 1 ? 1 : 0;
136   B = (B << 1) | y;
137   B = B & 0xFFFFFFFF;
138
139   if(b_26 != 0) {
140     y = number_of_ones(B & mb) % 2  == 1 ? 1 : 0;
141     B = (B << 1) | y;
142   }
143
144   return B & 0xFFFFFFFF;
145 }
146
147 /* Jedan korak Oryx - a */
```

```
148 int prvi_korak = 0;
149 char
150 step()
151 {
152   unsigned int a, b, k;
153   char key;
154
155   K = shift_K(K);
156   A = shift_A(A, K);
157   B = shift_B(B, K);
158
159   a = high_byte(A);
160   b = high_byte(B);
161   k = high_byte(K);
162
163   key = (L[a] + L[b] + k) & 0xFF; /* deljenje po
          modulu 256 */
164
165   if(prvi_korak == 0) {
166     printf("%x %x %x\n", K, A, B);
167     prvi_korak++;
168   }
169
170   fprintf(keys, "0x%X ", key);
171   return key;
172 }
173
174 /* ulazni karakter se kodira ORYX - om  */
175 char
176 encode_one_char(char in)
177 {
178   return in ^ step();
179 }
180
181 /* ulazni fajl 'in' se kodira ORYX - om i upisuje u
       fajl 'out' */
182 void
183 encode_file(char* in, char* out)
184 {
185    int c;
186    FILE *ulaz, *izlaz;
```

```
187
188     ulaz = fopen(in, "r");
189     if(ulaz == NULL) {
190         fprintf(stderr,"error fopen(): Neuspelo
              otvoranje datoteke ulaz.txt za citanje.\n");
191         exit(EXIT_FAILURE);
192     }
193
194     izlaz = fopen(out, "w");
195     if(izlaz == NULL) {
196         fprintf(stderr,"error fopen(): Neuspelo
              otvoranje datoteke izlaz.txt za pisanje.\n");
197         exit(EXIT_FAILURE);
198     }
199
200
201     while((c = fgetc(ulaz)) != EOF) {
202    fputc(encode_one_char(c), izlaz);
203       }
204
205     fclose(ulaz);
206     fclose(izlaz);
207 }
```

oryx.c

Глава 7

Прилог 2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Tabela permutacije */
5 unsigned char L[] = {
6 0xed, 0x3e, 0x0d, 0x20, 0xa9, 0xc3, 0x36, 0x75, 0x4c,
7 0x2c, 0x57, 0xa3, 0x00, 0xae, 0x31, 0x0f, 0x19, 0x4d,
8 0x44, 0xa0, 0x11, 0x56, 0x18, 0x66, 0x09, 0x69, 0x6e,
9 0x3d, 0x25, 0x9c, 0xdb, 0x3f, 0x65, 0x58, 0x1a, 0x6d,
10 0xff, 0xd7, 0x46, 0xb3, 0xb1, 0x2b, 0x78, 0xcf, 0xbe,
11 0x26, 0x42, 0x2f, 0xd8, 0xd4, 0x8e, 0x48, 0x05, 0xb9,
12 0x34, 0x43, 0xde, 0x68, 0x5a, 0xaa, 0x9d, 0xbd, 0x84,
13 0xa2, 0x3c, 0x50, 0xce, 0x8b, 0xc5, 0xd0, 0xa5, 0x77,
14 0x1f, 0x12, 0x6b, 0xc2, 0xb5, 0xe6, 0xab, 0x54, 0x81,
15 0x22, 0x9f, 0xbb, 0x5c, 0xa8, 0xdc, 0xec, 0x2d, 0x1e,
16 0xee, 0xd6, 0x6c, 0x5f, 0x9a, 0xfd, 0xc8, 0xd5, 0x94,
17 0xfc, 0x0c, 0x1c, 0x96, 0x4f, 0xf9, 0x51, 0xda, 0x9b,
18 0xdf, 0xe1, 0x47, 0x37, 0xd1, 0xeb, 0xaf, 0xf7, 0xa4,
19 0x03, 0xf0, 0xc7, 0x60, 0xe4, 0xf4, 0xb4, 0x85, 0xf6,
20 0x62, 0x04, 0x71, 0x87, 0xea, 0x17, 0x99, 0x1d, 0x3a,
21 0x15, 0x52, 0x0a, 0x07, 0x35, 0xe0, 0x70, 0xb6, 0xfa,
22 0xcb, 0xb0, 0x86, 0xa6, 0x92, 0xfb, 0x98, 0x55, 0x06,
23 0x4b, 0x5d, 0x4a, 0x45, 0x83, 0xbf, 0x16, 0x7c, 0x10,
24 0x95, 0x28, 0x38, 0x82, 0xf3, 0x6a, 0xf8, 0xfe, 0x79,
25 0x39, 0x27, 0x2a, 0x5e, 0xe7, 0x59, 0xb8, 0x1b, 0xca,
26 0x8d, 0xd3, 0x7b, 0x30, 0x33, 0x90, 0xd2, 0xd9, 0xac,
27 0x76, 0x8f, 0x5b, 0xa7, 0x0e, 0x63, 0xc4, 0xb2, 0xe9,
```

```
28 0x97, 0x91, 0x53, 0x7a, 0x0b, 0x41, 0x08, 0xc1, 0x8c,
29 0x7d, 0x88, 0x24, 0xf5, 0xf2, 0x01, 0x72, 0xe8, 0x80,
30 0x49, 0x13, 0x23, 0x9e, 0xc6, 0x14, 0x73, 0xad, 0x8a,
31 0x29, 0xef, 0xe5, 0x67, 0x61, 0xba, 0xe2, 0x7e, 0x89,
32 0x64, 0x02, 0xc0, 0x21, 0x6f, 0xf1, 0xdd, 0xb7, 0xc9,
33 0xe3, 0xcd, 0x3b, 0x93, 0x2e, 0x40, 0xbc, 0x4e, 0xa1,
34 0xcc, 0x74, 0x32, 0x7f
35 };
36
37 /* Niz kljuca */
38 unsigned char keys[128];
39 unsigned char ot[128];
40 unsigned char si[128];
41
42 void attac();
43 unsigned char determine_x(unsigned char, unsigned
       char);
44 unsigned char shift_A(unsigned char, unsigned int);
45 unsigned char shift_B(unsigned char, unsigned int,
       unsigned int);
46 unsigned char shift_K(unsigned char, unsigned int);
47 int compare(unsigned char, unsigned char, unsigned
       char, int);
48 void attempt(unsigned char, unsigned char, unsigned
       char, int, unsigned int, unsigned int, unsigned
       int, int);
49 void fill_array_keys(char*, char*);
50
51 int
52 main()
53 {
54   fill_array_keys("ulaz", "izlaz");
55   attac();
56   return 0;
57 }
58
59 /* Cita se bajt po bajt otvorenog teksta i sifrata i
       dobijeni bajt kljuca upisuje u niz keys[] */
60 void
61 fill_array_keys(char* in, char* out) {
62
```

```
63     int i;
64     FILE *ulaz, *izlaz;
65
66     ulaz = fopen(in,"r");
67     if(ulaz == NULL) {
68         fprintf(stderr,"error fopen(): Neuspelo
               otvoranje datoteke ulaz.txt za citanje.\n");
69         exit(EXIT_FAILURE);
70     }
71
72     izlaz = fopen(out, "r");
73     if(izlaz == NULL) {
74         fprintf(stderr,"error fopen(): Neuspelo
               otvoranje datoteke izlaz.txt za pisanje.\n");
75         exit(EXIT_FAILURE);
76     }
77
78     i = 0;
79     while((ot[i] = fgetc(ulaz)) != EOF && (si[i] =
           fgetc(izlaz)) != EOF && i < 25) {
80      keys[i] = ot[i] ^ si[i];
81      i++;
82      }
83
84     fclose(ulaz);
85     fclose(izlaz);
86 }
87
88 /* Pokusava se sa svim mogucim vrednostima za najvisih
     8 bita registara A i B,
89     zatim na osnovu tih vrednosti i prvog bajta kljuca
           odredjuje najvisih 8 bita registra K,
90     i pokusava korak dublje sa pogadjanjem
91  */
92 void attac()
93 {
94   unsigned char a, b;
95   unsigned char x;
96
97   /* pokusava se sa svim mogucim vrednostima za
          najvisih 8 bita registara A i B */
```

```
98    for(a = 0; a <= 255; a++) {
99      for(b = 0; b <= 255; b++) {
100       /* odredjuje se najvisih 8 bita registra K */
101       x = determine_x(a, b);
102
103       /* Pokusava se korak dublje */
104       attempt(a, b, x, 1, a << 24, b << 24, x << 24,
              8);
105
106       if(b == 255) {
107         break;
108       }
109     }
110
111     if(a == 255) {
112       break;
113     }
114   }
115 }
116
117 /* K = (k0 - L[a] - L[b]) % 256 */
118 unsigned
119 char determine_x(unsigned char a, unsigned char b)
120 {
121   unsigned char r = (keys[0] - L[a] - L[b]) & 0xFF;
122   return r;
123 }
124
125 /* Siftuje se registar A ulevo za 1 poziciju, i dodaje
      novi bit a sa desne strane */
126 unsigned
127 char shift_A(unsigned char A, unsigned int a)
128 {
129   return ((A << 1) | (a & 0x1)) & 0xFF;
130 }
131
132 /* Siftuje se registar B ulevo za 1 ili 2 pozicije u
      zavisnosti od bita x_26,
133    i dodaje 1 ili 2 nova bita iz b */
134 unsigned
```

```
135 char shift_B(unsigned char B, unsigned int b, unsigned
      int x_26)
136 {
137   if(x_26 == 0) {
138     return ((B << 1) | (b & 0x1)) & 0xFF;  /* b je 0,
          1 */
139   } else {
140     return ((B << 2) | (b & 0x3)) & 0xFF;  /* b je 00,
          01, 10, 11 */
141   }
142 }
143 /* Siftuje se registar K ulevo za 1 poziciju, i dodaje
      novi bit k sa desne strane */
144 unsigned
145 char shift_K(unsigned char K, unsigned int k)
146 {
147   return ((K << 1) | (k & 0x1)) & 0xFF;
148 }
149
150 /* Provera da li je k[i] == (L[a] + L[b] + K) % 256 */
151 int
152 compare(unsigned char A, unsigned char B, unsigned
      char K, int i)
153 {
154   unsigned char tmp = (L[A] + L[B] + K) & 0xFF;
155   if (keys[i] == tmp) {
156     return 1;
157   } else {
158     return 0;
159   }
160
161 }
162
163 /*
164   Parametri:
165   A - najvisih 8 bita registra A
166   B - najvisih 8 bita registra B
167   K - najvisih 8 bita registra K
168   i - korak 'dubine'
169   AR - 32 bitni registar A, tj. njegov sadrzaj posle
        prvog koraka oryx-a, koji se 'rekonstruise'
```

```
170   BR - 32 bitni registar B, tj. njegov sadrzaj posle
          prvog koraka oryx-a, koji se 'rekonstruise'
171   KR - 32 bitni registar K, tj. njegov sadrzaj posle
          prvog koraka oryx-a, koji se 'rekonstruise'
172   q - broj bita koji su upisani (rekonstruisani) u
          registru BR, ovaj podatak je potreban
173   zbog toga sto se registar B siftuje 1 ili 2 puta po
          iteraciji
174 */
175
176 void
177 attempt(unsigned char A, unsigned char B, unsigned
      char K, int i, unsigned int AR, unsigned int BR,
      unsigned int KR, int q)
178 {
179   unsigned char a, b, k, at, bt, kt;
180   unsigned int x_26;
181   unsigned int b_mogucnosti;
182   int t;
183
184   if(i == 25) {
185     /* izlaz iz rekurzije, pronadjeno resenje */
186     printf("%x %x %x\n", KR, AR, BR);
187     return;
188   }
189
190   /* 26 bit iz K odredjuje da li se B siftuje 1 ili 2
          puta,
191       ali taj bit pre siftovanja K registra se nalazi
              na poziciji 25 */
192   x_26  = (K >> 1) & 0x1;
193   b_mogucnosti = x_26 == 0 ? 2 : 4;
194
195   /* pokusava se sa mogucnostima za novi bit(bitove)
          pri siftovanju registara */
196   for(a = 0; a < 2; a++) {
197     for(b = 0; b < b_mogucnosti; b++) {
198       for(k = 0; k < 2; k++) {
199         kt = shift_K(K, k);
200         at = shift_A(A, a);
201         bt = shift_B(B, b, x_26);
```

```
202
203             t = compare(at, bt, kt, i);
204
205             /* ako ima preklapanja, na dobrom je putu,
                   pokusava se korak dublje */
206             if(t == 1) {
207               /* azuriraju se 32-bitni registri A, B i K
                     */
208               unsigned int ARt = AR | ((a & 0x1) << (24
                     - i));
209               unsigned int KRt = KR | ((k & 0x1) << (24
                     - i));
210               unsigned int BRt = 0;
211               int qt = q;
212
213               /* Da li se B siftovao 1 ili 2 puta */
214               if(x_26 == 0) {
215                 if(q < 32) {
216                   BRt = BR | ((b & 0x1) << (31 - q));
217                 } else {
218                   BRt = BR; /* ako je registar vec
                                  popunjen ne menja se */
219                 }
220               } else {
221                 if(q < 31) {
222                   BRt = BR | ((b & 0x3) << (30 - q));
223                 } else if(q == 31) {
224                   BRt = BR | ((b & 0x3) >> 1);
225                 } else {
226                   BRt = BR; /* ako je registar vec
                                  popunjen ne menja se */
227                 }
228                 qt++;
229               }
230               qt++;
231               /* pokusava se korak dublje */
232                     attempt(at, bt, kt, i + 1, ARt, BRt,
                           KRt, qt);
233
234             }
235           }
```

```
236             }
237         }
238 }
```

attac.c