Универзитет у Београду

Математички факултет

АНАЛИЗА АЛГОРИТМА ЗА ШИФРОВАЊЕ У ПРОГРАМУ PKZIP

— Мастер рад —

аутор: Зоран Ташић

ментор: др Миодраг Живковић

Београд
2009.

Садржај

Предговор

PKZIP је програм за компресију података. ZIP формат је један од општеприхваћених стандарда за архивирање и компресију података и веома је распрострањен. Поред компресије, програм PKZIP омогућује и шифровање података. Алгоритам шифровања зависности од верзије програма.

Предмет овог рада је алгоритам шифровања са интерним кључем од 96 бита уведен у верзији 2.04 програма PKZIP. Без обзира што дужина кључа обезбеђује довољну заштиту од напада грубом силом, испоставља се да слабост алгоритма шифровања омогућује напад са познатим паром (отворени текст, шифрат). Ако је на располагању 13 узастопних бајтова отвореног текста и одговарајући бајтови шифрата, напад се своди на проверу око 2^{38} уместо 2^{96} варијанти [2]. Ако се на неки начин дође до било којих узастопних 13 бајтова отвореног текста, може се одредити интерни кључ, лозинка којом је шифрован фајл, као и комплетан садржај фајла.

У уводу је дат кратак преглед развоја програма PKZIP и опис неких метода напада на PKZIP. У другом поглављу дат је опис структуре .ZIP фајла, приказ примењених метода компресије и опис алгоритма за шифровање развијеног специјално за примену у програму PKZIP. У трећем поглављу приказан је поступак одређивања интерног кључа и описан је алгоритам за сам напад. У наредном поглављу описан је програм за одређивање интерног кључа, односно лозинке. Тај програм је саставни део овог рада. На крају се наводе нека могућа усавршавања развијеног програма.

Поглавље 1

Увод

Програм PKZIP

PKZIP је програм за компресију и архивирање фајлова. Првобитну верзију програма написао је Фил Кац (Phil Katz[1]), а продавала га је компанија PKWARE, Inc. PKZIP је скраћеница за **P**hil **K**atz's **ZIP** program.

Историја

Програми за компресију података су се појавили крајем 60-их година двадесетог века[2]. Мада су вероватно програми тог типа постојали и раније, први познати програм је SQUOZE, направљен од стране IBM-а. То је био програм (као део SHARE оперативног система) који је служио да пакује податке на великим (mainframe) рачунарима типа 709 и 7090.

70-их година програми за архивирање били су стандардни сервис оперативних система. Они су, пре свега, развијени са циљем да више одвојених фајлова споје у један фајл, ради лакшег копирања и преношења. То су били нпр. стандардни UNIX сервиси ar, shar и tar.

Раних 80-их година, компанија System Enhancement Associates (SEA) направила је програм ARC, који не само да је више одвојених фајлова спајао у један велики фајл, него је истовремено вршио и компресију да би се уштедео простор на диску, односно цена преношења података модемом. Фајлови добијани као резултат рада програма ARC имали су наставак .arc.

Касније је Фил Кац направио сопствену верзију програма коју је назвао PKARC (за креирање архиве), односно PKXARC (за деархивирање). Коришћени формат фајлова био је исти као код ARC-а, али је PKARC био бржи. За разлику од фирме SEA, чији је програм био намењен и за креирање архиве и њено деархивирање, Кац је раздвојио те функције у два програма, чиме је смањио

[1] Phillip Walter Katz (1962 - 2000.), амерички програмер, најпознатији по томе што је креирао формат zip фајла и био аутор Pkzip-а.

[2] Детаљи о историји програма за компримовање података преузети су са интернет странице http://www.ursalab.com/article/PKZIP.html

количину меморије потребну за рад самог програма. То је такође отворило пут креирању архива које могу саме да се деархивирају, без потребе за постојањем посебног програма на рачунару где се врши деархивирање. То је урађено тако што је део програма потребног за деархивирање уграђиван у сваку архиву. Тиме су добијане тзв. самораспакујуће (self-extracting) архиве.

Компанија SEA, незадовољна појавом конкуренције, оптужила је Каца за кршење закона о ауторским правима, као и за наводно копирање дела кода програма. Кац је после судског процеса платио 62500 долара и морао је да промени имена својих програма у PKPAK и PKUNPAK. Убрзо, Кац је изменио своје програме, тако да су се сада базирали на потпуно новој техници компресије фајлова, па им је и имена променио у PKZIP и PKUNZIP. Кацови програми били су супериорнији у односу на ARC, тако да су највећим делом они коришћени за компресију података у MS-DOS-у.

Током 80-их година развијени су и други програми за компресију, као што су ZOO (Rahul Dhesi), DWC (Dean W. Cooper), LHarc (Haruhiko Okomura и Haruyasu Yoshizaki), ARJ (Robert Jung).

Прва верзија PKZIP-а појавила се 1989. године. Радила је под DOS-ом и дељена је као бесплатни софтвер (shareware) који треба регистровати и при том платити 25 долара.

Верзије PKZIP-а

PKZIP 0.9 подржава reducing алгоритам за компресију са четири различите опције, као и shrinking методу компресије.

PKZIP 1.0 подржава shrinking алгоритам за компресију. Иако јако популарни у то доба, фајлови прављени у верзији 1.0 данас су веома ретки, тако да на многим данашњим алатима за деархивирање нису уграђене опције за рад са shrinking и reducing алгоритмима.

PKZIP 1.1 (1990. године) је реализован са новим алгоритмом за компресију – imploding. Он је и данас обично подржан од стране свих програма за деархивирање.

У PKZIP 2.0 (1993. године) је уграђен нови алгоритам за компресију – deflating. Резултат је био општа присутност .ZIP фајлова на свим рачунарима који су користили Microsoft Windows или интернет.

PKZIP 2.04 је верзија која је била лиценцирана за IBM. Ово је и верзија на којој је базиран овај рад, јер она користи само алгоритам за шифровање који је PKWARE развио специјално за примену у програму PKZIP. Новије верзије користе и новије алгоритме за шифровање, који ће овде бити само побројани.

PKZIP 2.50 је прва верзија која је реализована за Windows 3.1, 95 и NT. DOS верзија 2.50 појавила се 1999. године и то је последња верзија која је урађена за DOS оперативни систем. Ова верзија је подржавала дугачка имена фајлова и deflate64 алгоритам за компресију. Такође промењена је и синтакса позива програма из командне линије.

PKZIP 2.6 је последња верзија која је подржавала Windows 3.1 и Windows NT за DEC Alpha и PowerPC платформе.

PKZIP 2.70 је верзија која је побољшала креирање и подешавање самораспакујућих архива.

PKZIP верзија 3.0 је прескочена јер се појавио тројанац са истим именом.

PKZIP 4.0 је унапређена верзија 2.7. Она је користила X.509 v3 сертификат за проверу аутентичности. Такође подржавала је и креирање великих .ZIP фајлова помоћу механизма дељења једног великог фајла на неколико мањих делова који су чувани на различитим локацијама (Span и Split .ZIP фајлови).

PKZIP 4.5 (2001. године) је укључио подршку за ZIP64 архиве, које омогућавају убацивање више од 65535 фајлова у једну архиву, као и убацивање у архиву фајлова већих од четири гигабајта.

PKZIP 5.0 (2002. године) је подржао *јако шифровање* (strong encryption specification); тиме је имплицитно призната слабост алгоритма о коме је реч у овом раду. Специјално, могуће је коришћење новијих алгоритама за шифровање као што су DES, 3DES, RC2, RC4 и алгоритам базиран на коришћењу X.509 v3 сертификата.

PKZIP 6.0 (2003. године) је подржао алгоритам за компресију BZip2 (базиран на Burrows-Wheeler трансформацији), као и 256-битну варијанту алгоритма за шифровање AES.

PKZIP 7.0 је изменио јако шифровање тако да користи неке нове могућности. Такође од ове верзије могуће је креирати и архиве других типова: BZIP2, GZIP, TAR, UUEncoded и XXEncoded.

PKZIP 8.0 (2004. године) је омогућио креирање архива са шифрованим заглављем.

PKZIP 9.0 је прва верзија која је подржавала Windows Vista оперативни систем (само за администраторе). Почеши од ове верзије више се не користе алгоритми за шифровање RC2 и DES.

PKZIP 10 је понудио креирање ZIP64 архива за произвољну циљну платформу.

24. априла 2007. године представљена је верзија 11. PKWARE је променио име PKZIP у SecureZIP. Ова верзија је подржавала LZMA компресију и додата је подршка за UTF-8 имена фајлова.

SecureZIP 12 је верзија представљена у фебруару 2008. године, а SecureZIP 12.1 је верзија из јуна 2008. године.

Напади на PKZIP

PKZIP карактерише доста робусно шифровање, али ипак постоје три начина за проналажење лозинке којом је шифрован фајл, као и за деархивирање података из архиве: напад методом грубе силе (brute force attack), речнички напад (dictionary attack) и напад на интерни кључ (напад са познатим отвореним текстом и шифратом).

Напад методом грубе силе

У нападу методом грубе силе тестира се свака могућа комбинација знакова (великих и малих слова, бројева и специјалних симбола) као кандидат за лозинку.

Напад овом методом обично траје јако дуго (осим ако се случајно брзо не погоди лозинка) и може бити успешан само код кратких лозинки. Може се урадити модификација овог метода тако да форсира одређени тип карактера (нпр. мала слова) или одређени подскуп свих могућих карактера, под условом да се може „наслутити“ ког је типа лозинка.

Успешност откривања лозинке овом методом је загарантована, али време рада самог програма може бити јако дуго. Проналажење дугих лозинки, са садашњом технологијом, може да потраје дуже од времена протеклог од настанка универзума до данас. На пример, нека се лозинка састоји само од малих и великих слова енглеске абецеде и нека је дугачка максимално 20 знакова. Укупно има $(26 + 26)^{20} \approx 2 * 10^{34}$ могућих лозинки. Ако рачунар обрађује милијарду могућих лозинки у секунди, требаће му $2 * 10^{25}$ секунди (око 10^{17} година) да сигурно пронађе лозинку.

Речнички напад

Речнички напад је поступак одређивања лозинке испитивањем могућих речи — кандидата за лозинку из дуге, унапред формиране листе. Та листа се формира од што је могуће више речи из различитих језика (одатле потиче име овог напада), од имена, бројева, фраза, цитата и других кандидата за лозинку. Такође се испитују и различите комбинације две или више речи, уз комбиновање великих и малих слова.

Рачунар може испитати велики број кандидата за лозинку у кратком временском интервалу, тако да овакви напади релативно кратко трају, али се не гарантује проналажење праве лозинке. Основна идеја оваквог напада је да су људи у принципу слични и да већина људи за лозинку бира неки појам, име, број или њихову комбинацију. Лозинка је обично кратка (да би се лако запамтила) и на неки начин везана за самог човека (датум рођења, број регистарских таблица, име унука . . .). Дугачке и компликоване комбинације речи или неких бесмислених знакова се јако тешко памте, а ако се негде запишу, онда то угрожава њихову тајност.

Једна од могућих препорука је да се лозинке састоје од неколико бесмислено повезаних слогова (нпр. што ла су кро). Овакве лозинке се лако памте; слогова има веома много, а њихових комбинација има још више. Пошто су комбинације слогова најчешће бесмислене, ниједан речник их не може све пописати, па за овако изабране лозинке речнички напад ретко даје резултате.

Напад на интерни кључ

Напад овом методом детаљно се описује у наставку. Напад је из категорије напада са познатим отвореним текстом и шифратом. Ако је познат изглед једног дела података нешифрованог фајла (отворени текст) и њему одговарајући део података шифрованог фајла, онда је могуће одредити преостале, непознате податке из тог фајла и лозинку којом је шифрована архива, као и прочитати податке из осталих фајлова садржаних у истој архиви. У самом нападу најпре се одређује 96-битни интерни кључ, који је довољан за дешифровање преосталог

дела фајла, а затим се реконструише и сама лозинка. Идеју за овај напад дали су Ели Бихам (Eli Biham[3]) и Пол Кохер (Paul C. Kocher[4]) [2]. Да би се извршио напад потребно је знати бар 13 бајтова отвореног текста (у компримованом облику) и њима одговарајуће бајтове шифрованог фајла. Ако је познато више од 13 бајтова, програм брже проналази лозинку јер се део познатих бајтова користи за редукцију броја могућих кандидата. Што је дужи познати отворени текст, то је потребно краће време за проналажење саме лозинке. Овај метод је доста компликован за разумевање, али даје доста добре резултате и најчешће омогућује успешно одређивање лозинке.

[3] Computer Science Department, Technion – Israel Institute of Technology, Хаифа, Израел
[4] независни криптографски консултант, САД

Поглавље 2

Програм PKZIP

2.1 Структура .ZIP фајла

Сваки .ZIP фајл садржи један или више произвољних фајлова који могу (а не морају) бити компримовани и/или шифровани. Ти фајлови су смештени у меморији у произвољном редоследу. Ако се не каже другачије, целобројне вредности се чувају у меморији тако да је последњи бајт најмање тежине (little-endian).

Сваки .ZIP фајл има следећу општу структуру [3]:

Заглавље првог фајла (*local file header 1*)
Садржај (подаци) првог фајла (*file data 1*)
Дескриптор података првог фајла (*data descriptor 1*)
Заглавље другог фајла
Садржај (подаци) другог фајла
Дескриптор података другог фајла
. . .
Заглавље n–тог фајла
Садржај (подаци) n–тог фајла
Дескриптор података n–тог фајла
Заглавље за дешифровање архиве (*archive decryption header*)
Поље са додатним подацима о архиви (*archive extra data record*)
Централни директоријум (*central directory*)

Поља **заглавље фајла** и **садржај фајла** морају бити присутни за сваки фајл садржан у .ZIP архиви, док поље **дескриптор података фајла** постоји само ако је постављен одговарајући индикатор (flag) у заглављу фајла.

Поља **заглавље за дешифровање архиве** и **поље са додатним подацима о архиви** појављују се у структури фајла од верзије 6.2 и то само ако је централни директоријум шифрован. Ова поља садрже детаље о начину шифровања.

У пољу централни директоријум сакупљени су подаци о дескрипторима појединачних фајлова укључених у архиву и разне додатне информације, као што

су атрибути фајлова, коментари о фајловима, локације у меморији заглавља фајлова, итд.

Заглавље фајла

Заглавље сваког фајла има следећу структуру:

Поље	**Величина**
Сигнатура (*local file header signature*)	4 бајта
Верзија потребна за екстракцију(*version needed to extract*)	2 бајта
Флегови (*general purpose bit flag*)	2 бајта
Метод компресије (*compression method*)	2 бајта
Време (*last modification file time*)	2 бајта
Датум (*last modification file date*)	2 бајта
CRC-32	4 бајта
Величина компримованог фајла (*compressed size*)	4 бајта
Величина некомпримованог фајла (*uncompressed size*)	4 бајта
Дужина имена фајла (*file name length*)	2 бајта
Дужина допунског поља (*extra file length*)	2 бајта
Име фајла (*file name*)	променљива
Садржај допунског поља (*extra field*)	променљива

Поље **сигнатура** показује да се ради о фајлу који је део .ZIP архиве. Она увек има вредност 0x504b0304.

Поље **верзија потребна за екстракцију** показује која је верзија PKZIP-а потребна да би фајл могао да се деархивира.

Поље **флегови** даје разне информације о фајлу:

- Бит 00 – ако је постављен на један, фајл је шифрован;
- Бит 01 и бит 02 – користе се за постављање разних опција при компримовању фајлова. Њихова конкретна употреба зависи од типа компресије који се користи. За различите типове компресије ови битови имају потпуно различита значења;
- Бит 03 – ако је постављен на један, поља crc-32, величина компримованог фајла и величина некомпримованог фајла имају вредност нула у заглављу фајла. Права вредност тих поља налази се у дескриптору података фајла. Другим речима, ако је бит постављен на један, поље дескриптор података фајла постоји у структури фајла, а ако има вредност нула, дескриптор података фајла не постоји у структури фајла[1];
- Бит 04 – користи се при компресији типа enhanced deflating;
- Бит 05 – ако је постављен на један, то је показатељ да је извршена компресија помоћу издељених података[2] (pached data);

[1] У верзији 2.04 овај бит се препознаје једино ако је коришћена компресија типа deflate, док се у новијим верзијама препознаје при свим типовима компресије.

[2] Користи се у верзији 2.70 или већој.

- Бит 06 – Ако је постављен на један, то је показатељ да се користи јако шифровање. Ово такође значи да се поље верзија потребна за екстракцију мора поставити на вредност 50 или већу и да се мора поставити бит 00. Ако се користи шифровање алгоритмом AES, поље верзија потребна за екстракцију мора имати вредност 51 или већу;
- Битови 07-10 – тренутно се не користе;
- Бит 11 – ако је постављен на један, име фајла и поље које садржи коментар о фајлу кодирани су коришћењем UTF-8;
- Бит 12 – резервисано за PKWARE;
- Бит 13 – користи се приликом шифровања централног директоријума;
- Битови 14-15 – резервисани за PKWARE.

Поље **метод компресије** показује који је тип компресије коришћен и има следеће могуће вредности:

- 0 – фајл није компримован
- 1 – коришћена је компресија типа shrinking
- 2 – коришћена је компресија типа reduction са фактором 1
- 3 – коришћена је компресија типа reduction са фактором 2
- 4 – коришћена је компресија типа reduction са фактором 3
- 5 – коришћена је компресија типа reduction са фактором 4
- 6 – коришћена је компресија типа imploded
- 7 – резервисан за компресију типа tokenizing коју Pkzip не користи
- 8 – коришћена је компресија типа deflating
- 9 – коришћена је компресија типа enhanced deflating
- 10 – коришћена је компресија типа PKWARE DCL imploding
- 11 – резервисано за PKWARE
- 12 – компресија је извршена коришћењем алгоритма BZIP2
- 13 – резервисано за PKWARE
- 14 – коришћена је компресија типа LZMA
- 15-17 – резервисано за PKWARE
- 18 – коришћена је компресија типа IBM TERSE

- 19 – коришћена је компресија типа IBM LZ77 z
- 97 – коришћена је компресија типа WavPack
- 98 – коришћена је компресија типа PPMd

Поља **време** и **датум** показују време и датум последње модификације фајла. Оба податка сачувана су у стандардном MS-DOS формату.
За време:

- битови 00-04 показују секунде (подељене са 2) последње модификације фајла;
- битови 05-10 показују минуте последње модификације фајла;
- битови 11-15 показују сате последње модификације фајла.

За датум:

- битови 00-04 показују дан последње модификације фајла;
- битови 05-08 показују месец последње модификације фајла;
- битови 09-15 показују годину (заправо број година протеклих од 1980. године) последње модификације фајла.

Поље **crc-32** садржи вредност добијену алгоритмом CRC-32 [3] са улазом једнаким садржају фајла сачуваног у архиви и „магичним бројем“ 0xdebb20e3 (видети страну 23).

Поља **величина компримованог фајла** и **величина некомпримованог фајла** показују величину компримованог, односно некомпримованог фајла. Ако је архива у ZIP64 формату, та поља садрже вредности 0xffffffff.

Поље **дужина имена фајла** садржи дужину имена фајла. Ако се име фајла задаје са стандардног улаза, дужина имена фајла се поставља на 0.

Поља **име фајла** и **садржај допунског поља** су променљиве дужине. Њихова дужина се види у пољима дужина имена фајла и дужина допунског поља.

Садржај фајла

Подаци из изворног фајла у .ZIP архиви могу бити шифровани и/или компримовани. При томе се најпре врши компримовање података, а затим њихово шифровање.

Дескриптор података фајла

Ово поље постоји само ако је бит 03 у пољу флегови постављен на један. У том случају ово поље садржи три податка: crc-32, величину компримованог фајла и величину некомпримованог фајла.

[3] Првобитну реализацију урадио је Дејвид Шведерер (David Schwaderer), а једна имплементација прилог је у практичном делу овог рада.

Заглавље за дешифровање архиве

Ово поље[4] користи се само при шифровању централног директоријума. Конципирано је тако да обезбеди подршку јаком шифровању.

Поље са додатним подацима о архиви

Као и претходно и ово поље[5] се користи приликом шифровања централног директоријума. Ако је присутно, ово поље непосредно претходи централном директоријуму. Величина овог поља укључена је у поље величина централног директоријума које се налази у пољу крај централног директоријума. Ово поље има три саставна елемента:

- Сигнатура (*archive extra data signature*) – поље величине 4 бајта. Увек има вредност 0x08064b50;
- Дужина допунског поља (*extra field length*) – поље величине 4 бајта;
- Допунско поље (*extra field data*) – садржај допунског поља, променљиве је величине.

Централни директоријум

Централни директоријум садржи информације о фајловима садржаним у архиви, као и информације о Zip64 архивама[6]. Структура централног директоријума је следећа:

Заглавље фајла 1 централног директоријума (*file header 1*)
Заглавље фајла 2 централног директоријума
. . .
Заглавље фајла n централног директоријума
Дигитални потпис (*digital signature*)
Крај централног директоријума за Zip64 архиве (*zip64 end of central directory record*)
Локација краја централног директоријума за Zip64 архиве (*zip64 end of central directory locator*)
Крај централног директоријума (*end of central directory record*)

Заглавља фајлова централног директоријума слична су заглављима самих фајлова на почетку архиве, али садрже и неке додатне информације.

Поља **крај централног директоријума за Zip64 архиве** и **локација краја централног директоријума за Zip64 архиве** садрже информације о 64-битним Zip архивама које нису предмет овог рада.

Овде се не наводе детаљи о пољу **дигитални потпис**, јер се оно користи при јаком шифровању[7].

[4] Представљено у верзији 6.2 спецификације ZIP формата.
[5] Неопходна верзија 6.2 PKZIP-а или већа.
[6] Ово су 64-битне архиве. О њима неће бити речи у овом раду.
[7] Неопходна верзија 6.2 PKZIP-а или већа.

Заглавље фајла у централном директоријуму

Структура заглавља фајла у централном директоријуму има следећи облик (већина поља иста је као у заглављу фајла, па се коментаришу само она поља која се први пут помињу):

Поље	**Величина**
Сигнатура (*central file header signature*)	4 бајта
Архива направљена у верзији (*version made by*)	2 бајта
Верзија потребна за екстракцију (*version needed to extract*)	2 бајта
Флегови (*general purpose bit flag*)	2 бајта
Метод компресије (*compression method*)	2 бајта
Време (*last modification file time*)	2 бајта
Датум (*last modification file date*)	2 бајта
CRC-32	4 бајта
Величина компримованог фајла (*compressed size*)	4 бајта
Величина некомпримованог фајла (*uncompressed size*)	4 бајта
Дужина имена фајла (*file name length*)	2 бајта
Дужина допунског поља (*extra file length*)	2 бајта
Дужина коментара о фајлу (*file comment length*)	2 бајта
Број диска почетка фајла (*disk number start*)	2 бајта
Интерни атрибути фајла (*internal file attributes*)	2 бајта
Екстерни атрибути фајла (*external file attributes*)	4 бајта
Размак заглавља фајла од почетка директоријума (*relative offset of local header*)	4 бајта
Име фајла (*file name*)	променљива
Допунско поље (*extra field*)	променљива
Коментар о фајлу (*file comment*)	променљива

Поље **сигнатура** увек има вредност 0x054b0102.

У пољу **архива направљена у верзији**, бајт мање тежине показује која је верзија Zip спецификације коришћена, а бајт веће тежине показује у ком оперативном систему и фајл систему је направљена архива. Он може да има следеће вредности (у зависности од оперативног система):

0 – MS-DOS или OS/2 (FAT/VFAT/FAT32 оперативни систем)

1 – Amiga	2 – OpenVMS	3 – UNIX	4 – VM/CMS
5 – Atari ST	6 – OS/2 H.P.F.S	7 – Macintosh	8 – Z-system
9 – CP/M	10 – Windows NTFS	11 – MVS (OS/390 -Z/OS)	
12 – VSE	13 – Acorn Risc	14 – VFAT	15 - alternateMVS
16 – BeOS	17 – Tandem	18 – OS/400	19 - OS/X(Darwin)

20 – 255 – не користе се.

Поље **број диска почетка фајла** показује број диска на коме се налази почетак фајла.

У пољу **интерни атрибути фајла**

- бит 0, ако је постављен на јединицу, показује да је реч о текстуалном или ASCII фајлу, а ако је постављен на нулу, реч је о бинарном фајлу.

- Бит 1 је резервисан за употребу од стране PKWARE.

- Бит 2, ако је постављен на јединицу, показује да сваком логичком запису претходи четворобајтно поље у коме је записана дужина записа. Ова вредност је предвиђена за подршку размени података између великих рачунара.

- Битови 3-15 се не користе.

Садржај поља **екстерни атрибути фајла** зависи од оперативног система на коме је направљена конкретна архива.

Поље **Размак заглавља фајла од почетка директоријума** показује где се налази заглавље фајла у односу на почетак диска на коме се налази почетак тог фајла.

Крај централног директоријума

Крај централног директоријума има следећу структуру:

Поље	**Величина**
Сигнатура (*end of central directory signature*)	4 бајта
Број диска (*disk number*)	2 бајта
Број старт диска (*disk number with the start of the central directory*)	2 бајта
Број фајлова централног директоријума на текућем диску (*total number of entries in the central dir. on this disc*)	2 бајта
Укупан број фајлова централног директоријума (*total number of entries in the central directory*)	2 бајта
Величина централног директоријума (*size of central directory*)	4 бајта
Почетак централног директоријума (*offset of start of central directory with respect to the starting disk number*)	4 бајта
Дужина коментара (*.ZIP file comment length*)	2 бајта
Коментар (*.ZIP file comment*)	променљива

Поље **сигнатура** увек има вредност 0x054b0506.

Поље **број диска** садржи број диска на коме се налази крај централног директоријума.

Поље **број старт диска** садржи број диска на коме се налази почетак централног директоријума.

Поље **почетак централног директоријума** садржи локацију почетка централног директоријума на диску.

У пољу **коментар** чувају се коментари о .ZIP фајлу. Ови подаци нису заштићени, тј. на њих се не примењује шифровање и проверавање аутентичности, тако да ту не би требало уписивати поверљиве податке.

2.2 Методи компресије

PKZIP користи више различитих метода компресије [5]. У пољу компресија (в. стр. 13) чува се метод компресије који се користи. У зависности од метода компресије, неки битови поља флегови (в. стр. 12) користе се за додатне информације о изабраном методу компресије.

Пошто има доста метода компресије (сваки од њих захтева доста додатних објашњења), а сама компресија није предмет овог рада, за сваки метод дају се само основне карактеристике, без улажења у детаље (такође наводи се литература у којој се може сазнати више о конкретним методама). Као илустрација функционисања компресије, мало детаљније се приказује алгоритам за компримовање метода reducing.

Shrinking (метод 1)

Ово је динамички Ziv-Lempel-Welch[8] алгоритам за компресију са делимичним чишћењем (*partial clearing*). Овај алгоритам разликује се од стандардног Ziv-Lempel-Welch алгоритма у неколико детаља (алгоритам и детаљан опис може се видети на интернет адреси http://marknelson.us/1989/10/01/lzw-data-compression.

Reducing (методи 2-5)

Овај метод је комбинација два различита алгоритма. Први алгоритам компримује низове бајтова који се понављају, а други алгоритам узима компримоване низове бајтова из првог алгоритма и на њих примењује вероватносни метод компресије (*probabilistic compression method*). Овај метод компресије обједињује четири метода који су слични, а разликују се само у фактору компресије.

Вероватносна компресија чува низ скупова $S(j), j = 0, 255$, који одговарају сваком могућем ASCII карактеру. Сваки од ових скупова садржи између 0 и 32 карактера, који су означени са $S(j)[0], \ldots, S(j)[m]$, где је $m < 32$. Скупови се чувају на почетку поља са подацима о редукованом фајлу и то у обрнутом редоследу: $S(255)$ је први, а $S(0)$ последњи. Ови скупови су кодирани у облику $\{N(j), S(j)[0], \ldots, S(j)[N(j)-1]\}$, где је $N(j)$ величина скупа $S(j)$. Свака вредност $N(j)$ се кодира са 6 битова, а за њом следе $N(j) * 8$ бита који одговарају вредностима $S(j)[0], \ldots, S(j)[N(j)-1]$. $N(j)$ може бити нула. У том случају скуп $S(j)$ је празан и не чува се вредност за $S(j)$, него одмах следи вредност $N(j-1)$. Одмах иза ових скупова чувају се компримовани подаци фајла. Декомпримовање података врши се у два корака. Прво се врши вероватносна декомпресија и то следећим алгоритмом:

$posledni\_karakter = 0$;
while(има карактера на улазу) **do**
  if(S($posledni\_karakter$) је празан)

[8] Jacob Ziv, израелски информатичар,
Abraham Lempel, израелски информатичар,
Terry A. Welch, амерички информатичар.

прочитај осам битова са улаза и копирај ту вредност на излаз;
else
прочитај један бит са улаза;
if(прочитани бит није нула)
прочитај осам битова са улаза и копирај ту вредност на излаз;
else
прочитај B(N(*poslednji_karakter*)) битова са улаза и
означи ту вредност са I;
копирај вредност S(*poslednji_karakter*)[I] на излаз;
poslednji_karakter=последња вредност која је стављена на излаз;

где B(N(j)) представља минималан број битова потребних за кодирање вредности N(j)-1.

У другом кораку декомпресије узимају се излазни подаци из првог дела и да би се добио оригинални фајл (пре компримовања), примењује се следећи алгоритам:

stanje = 0;
while (има карактера на улазу) **do**
учитај осам битова са улаза у C;
case (*stanje*) **of**
0: **if** (C != DLE(144, децимално))
копирај вредност C на излаз;
else *stanje* = 1;
1: **if**(C != 0)
V = C;
duzina = L(V);
stanje = F(*duzina*);
else
копирај вредност 144(децимално) на излаз;
stanje = 0;
2: *duzina* = *duzina* + C;
stanje = 3;
3: враћају се D(V,C) бајтова из излазног тока података (ако се тиме позиција текућег елемента излазног тока података поставља на елемент пре старта, тада се сви подаци од те позиције до почетка података излазног тока постављају на нуле);
копирај *дужина* + *три бајта* (од тренутне позиције) на излаз;
stanje = 0;

Функције F, L и D зависе од фактора компресије, који може бити 1, 2, 3 или 4 (од тог фактора зависи да ли је то метода типа 2, 3, 4 или 5). Ове функције дефинисане су на следеће начине:

Компресија са фактором 1:
L(X) = нижих седам битова податка X.
F(X) = 2, ако је X = 127, у супротном F(X) = 3.
D(X,Y) = (бит највеће тежине податка X) * 256 + Y + 1.

Компресија са фактором 2:
L(X) = нижих шест битова податка X.

F(X) = 2, ако је X = 63, у супротном F(X) = 3.
D(X,Y) = (2 бита највеће тежине податка X) * 256 + Y + 1.
Компресија са фактором 3:
L(X) = нижих пет битова податка X.
F(X) = 2, ако је X = 31, у супротном F(X) = 3.
D(X,Y) = (3 бита највеће тежине податка X) * 256 + Y + 1.
Компресија са фактором 4:
L(X) = нижа четири бита податка X.
F(X) = 2, ако је X = 15, у супротном F(X) = 3.
D(X,Y) = (4 бита највеће тежине податка X) * 256 + Y + 1.

Imploding (метод 6)

Imploding алгоритам је комбинација два различита алгоритма. Први алгоритам компримује низове бајтова који се понављају, користећи *метод локалног речника* (sliding dictionary алгоритам). Други алгоритам компримује излазне податке из првог алгоритма користећи вишеструка Шанон-Фано[9] (Shannon-Fano) дрвета (алгоритам и детаљан опис метода се може погледати на интернет страници http://en.wikipedia.org/wiki/ShannonFano_trees или на http:// compqt.googlepages.com/shannonfano).

Овај метод зависи од неких битова из поља флегови. Број Шанон-Фано дрвета зависи од бита 2 из поља флегови. Ако је он постављен на нулу, постоје два дрвета, а ако је постављен на јединицу, постоје три дрвета. У случају да постоје два дрвета, прво је *дрво дужине* (Length tree), а друго је *дрво одстојања* (Distance tree). У случају да постоје три дрвета, на почетку се налази *дрво слова* (Literal characters tree), а за њим следе дрво дужине, па дрво одстојања.

Дрво слова се користи да представи целокупан ASCII скуп карактера и садржи 256 вредности. Ово дрво се користи за компримовање оних података који нису компримовани sliding dictionary алгоритмом. Када је дрво слова присутно, вредност променљиве Minimum Match Length из sliding dictionary алгоритма има вредност три, а ако дрво слова не постоји ова променљива има вредност два.

Излаз из метода локалног речника је низ уређених парова (*дужина, удаљеност*). Дрво дужине се користи за компримовање оних података који представљају део *дужина*, а дрво одстојања се користи за компримовање података који представљају део *удаљеност* излазних података из метода локалног речника.

Шанон-Фано дрвета чувају се на почетку података о садржају компримованог фајла и то у компримованом облику.

Tokenizing (метод 7)

Ова метода компресовања се не користи у PKZIP-у.

---

[9] Claude Elwood Shannon, амерички електротехничар и математичар, (1916.–2001.), Roberto Mario Fano, италијански информатичар, (1917. –).

Deflating (метод 8)

Deflate алгоритам се састоји из два дела. У првом делу алгоритма врши се компримовање низова бајтова који се понављају коришћењем метода локалног речника (као у imploding методи). У другом делу алгоритма, на излазне податке из првог дела примењује се Хофманово[10] кодирање. Компримовани подаци се чувају у блоковима који имају заглавље. У заглављу су дате информације о самом блоку и Хофмановом коду који се користи у том блоку.

Основна идеја Хофмановог кодирања је да се сваком знаку који се појављује у фајлу који се компримује додели по један код. Ако се знак чешће појављује у фајлу - он има краћи код, а ако се знак ретко појављује у фајлу - онда он има дужи код. Ако је низ улазних знакова другачији, и само кодирање ће бити другачије. Знаковима се придружују кодови у зависности од скупа улазних знакова и њихове фреквенције, тј. једно исто слово у зависности од осталих улазних слова може у различитим блоковима добијати различите кодове. Детаљи се могу видети у [8].

Enhanced Deflating (метод 9)

Овај алгоритам за компресију је сличан методу Deflate, само што користи метод локалног речника на другачији начин.

BZIP2 (метод 12)

BZIP2 је алгоритам за компресију који је развио Џулијан Сивард[11]. Детаљне информације и изворни код овог алгоритма могу се наћи на интернет страници http://www.bzip.org.

LZMA (метод 14)

LZMA (Lempel-Ziv-Markov chain-Algorithm) је блок-орјентисани алгоритам за компресију који је развио Игор Павлов[12] 1998. године. Он је изведен из LZ77 (Lempel-Ziv) алгоритма и примењује Марковљеве[13] ланце. Детаљне информације и изворни код овог алгоритма могу се пронаћи на интернет страници http://www.7-zip.org.

WavPack (метод 97)

Овај метод се ослања на WavPack аудио компресију коју је развио Дејвид Брајант[14]. Користи се од 1998. године. Детаљне информације о овом методу компресије могу се наћи на интернет страници http://www.wavpack.com.

---

[10] David A. Huffman, амерички информатичар, (1925 – 1999.).
[11] Julian Seward, британски информатичар.
[12] Игор Павлов, руски програмер, познат по развоју програма 7-zip.
[13] Андрей Марков, руски математичар, (1856 – 1922).
[14] David Bryant, новозеландски математичар.

PPMd (метод 98)

PPMd (Prediction by Partial Matching) је алгоритам за компресију који су развили Димитриј Шкарин и Димитриј Суботин[15] 2003. године. Опис овог алгоритма за компресију може се наћи нпр. на http://www.compression.ru/ds/.

2.3 Алгоритам шифровања

У овом раду разматра се алгоритам шифровања који је развио Роџер Шлафли[16] специјално за примену у програму PKZIP [5].

У програму PKZIP компресија података претходи шифровању. Шифровани подаци се најпре дешифрују, па тек онда декомпримују.

Сваки шифровани фајл садржи додатних 12 бајтова ($P_1, P_2, \ldots, P_{12}$), тзв. *шифарско заглавље* (encryption header). Ових 12 бајтова чувају се иза заглавља фајла, а пре почетка поља са компримованим садржајем фајла. Првих 11 бајтова се добијају као случајни бројеви, а 12. бајт је повезан са заглављем фајла и представља *бајт највеће тежине* (most significant byte – MSB) поља CRC-32. Овај податак се користи за проверу коректности унете лозинке за приступ фајлу. Заглавља фајлова се не шифрују.

Алгоритам шифровања користи коначни аутомат $\mathcal{K}$ са скупом од 2^{96} могућих стања и улазном, односно излазном, азбуком која се састоји од свих 2^8 бајтова. Стање $\mathcal{K}$ одређено је садржајем три 32-битна регистра $key0$, $key1$ и $key2$. Ако се $\mathcal{K}$ налази у стању $(key0, key1, key2)$ и добије улазни бајт $char$, онда он прелази (на начин који ће бити описан, видети функцију $update\_keys$) у ново стање $(key0', key1', key2')$ и на излазу даје бајт $key3$. Излазни бајт $key3$ зависи само од $key2'$.

Почетно стање $\mathcal{K}$ је $key0 = 0x12345678$, $key1 = 0x23456789$, $key2 = 0x34567890$. У том стању $\mathcal{K}$ добија редом на улаз бајтове лозинке. Стање у коме се $\mathcal{K}$ затекне по уношењу последњег бајта лозинке зове се *интерни кључ*.

У процесу шифровања и дешифровања користе се функције crc_32, $crc32$ и $update\_keys_i$.

Функција $crc\_32(temp, i)$ има два аргумента: 32-битну реч $temp$ и бајт i. Позив функције мења вредност $temp$.

$crc\_32(temp, i)$:
    $temp = temp \ \oplus \ i$;
    for $j = 0$ **to** 7
        if $odd(temp)$ **then**
            $temp = (temp \gg 1) \ \oplus \ 0xEDB88320$;
        else
            $temp = temp \gg 1$;
    return $temp$;

Овде је $odd(temp)$ функција која испитује да ли је вредност променљиве $temp$

[15] Дмитри Шкарин, руски информатичар,
Дмитри Суботин, руски информатичар.
[16] Roger Schlafly, амерички програмер.

непаран број и у том случају враћа вредност један, а ако је $temp$ паран број, враћа вредност нула.

Функција crc_32 се користи за иницијализацију табеле $crctab$:

$init\_crc()$:
  $local\ unsigned\ long\ temp$;
  for $i = 0$ **to** 255
    $temp = crc\_32(0, i)$;
    $crctab[i] = temp$;

Функција $crc32$ има као аргументе један знак $char$ и претходну 32-битну вредност $pval$. Њиховим комбиновањем добија се нова 32-битна вредност

$$nval = crc32(pval, char) = (pval \gg 8) \ \oplus \ crctab[LSB(pval) \ \oplus \ char].$$

Овде $LSB(pval)$ означава најнижи бајт вредности $pval$.

Функција $update\_keys(char)$ израчунава ново стање $(key0_{i+1}, key1_{i+1}, key2_{i+1})$ аутомата $\mathcal{K}$ и излазни бајт $key3_{i+1}$ полазећи од тренутног стања $(key0_i, key1_i, key2_i)$ аутомата и улазног знака $char$:

$update\_keys_i(char)$
  $local\ unsigned\ short\ temp$;
  $key0_{i+1} = crc32(key0_i, char)$;
  $key1_{i+1} = (key1_i + LSB(key0_{i+1}) * 134775813) + 1 \pmod{2^{32}}$;
  $key2_{i+1} = crc32(key2_i, MSB(key1_{i+1}))$;
  $temp_{i+1} = key2_{i+1}|3$;
  $key3_{i+1} = LSB((temp_{i+1} * (temp_{i+1} \ \oplus \ 1)) \gg 8)$;

где $\gg$ представља померање удесно за 8 бита, $|$ је бинарно или, $\oplus$ је логичка операција xor, а LSB и MSB су бајтови најмање, односно највеће тежине одговарајућег четворобајтног податка.

Процеси шифровања и дешифровања грубо су описани следећим псеудокодом:

Шифровање

додавање бајтова $P_1, \ldots P_{12}$ испред компримованог текста;
for $i = 1$ **to** n
  $C_i = P_i \ \oplus \ key3_i$;
  $update\_keys_i(P_i)$;

Дешифровање

for $i = 1$ **to** n
  $P_i = C_i \ \oplus \ key3_i$;
  $update\_keys_i(P_i)$;
уклањање бајтова $P_1, \ldots P_{12}$;

Овде је C_i i-ти знак шифрованог фајла, а P_i је одговарајући знак нешифрованог фајла, $i = 1, \ldots, n$. Дужина нешифрованог фајла је $n - 12$. Шифарско заглавље чине бајтови $P_1, \ldots, P_{12}$.

Само шифровање се изводи у три корака. У првом кораку се поставља (фиксирано) почетно стање $\mathcal{K}$ и мења се под утицајем лозинке (коју уноси корисник програма):

$key0 = 0x12345678$;
$key1 = 0x23456789$;

$key2 = 0x34567890$;
for $i = 0$ **to** $strlen(password) - 1$
$update\_keys_i(password[i])$;

Другим речима, стање $\mathcal{K}$ се поставља на почетну вредност, затим се узима један по један знак из лозинке (password) и поставља се на улаз у функцију $update\_keys$, која врши модификацију стања $\mathcal{K}$. По завршетку првог корака шифровања, стање $\mathcal{K}$ је једнако интерном кључу.

У другом кораку шифрује се шифарско заглавље, уз истовремену модификацију стања $\mathcal{K}$

нека је $P_1, \ldots, P_{12}$ *шифарско заглавље*
for $i = 1$ **to** 12
$C_i = P_i \oplus key3_i$;
$update\_keys_i(P_i)$;

У трећем кораку чита се један по један знак из улазног фајла (добијеног после евентуалне компресије података) и шифрује се, уз истовремену модификацију стања $\mathcal{K}$:

for $i = 1$ **to** $sizeof(nesifr\_file)$
учитај P_i;
$C_i = P_i \oplus key3_i$;
$update\_keys_i(P_i)$;

где је $nesifr\_file$ фајл који се шифрује.

Процес дешифровања је веома сличан. Први корак је потпуно исти као код шифровања. У другом кораку учитавају се првих 12 бајтова (шифарско заглавље) и дешифрују се. При томе се проверава да ли је последњи (12.) дешифровани бајт једнак бајту највеће тежине поља CRC32, које се чита из заглавља фајла. Ако се те две вредности подударају, претпоставља се да је унета исправна лозинка наставља се са дешифровањем самог фајла (тих 12 бајтова се одбацују). У трећем кораку чита се један по један шифровани знак и добијају се одговарајући нешифровани (али компримовани) знакови.

Поглавље 3

Напад са познатим отвореним текстом и шифратом

Напад овом методом претпоставља да је позната комплетна шифрована архива и да је познато 13 или више узастопних бајтова једног фајла који припада тој архиви и није шифрован, али јесте компримован. У случају да нису познати компримовани бајтови, већ су познати бајтови оригиналног нешифованог и некомпримованог фајла, из заглавља саме шифроване архиве чита се којим је методом извршена компресија и у којој верзији програма PKZIP. Затим се том методом и у тој верзији PKZIP-а врши компримовање познатих бајтова оригиналног фајла и одатле се чита 13 или више потребних бајтова отвореног текста.

У наставку рада претпоставља се да је позната вредност почетних n бајтова, тј. познати су шифровани(C_i) и њима одговарајући нешифровани(P_i) бајтови, $i = 1, \ldots, n$. Одатле се добија низ вредности $key3_i$:

$$key3_i = P_i \ \oplus \ C_i.$$

Низ познатих бајтова (P_i) представља улазне податке за функцију $update\_keys$ (видети страну 23), а низ вредности $key3_i$ представља излазне податке из те функције, односно аутомата $\mathcal{K}$. На основу тога може се одредити стање $\mathcal{K}$ у неком, а затим и у било ком другом тренутку. Тако се добија интерни кључ — стање $\mathcal{K}$ пре уношења P_1. У наставку рада показује се како се добијају све могуће вредности за листу $key2_i$, затим како се на основу те листе добијају листе могућих вредности $key1_i$ и $key0_i$. На крају, показује се како се елиминишу погрешне вредности, после чега остаје само по једна вредност за $key0, key1$ и $key2$, односно интерни кључ.

3.1 Одређивање могућих вредности key2

На основу

$$temp = key2 \mid 3;$$
$$key3 = LSB((temp * (temp \oplus 1)) \gg 8);$$

вредност $key2$ одређује вредност $key3$. Осмобитна променљива $key3$ узима само битове са позиција 0-7 израза $(temp * (temp \oplus 1)) \gg 8$, односно битове са позиција 8-15 израза $temp * (temp \oplus 1)$. Тих осам битова зависе само од битова са позиција 0-15 вредности $temp$ (битови са позиција 16-31 вредности $temp$ не утичу на битове на позицијама 8-15 вредности израза $temp * (temp \oplus 1)$). Пошто се на позицијама 0 и 1 променљиве $temp$ увек налазе јединице, то значи да $key3$ зависи само од 14 битова променљиве $temp$ (на позицијама 2-15). Променљива $key3$ може узети 256 различитих вредности. Непосредно се проверава да се свака од тих вредности може се добити од тачно $2^{14}/256 = 64$ различите вредности 14 битова променљиве $temp$. Свакој од од те 64 вредности одговара 2^{16} могућности за горњих 16 битова, па укупно има 2^{22} могућности за $temp$. Како се $temp$ и $key2$ подударају свуда осим на два бита најмање тежине, закључује се да за било коју фиксирану вредност $key3$ постоји 2^{22} одговарајућих вредности горњих 30 битова променљиве $key2$.

Као што је речено, $key2_{i+1}$ се добија на основу израза $key2_{i+1} = crc32(key2_i, MSB(key1_{i+1}))$.
Функција $crc32$ дефинисана је изразом

$$nval = crc32(pval, char) = (pval \gg 8) \oplus crctab[LSB(pval) \oplus char],$$

где је $crctab$ табела унапред израчуната у функцији $init\_crc()$ (в. стр. 23). Садржај r елемента са индексом i, $i = 0, \ldots, 255$, у овој табели једнак је производу 32×8 матрице C са колоном i. Подматрица од првих осам врста матрице C је инвертибилна, па горњих осам бита r једнозначно одређује i. Нека за $i = 0, \ldots, 255$ реч која се добија проширивањем i са доњих 24 бита r померених улево за осам бита буде вредност са индексом i у матрици $crcinvtab$. Функција $init\_crc()$ може се проширити израчунавањем табеле $crcinvtab$:

```
init_crc() :
    local unsigned long temp;
    for i = 0 to 255
        temp = crc_32(0, i);
        crctab[i] = temp;
        crcinvtab[temp >> 24] = (temp << 8) ⊕ i;
```

Функција $crc32^{-1}$ дефинисана са

$$pval = crc32^{-1}(nval, char) = (nval \ll 8) \oplus crcinvtab[MSB(nval)] \oplus char$$

је на основу дефиниције $crcinvtab$ инверзна функцији $crc32$.

Из изложених чињеница следи да је

$$\begin{aligned} key2_i &= crc32^{-1}(key2_{i+1}, MSB(key1_{i+1})) \\ &= (key2_{i+1} \ll 8) \oplus crcinvtab[MSB(key2_{i+1})] \oplus MSB(key1_{i+1}) \end{aligned} \quad (3.1)$$

Претпоставимо да је познато горњих 30 битова вредности $key2_{i+1}$. Тада су горња 22 бита вредности $key2_i$ (често) једнозначно одређени. Заиста, за израз $key2_{i+1} \ll 8$ позната су горња 22 бита, за $crcinvtab[MSB(key2_{i+1})]$ позната су сва 32 бита, а $MSB(key_{i+1})$ утиче практично само на доњих 8 битова, док на горња 22 ретко утиче. Преглед чињеница о једнакости 3.1 наведен је у следећој табели:

Страна једнакости	Израз	Број познатих битова	Позиција познатих битова	Број могућих вредности
Лева	$key2_i$	14	2-15	64
Десна	$key2_{i+1} \ll 8$	22	10-31	1
	$crcinvtab[MSB(key2_{i+1})]$	32	0-31	1
	$MSB(key1_{i+1})$	8	0-7	1
Укупно на левој страни једначине		14	2-15	64
Укупно на десној страни једначине		22	10-31	1
Заједнички битови		6	10-15	

За $key2_i$ постоји 64 кандидата за битове на позицијама 2-15. Из табеле се види да су шест битова заједнички (на позицијама 10-15) за леву и десну страну једнакости. У просеку 2^{-6} могућих вредности за 14 битова променљиве $key2_i$ могу имати исте битове на позицијама 10-15 као вредност добијена у десној страни једнакости 3.1. Због тога, ако је позната вредност горњих 30 битова $key2_{i+1}$, у просеку остаје једна ($2^{-6} * 64 = 1$) могућа вредност за битове 2-15 вредности $key2_i$. Самим тим, пошто је остала једна могућа вредност за битове на позицијама 2-15 променљиве $key2_i$, а горњих 16 битова су познати (добијени у десној страни једнакости 3.1), онда је познато и свих горњих 30 битова вредности $key2_i$. Дакле, претпостављених горњих 30 битова $key2_{i+1}$ одређује практично једнозначно горњих 30 битова $key2_i$.

На сличан начин добија се 30 битова највеће тежине за $key2_{i-1}$, затим $key2_{i-2}$ и тако даље до $key2_1$.

Добијена листа 30-битних вредности може се комплетирати до 32-битних вредности коришћењем израза $key2_{i+1} = crc32(key2_i, MSB(key1_{i+1}))$ (детаљније у следећем поглављу, стр. 40). Тиме је добијена листа комплетних вредности $key2_n, key2_{n-1}, \ldots, key2_2$ (вредност $key2_1$ се не може комплетно реконструисати).

Наведеним поступком добија се око 2^{22} могућих листа вредности ($key2_n$, $key2_{n-1}, \ldots, key2_2$) од којих се једна заиста појављује у процесу шифровања.

3.2 Редуковање броја могућих вредности key2

Као што ће бити показано, укупан број случајева које треба разматрати у току напада (сложеност напада), је $2^{16} * broj\ mogucih\ listi\ key2_i$ (видети страну 31). Ако број могућих листи $key2_i$ остане 2^{22}, укупна сложеност напада је 2^{38}. Сложеност се може редуковати смањивањем броја потенцијалних листи $key2_i$

(наравно, при том одбацивању појединих листа вредности не сме се одбацити права).

За напад на интерни кључ потребно је знати најмање 13 узастопних бајтова отвореног текста. Ако је познато више од 13 узастопних бајтова, тај вишак познатих бајтова може се искористити за смањивање броја потенцијалних листи $key2_i$.

Напад на интерни кључ креће од 2^{22} могуће вредности $key2_n, (n \geq 13)$. Израчунавају се вредности $key2_{n-1}$, $key2_{n-2}, \ldots, key2_{13}$ користећи једнакост 3.1. У неким случајевима није могуће добити следећу вредност у низу, јер се ниједна од вредности $key2_{i-1}$ не може добити из $key2_i$. Та започета листа не даје ни једну прихватљиву вредност за $key2_{13}$. Што је листа $key2_i$, $i = 13, \ldots, n$ дужа, то се мање кандидата добија за $key2_{13}$ (односно за листу вредности $key2_i$). Поред тога, у неким случајевима различите вредности $key2_i$ дају исту вредност $key2_{i-1}$. Ако се ти дупликати одбаце, број преосталих вредности $key2_{i-1}$ се значајно смањује. Велики број потенцијалних листи отпада, што битно смањује сложеност напада.

Према [2], следећа табела приказује смањивање броја кандидата за $key2_{13}$ (а тиме и сложености напада) у зависности од броја познатих бајтова отвореног текста[1]:

Број познатих бајтова	Број кандидата за $key2_{13}$	Број познатих бајтова	Број кандидата за $key2_{13}$
13	4194304	112	88248
14	3473408	212	43796
15	2152448	312	31088
16	1789183	512	16822
17	1521084	1012	7785
22	798169	2012	5196
27	538746	4012	3976
32	409011	6012	3000
37	332606	8012	1296
42	283930	10012	1857
52	213751	12012	243
62	174471	12301	801

На пример, ако је познато 52 бајта отвореног текста (од тога 13 бајтова се користи за напад, а преостали бајтови за редуковање броја потенцијалних вредности $key2_{13}$), број кандидата за $key2_{13}$ смањује се са 2^{22} на 2^{18}, а сложеност напада смањује се са 2^{38} на 2^{34}. Користећи познатих 10000 бајтова, сложеност напада редукује се на око 2^{27}.

[1] Вредности у табели добијене су експерименталним путем. У сличном експерименту могу се добити другачије вредности, али оне ипак неће много одступати од вредности приказаних у овој табели.

3.3 Одређивање могућих вредности key1

На основу израза $key2_{i+1} = crc32(key2_i, MSB(key1_{i+1}))$ и листе ($key2_n$, $key2_{n-1}$, $\ldots, key2_2$) израчунавају се бајтови највеће тежине листе вредности $key1_i$:

$$MSB(key1_{i+1}) = (key2_{i+1} \ll 8) \oplus crcinvtab[MSB(key2_{i+1})] \oplus key2_i.$$

Тако се добија листа вредности ($MSB(key1_n), MSB(key1_{n-1}), \ldots, MSB(key1_3)$). Како за израчунавање $MSB(key1_i)$ треба знати две узастопне вредности $key2_i$ и $key2_{i-1}$, на овај начин не може се одредити вредност $MSB(key1_2)$.

Претпоставимо да су добијени $MSB(key1_n)$ и $MSB(key1_{n-1})$. Доња 24 бита вредности $key1_n$ нису позната. Они се бирају на 2^{24} начина. Али, нису сви прави кандидати за битове на позицијама 0-23 вредности $key1_n$.
До овог закључка долази на основу везе $key1_i$ и $key1_{i+1}$:

$$key1_{i+1} = (key1_i + LSB(key0_{i+1})) * 134775813 + 1 \ \ (mod\ 2^{32}). \qquad (3.2)$$

Од 2^{24} могућих вредности за $key1_n$ (са фиксираних 8 горњих битова), само је 2^{-8} у одговарајућој вези са вредношћу $key1_{n-1}$ (код које је такође фиксирано 8 битова највеће тежине). Добија се $2^{16}(= 2^{24} * 2^{-8})$ могућности за вредност $key1_n$ и исто толико могућности за $key1_{n-1} + LSB(key0_n)$ (у већини случајева су највиши бајтови $key1_{n-1} + LSB(key0_n)$ и $key1_{n-1}$ једнаки).

Последица једнакости (3.2) је једнакост

$$key1_{n-1} + LSB(key0_n) = (key1_n - 1) * 134775813^{-1} \ \ (mod\ \ 2^{32}). \qquad (3.3)$$

Вредности $key1_{n-1} + LSB(key0_n)$ и $key1_{n-1}$ су веома блиске. Разлика између њих је од 0 до највише 255. Зато $key1_{n-1}$ узима једну од 256 различитих вредности из интервала $\{(key1_n - 1) * 134775813^{-1} - 255\}$ – $\{(key1_n - 1) * 134775813^{-1}\}$. Поставља се питање – коју од вредности из тог интервала узима $key1_{n-1}$ и да ли $key1_{n-1}$ може узети више таквих вредности?

Као што су $key1_n$ и $key1_{n-1}$ повезани изразом (3.2), на аналогни начин повезани су $key1_{n-1}$ и $key1_{n-2}$. $MSB(key1_{n-2})$ је познат. Значи, 8 горњих битова је (практично) фиксирано, па само 2^{-8} могућих вредности $key1_{n-1}$ води ка правој вредности $MSB(key1_{n-2})$. Из претходног пасуса види се да $key1_{n-1}$ може узети 256 различитих вредности, што би значило да у просеку само једна ($1 = 2^{-8} * 256$) вредност $key1_{n-1}$ води ка одговарајућем бајту највеће тежине $key1_{n-2}$. О самој техници одређивања тачне вредности $key1_{n-1}$ биће више речи у наредном поглављу (стр. 42).

Ако се овај поступак настави, добија се листа вредности ($key1_n, key1_{n-1}, ...$, $key1_4$). Ако је позната конкретна вредност $key1_n$, добија се у просеку само једна вредност за $key1_{n-1}$ итд, што значи да нема даљег повећавања броја потенцијалних листа вредности $key1_i$. Укупно за једну конкретну листу ($key2_n, \ldots key2_2$), добија се 2^{16} могућих листи вредности за ($key1_n, \ldots, key1_4$). Укупно има 2^{38} могућих кандидата за ($key2_n, key2_{n-1}, \ldots, key2_2, key1_n, key1_{n-1}, \ldots, key1_4$).

На почетку овог одељка објашњено је зашто се не може добити $MSB(key1_2)$. У низу израчунавања $key1_i$, добијена је вредност $key1_4$. Из ње се, коришћењем формуле (3.3), добија $key1_3 + LSB(key0_4)$. Одатле се закључује да постоји

256 могућности за $key1_3$. Пошто $MSB(key1_2)$ није познат, а та вредност је неопходна за израчунавање $key1_3$, не може се са сигурношћу знати коју од 256 могућих вредности узима $key1_3$.

3.4 Одређивање могућих вредности key0

Из добијене листе вредности $(key1_n, key1_{n-1}, \ldots, key1_4)$ добијају се вредности бајтова најмање тежине за листу вредности $(key0_n, key0_{n-1}, \ldots, key0_5)$. Како се у функцији $update\_keys_i$ (в. стр. 23) користи израз $key1_{i+1} = (key1_i + LSB(key0_{i+1})) * 134775813 + 1 \ \ (mod\ 2^{32})$, бајтови најмање тежине вредности $key0_i$ одређени су изразом:

$$LSB(key0_{i+1}) = ((key1_{i+1} - 1) * 134775813^{-1}) - key1_i \ \ (mod\ 2^{32}) \qquad (3.4)$$

У функцији $update\_keys_i$ користи се израз који повезује две узастопне вредности $key0_i$: $key0_{i+1} = crc32(key0_i, P_i)$, односно

$$key0_{i+1} = (key0_i \gg 8) \oplus crctab[LSB(key0_i) \oplus P_i] \qquad (3.5)$$

Користећи израз 3.4 добијају се $LSB(key0_n), \ldots, LSB(key0_5)$, а затим се на основу 3.5 добијају (на начин који ће бити објашњем) комплетне вредности $key0_i$. Прецизније, тачна вредност $key0_i$ може се одредити ако су познате вредности $LSB(key0_i), LSB(key0_{i-1})$, $LSB(key0_{i-2})$ и $LSB(key0_{i-3}))$. Када се одреди једна вредност $key0_i$, коришћењем израза 3.5 израчунавају се вредности $key0_{i-1}$ и $key0_{i+1}$.

Показаћемо сада како се на основу познатих вредности бајтова најмање тежине променљивих $key0_n$, $key0_{n-1}$, $key0_{n-2}$ и $key0_{n-3}$ одређује тачна вредност $key0_{n-3}$. Заиста, на основу 3.5 добија се:

$key0_n = (key0_{n-1} \gg 8) \oplus crctab[LSB(key0_{n-1}) \oplus P_{n-1}]$, односно
$(key0_{n-1} \gg 8) = key0_n \oplus crctab[LSB(key0_{n-1}) \oplus P_{n-1}]$.

Битови на позицијама 0-7 променљиве $key0_n$ су познати, као и сви битови израза $crctab[LSB(key0_{n-1}) \oplus P_{n-1}]$. Изједначавањем леве и десне стране добијају се битови на позицијама 0-7 вредности $(key0_{n-1} \gg 8)$, односно битови на позицијама 8-15 вредности $key0_{n-1}$. С обзиром да је већ познат бајт најмање тежине вредности $key0_{n-1}$, познати су битови на позицијама 0-15 вредности $key0_{n-1}$. Аналогним поступком добијају се битови на позицијама 0-23 вредности $key0_{n-2}$, односно комплетна вредност (битови на позицијама 0-31) променљиве $key0_{n-3}$.

Полазећи од добијене вредности $key0_{n-3}$, на основу 3.5 добијају се преостали битови вредности $key0_{n-2}, key0_{n-1}$ и $key0_n$, а затим се на основу израза $key0_i = crc32^{-1}(key0_{i+1}, P_i)$ добијају и вредности $key0_{n-4}, key0_{n-5}, \ldots, key0_1$.

Из познатих вредности $LSB(key0_i), (i = n-3, \ldots, n)$ добијени су кандидати за листу вредности $(key0_n, key0_{n-1}, \ldots, key0_1)$. Са друге стране познати су (а неискоришћени) $LSB(key_{n-4}), LSB(key0_{n-5}), \ldots, LSB(key0_5)$. Те вредности се користите за одбацивање свих погрешних кандидата за листу вредности $(key0_n, key0_{n-1}, \ldots, key0_1)$. Кад се одбаце све погрешне вредности, остаје само једна листа вредности $key0_i$ - она која се заиста користи у шифровању.

Ако се упореде познате вредности бајтова најмање тежине $key0_{n-4}, key0_{n-5}$, $key0_{n-6}$ и $key0_{n-7}$ са одговарајућим бајтовима из добијене потенцијалне листе вредности за $key0_i$, добија се подударање у 2^{-32} случаја. С обзиром да (у најгорем случају) постоји 2^{38} кандидата, ово не гарантује добијање јединствене листе, па се мора извршити упоређивање и бајта најмање тежине $key0_{n-8}$. У овом случају добија се подударање у 2^{-40} случаја, па ово гарантује добијање јединствене (праве) листе вредности за $key0_i$, а самим тим и за $key1_i$ и $key2_i$.

Како позната листа вредности $LSB(key0_i)$ креће од вредности $i = 5$, добија се $n - 8 = 5$, тј. $n = 13$. То значи да је потребно знати бар 13 бајтова отвореног текста да би се са сигурношћу одредило стање коначног аутомата $\mathcal{K}$ у неком тренутку.

С обзиром на то да је $crc32$ линеарна функција, приликом одређивања вредности $key0_i$ (у зависности од $key1_i$) програм не добија на сложености и не проширује се број листа које треба испитати. Да би се са сигурношћу одредило стање $\mathcal{K}$ у неком тренутку, довољно је знати 13 узастопних бајтова отвореног текста и испитати максимално 2^{38} могућих кандидата.

3.5 Одређивање интерног кључа

Ако су познате вредности $key0_j, key1_j$ и $key2_j$ (стање $\mathcal{K}$ у неком тренутку), могу се добити вредности $key0_i, key1_i$ и $key2_i$ за свако i, $i = 1, \ldots, j$ (сва претходна стања $\mathcal{K}$ у процесу шифровања), коришћењем само података из шифрованог фајла. Не мора бити познат ниједан бајт нешифрованог фајла. Добијају се сви бајтови нешифрованог фајла, добија се 12 бајтова који чине шифарско заглавље и на крају добија се интерни кључ ($key0_1, key1_1, key2_1$). Заиста, ако су познате вредности $key0_{i+1}, key1_{i+1}$ и $key2_{i+1}$, следећим поступком добијају се вредности $key0_i, key1_i$ и $key2_i$:

$$\begin{aligned}
key2_i &= crc32^{-1}(key2_{i+1}, MSB(key1_{i+1})) \\
key1_i &= ((key1_{i+1} - 1) * 134775813^{-1}) - LSB(key0_{i+1}) \ \ (mod\, 2^{32}) \\
temp_i &= key2_i \mid 3 \\
key3_i &= LSB((temp_i * (temp_i \oplus 1)) \gg 8) \\
P_i &= C_i \oplus key3_i \\
key0_i &= crc32^{-1}(key0_{i+1}, P_i)
\end{aligned} \tag{3.6}$$

Интерни кључ не зависи од садржаја шифрованог фајла. Такође не зависи ни од 12 бајтова који чине шифарско заглавље. Он зависи само од лозинке којом се врши шифровање. Када се добије интерни кључ, комплетан фајл се може дешифровати. Ако су два различита фајла шифрована истом лозинком, њихов интерни кључ је исти. Ако једна архива садржи више фајлова који су шифровани истом лозинком, проналажењем интерног кључа за било који фајл, могу се дешифровати и сви остали фајлови садржани у тој архиви, иако се о њима не зна апсолутно ништа. Чак се могу дешифровати и фајлови било које друге архиве (ако се подударају компресија и верзија програма) која је шифрована истом лозинком.

3.6 Одређивање лозинке

Добијањем интерног кључа, могуће је комплетно дешифровати фајл. Може се ићи и корак даље и доћи до конкретне лозинке којом је шифрован дати фајл. То наравно повећава трајање извршавања програма.

Алгоритам за налажење лозинке испробава различите дужине лозинки, почевши од 0, 1, 2,... па све до неке максималне вредности и у зависности од дужине лозинке (l), поступа на један од следећих начина:

Ако је $l \leq 4$, неки битови вредности $key0_0$, $key0_{-1}$, $key0_{-2}$ и $key0_{-3}$ могу се директно добити из вредности $key0_1$ користећи формуле $key0_{i-1} = crc32^{-1}(key0_i, char)$. Ако је лозинка дужине нула, $key0_1$ мора имати вредност $0x12345678$. Ако је лозинка дужине један, $char$ се поставља на нулу ($char$ утиче само на бајт најмање тежине), примењује се горња формула (за i=1) и добија се $key0_0$ чија се горња три бајта (они су добијени једнозначно из $key0_1$) морају подударати са горња три бајта вредности $0x12345678$. Бајт најмање тежине $key0_0$ се упоређује са вредношћу $0x12345678$ коришћењем функције xor и добија се потенцијална вредност лозинке (детаљно објашњење на страни 45). Аналогним поступком добијају се вредности $key0_{-1}$, $key0_{-2}$ и $key0_{-3}$ и потенцијалне вредности лозинке. У сваком од ових случајева проверава се да ли је добијена лозинка која се заиста користи при датом шифровању. Врши се први корак шифровања (в. стр. 23) и проверава се да ли се добија интерни кључ. Ако то јесте случај, пронађена је права лозинка, а у супротном прелази се на испитивање лозинки следеће дужине.

Ако је $5 \leq l \leq 6$, применом израза 3.6 могу се директно израчунати $key1_0$, $key2_0$ и $key2_{-1}$. Вредности $key2_{2-l}, \ldots, key2_{-2}$ могу се одредити јер су $key2_i$ и $key2_{i+1}$ повезани линеарном функцијом $crc32$ (слично као $key0_i$ и $key0_{i+1}$). Четири пута се примењује израз 3.1, с том разликом што се уместо $MSB(key1_{i+1})$ користи вредност $0x00$, ($MSB(key1_{i+1})$ утиче само на бајт најмање тежине, док на преостала три бајта $key2_i$ нема утицаја), упоређује се са иницијалном вредношћу $key2(0x34567890)$ и коришћењем особина функције xor налазе се потенцијални кандидати за $MSB(key1_{2-l}), \ldots, MSB(key1_{-1})$. Затим се врши комплетирање листа вредности $key1_i$ (в. стр. 29), а одатле се добијају $LSB(key0_{2-l})$ и $LSB(key0_{3-l})$. На основу тих вредности, добија се пети (тј. пети и шести, за лозинке дужине шест) карактер лозинке (начин детаљно објашњен на страни 45). Прва четири карактера лозинке налазе се на исти начин као при тражењу лозинке дужине четири. Када је пронађен кандидат за лозинку, шифровањем се проверава да ли је то исправна лозинка (као код лозинке дужине ≤ 4).

Ако је $l > 6$, првих $l-6$ карактера лозинке варирају се на све могуће начине, а за преосталих шест карактера примењује се поступак као при тражењу лозинке дужине шест. Првих $l-6$ карактера лозинке може се изабрати на укупно $2^{8*(l-6)}$ начина.

У сваком од ова три случаја, не мора се добити она лозинка која је заиста употребљена при шифровању. Може се добити и еквивалентна лозинка у смислу да се њеним коришћењем добија исти интерни кључ. Оваквом лозинком може се дешифровати фајл исто као и коришћењем оригиналне лозинке. Овај случај

често се среће при лозинкама дужим од 12 бајтова. Тада најчешће постоји еквивалентна лозинка дужине до 12 бајтова, јер је интерни кључ дугачак само 12 бајтова и нема простора за већи број комбинација.

3.7 Могућности за напад без познавања отвореног текста

Пошто се ради о нападу са познатим отвореним текстом, основни проблем је: како доћи до 13 бајтова отвореног текста, потребних за напад? Ако је познат тип фајла који шифрован, може се доћи до неколико познатих бајтова. На пример, за сваки .pdf фајл познато је првих седам бајтова $0x\ 25\ 50\ 44\ 46\ 2d\ 31\ 2e$ = (%PDF-1.). Они показују да је то заиста .pdf фајл и дају почетни број верзије (све досадашње верзије означаване су са 1.x). Слично, сваки .html фајл почиње са $< html >$ (познато првих шест бајтова) или са $<!DOCTYPE\ HTML$ (познато првих 14 бајтова). Стеј (Michael Stay [6]) је показао како се може доћи до неких познатих бајтова отвореног текста. Такође је показао да се може извршити напад и са мање од 13 познатих бајтова (али по цену дужег времена рада програма), као и да ако у архиви има више фајлова шифрованих истом лозинком, онда је потребно мање познатих бајтова отвореног текста.

Постоје и други начини за откривање садржаја шифрованих архива. Стивенс (Mike Stevens) и Фландерс (Ellisa Flanders) су у раду [7] показали како се може извршити напад одређивањем 12 бајтова који чине шифарско заглавље. Кохно (Tadayoshi Kohno) је у раду [4] показао још неке начине напада на шифроване архиве. Може се добити пуно информација о фајловима из заглавља саме архиве: име фајла, његова екстензија, CRC, дужина фајла, метод компресије, време последње модификације, ... Сви ови подаци могу се искористити за напад на шифровани фајл. Затим се могу искористити неке везе између компримовања и шифровања, као и веза имена фајла и програма који користи тај фајл. Такође, може се извршити успешан напад ако архива истовремено садржи шифроване и нешифроване фајлове. У зависности од методе шифровања која се користи, врше се напади на шифрован фајл користећи неке специфичности својствене само том конкретном алгоритму. Нпр. Бихам је у раду [1] показао како се могу искористити неке слабости при шифровању алгоритмом DES.

Поглавље 4

Реализација напада

Да би се проверио описани напад, реализована су три програма: *sifrovanje*, *desifrovanje* и *pkzipcrack*. Програм sifrovanje трансформише нешифровану zip архиву у шифровану zip архиву, ако је позната лозинка. Програм desifrovanje трансформише шифровану zip архиву у нешифровану zip архиву, ако је позната лозинка. Најважнији је програм pkzipcrack који на основу шифроване архиве и довољног броја познатих бајтова отвореног текста проналази интерни кључ, а ако корисник то захтева — и лозинку којом је шифрована архива.

С обзиром да ови програми доста користе операције на нивоу појединачних битова и бајтова, коришћен је програмски језик C. Програми су развијани под Windows-ом, коришћењем Microsoft Visual C++ окружења (верзија 6.0), али би се уз мање измене могли компајлирати и под Linux-ом. Сва тестирања вршена су на персоналном рачунару са процесором од 2,4 GHz, 512 MB меморије и Windows Xp Professional оперативним системом.

Сви програми претпостављају да се у архиви налази само један фајл. Лако се може извршити модификација, тако да програми раде и са већим бројем фајлова у једној архиви.

4.1 Шифровање и дешифровање

Ови програми трансформишу нешифровану архиву у шифровану архиву и обрнуто. Лозинка за (де)шифровање је позната. Програми се позивају из командне линије са три аргумента.
За sifrovanje:
име нешифроване архиве, име шифроване архиве која ће се добити, лозинка.
За desifrovanje:
име шифроване архиве, име нешифроване архиве која ће се добити, лозинка.
Излаз из програма sifrovanje је шифрована архива која се може дешифровати коришћењем програма Pkunzip или било ког другог Windows-овог деархивера. Излаз из програма desifrovanje је нешифрована (али компримована) архива која се може отворити помоћу програма Pkunzip или коришћењем било ког

другог деархивера. На тај начин проверена је исправност ова два програма.

Шифровање и дешифровање се упоредо обрађују јер користе веома сличан алгоритам (в. стр. 23) и исте променљиве. Учитавају се шифрована, односно нешифрована архива (имена тих архива су аргументи командне линије), иницијализују се табеле crc32tab и invcrc32tab које одговарају табелама crctab и crcinvtab (в. стр. 26), иницијализују се вредности key[0]-key[3] и комбинују са добијеном лозинком функцијом init_keys(лозинка) (видети први корак шифровања, стр. 23). Већина поља из заглавља фајла (в. стр. 12), заглавља фајла централног директоријума (в. стр. 16) и краја централног директоријума (в. стр. 17) само се преписује из једног у други фајл. Мењају се следећа поља:

- флегови (из заглавља фајла и из заглавља фајла централног директоријума) – код шифровања бит 00 се мења са нуле на јединицу, а код дешифровања се са јединице мења на нулу;

- величина компримованог фајла (из заглавља фајла и из заглавља фајла централног директоријума)– код шифровања се повећава за 12 због додавања шифарског заглавља , а код дешифровања се смањује за 12 због одбацивања шифарског заглавља;

- почетак централног директоријума (из краја централног директоријума) – код шифровања се повећава за 12, а код дешифровања се смањује за 12 због додавања тј. уклањања шифарског заглавља.

Код шифровања се додаје заглавље шифровања – 11 бајтова (добијених применом функције $rand()$) и 12. бајт који се чита из поља CRC-32 (бајт највеће тежине, из заглавља фајла). Ти бајтови се шифрују (коришћењем функције xor), мења се стање коначног аутомата $\mathcal{K}$ (коришћењем функције $update$) и уписују се у шифровану архиву. Затим се учитава један по један знак из нешифроване архиве, шифрује се, мења се стање $\mathcal{K}$ и шифровани знак се уписује у шифровану архиву.

Код дешифровања се из шифроване архиве читају 12 бајтова који представљају шифарско заглавље, дешифрују се и мења се стање $\mathcal{K}$. Првих 11 дешифрованих карактера се одмах одбацују (битни су само за промене стања $\mathcal{K}$), а 12. карактер се упоређује са бајтом највеће тежине поља CRC-32. Ако се подударају, то је знак да је вероватно употребљена права лозинка и прелази се на дешифровање самог фајла, а у супротном се прекида програм. Затим се учитава један по један знак из шифроване архиве, дешифрује се, мења се стање $\mathcal{K}$ и дешифровани знак се уписује у нешифровану архиву.

За потребе шифровања и дешифровања написане су функције $ucitaj2$, $ucitaj4$, $ispis2$ и $ispis4$. Функције $ucitaj2(FILE*)$ и $ucitaj4(FILE*)$ из фајла (који се задаје као аргумент функције) учитавају један двобајтни цео број (short int), односно један четворобајтни цео број (int). При читању вишебајтног броја из фајла, прво се учитава бајт најмање тежине, а затим и остали бајтови у обрнутом редоследу у односу на њихов поредак у самом броју. Зато се из фајла чита један по један бајт, а затим се коришћењем померања битова и битске операције or формира прави број.

Функције $ispis2(FILE*, short\ int)$ и $ispis4(FILE*, int)$ у фајл (задат као први аргумент функције) уписују двобајтни, тј. четворобајтни цео број (задат као други аргумент функције). Прво се уписује бајт најмање тежине, а затим и остали бајтови у обрнутом редоследу у односу на њихов поредак у самом броју.

4.2 Програмска реализација напада

Програм pkzipcrack из познате шифроване архиве (шифроване помоћу програма pkzip 2.04 или помоћу програма sifrovanje) и одређеног броја бајтова отвореног текста проналази интерни кључ, а ако корисник то захтева — и лозинку којом је извршено шифровање. Програм се позива из командне линије са три аргумента: име шифроване архиве; име одговарајуће нешифроване архиве; број бајтова отвореног текста. Број бајтова отвореног текста мора бити већи или једнак 13 (са мањим бројем познатих бајтова није могуће извршити комплетан напад), а не може бити већи од величине нешифрованог фајла. У случају да ови услови нису испуњени, програм одмах завршава рад, уз испис одговарајуће поруке. Ако се чита тачно 13 нешифрованих бајтова, програм веома дуго ради. Ако је на располагању више од 13 бајтова, најпре се врши редукција броја кандидата за листу вредности key2 (помоћу вишка познатих бајтова), а затим се са преосталих 13 бајтова врши напад и програм брже даје резултате (о анализи ефикасности рада програма више на стр. 47). Претпоставља се да се у архивама налази само по један фајл.

Програм се извршава у неколико етапа. За сваку етапу написана је једна или више функција.

На почетку се отварају (само за читање) шифрована и нешифрована архива. Функција $provera\_fajlova(\ )$ чита заглавља оба фајла (садржана у обе архиве) и врши неопходне провере: да ли су у питању zip архиве, да ли је један фајл шифрован, а други није, да ли се поклапају дужина имена, име, величина компримованог фајла, величина некомпримованог фајла, итд. Ако нека од провера не прође успешно, програм прекида рад уз одговарајућу поруку, а у супротном наставља рад и учитава шифарско заглавље (уписује га у глобалну променљиву $random\_bajtovi[\,]$).

Учитавају се један по један знак из шифрованог и њему одговарајућег нешифрованог фајла и формирају се стрингови $sifr\_podaci$ и $nesifr\_podaci$. Позива се функција $initcrc()$ која иницијализује табеле $crc32tab$ и $invcrc32tab$ (функција је описана на стр. 26).

После тога, позива се функција $zip\_crack(char*, char*, unsigned\ int)$ која одређује вредност интерног кључа. Као аргументе функција добија стрингове $sifr\_podaci$, $nesifr\_podaci$ и $koliko\_bajtova$ - заједничку дужину тих стрингова.

У функцији zip_crack најпре се позивом функције $init\_key2tab()$ формирају табеле неопходне за ефикасније рачунање — матрице $key2tab$, $broj\_povezanih\_key2$ и тродимензиони низ ima_jos_key2.

- Матрица $key2tab$ је димензија 256 x 64. У њој редни број врсте представља могућу вредност $key3$. У свакој врсти налази се низ 64 различите комбинације битова на позицијама 2-15 променљиве $key2$. Свака од тих

комбинација помоћу $temp = key2|3$; $key3 = LSB(temp*(temp \oplus 1)) \gg 8$ даје ону вредност $key3$ која је једнака редном броју те врсте. Зато се узимају све могуће комбинације за битове 2-15 променљиве $key2$, проверава се коју вредност $key3$ они дају и смештају се у ону врсту чији је редни број једнак вредности добијене $key3$. Тиме је завршена иницијализација матрице $key2tab$.

- Тродимензиони низ $ima\_jos\_key2[key3_{i-1}][koji\_bitovi][koliko\_ima]$ је димензије 256 x 64 x 5. У листи вредности $key2_i$, вредности $key2_i$ и $key2_{i-1}$ су повезане изразом

$$key2_{i-1} = (key2_i \ll 8) \oplus invcrc32tab[MSB(key2_i)] \oplus MSB(key1_i).$$

  Као што је речено (стр. 26) користи се једнакост битова на позицијама 10-15 леве и десне стране. На основу тих шест битова, из вредности $key2_i$, могу се добити потенцијални кандидати за $key2_{i-1}$ (не увек јединствени). Управо за то служи тродимензиони низ ima_jos_key2: он даје све могуће кандидате из табеле $key2tab$ који се налазе у врсти $key3_{i-1}$, а битови на позицијама 10-15 су исти као $koji\_bitovi$. Трећа димензија ($koliko\_ima$) овог низа показује који је редни број тог кандидата. Провером свих случајева закључено је да оваквих кандидата никада нема више од пет, што значи да $koliko\_ima$ узима вредности из интервала $[0-5]$. Вредности овог тродимензионалног низа су редни бројеви елемената табеле $key2tab$ (из врсте $key3_{i-1}$) који имају исте вредности битова 10-15 као $(key2_i \ll 8) \oplus invcrc32tab[MSB(key2_i)] \oplus MSB(key1_i)$. Прва димензија овог тродимензионалног низа мора бити 256, јер $key2tab$ има 256 врста, а друга димензија низа је 64, јер за 6 битова (на позицијама 10-15) постоје 2^6 могућих комбинација.

- Матрица $broj\_povezanih\_key2[key3][bitovi]$ је димензије 256 x 64. Она показује колико има вредности у низу $ima\_jos\_key2[key3][bitovi]$ (различитих од нуле).

Матрице ima_jos_key2 и $broj\_povezanih\_key2$ се истовремено иницијализују. Постављају се све вредности на нуле, а затим се за сваку могућу вредност $key3$ (0-255) и сваку могућу комбинацију битова на позицијама 10-15 променљиве $key2$ (0-63), одређују редни бројеви елемента врсте матрице $key2tab$ (редни број врсте је $key3$) који има одговарајућу комбинацију битова на позицијама 10-15 и уписују се у низ ima_jos_key2. Истовремено се (за сваку унету вредност у низ ima_jos_key2) повећава за један вредност одговарајућег елемента у $broj\_povezanih\_key2$. Комплетна иницијализација описана је следећим кодом:

```
for(key3 = 0; key3 < 256; key3 ++)
  for(pom = 0; pom < 64; pom ++)
  {
    ima_jos_key2[key3][(key2tab[key3][pom] >> 10)&63]
        [broj_povezanih_key2[key3][(key2tab[key3][pom] >> 10)&63]] = pom;
```

$broj\_povezanih\_key2[key3][(key2tab[key3][pom] \gg 10)\&63]++;$
}

По завршетку иницијализације ових табела неопходних за рад програма, алоцира се меморија за $key0list$, $key1list$, $key2list$ и $key3list$. Ово су променљиве које садрже могуће листе вредности $key0_i$, $key1_i$, $key2_i$ и $key3_i$, $i = 1, \ldots, 13$.

На основу $sifr\_podaci$ и $nesifr\_podaci$ формира се листа вредности $key3list_i$:

$$key3list[i] = sifr\_podaci[i] \oplus nesifr\_podaci[i].$$

4.2.1 Одређивање могућих листи key2

Одређивање могућих листи key2 остварује се позивом функције $generisi\_prvi\_skup\_key2(n)$ која узима све могуће вредности $key3[n]$ и $key3[n{-}1]$, $n{=}argv[3]$ и формира листу "перспективних" вредности $key2[n]$, тј. оних вредности које се могу добити од неке вредности $key2[n{-}1]$. Све "перспективне" вредности убацују се у низ $key2n[\,]$. Из табеле $key2tab$ (из врсте чији је редни број једнак вредности $key3[n]$) узимају се све 64 могућности за битове на позицијама 2-15, а затим се свакој од њих додају на 2^{16} начина вредности за битове на позицијама 16-31, тако да се добија 2^{22} могућности за битове на позицијама 2-31 вредности $key2[n]$. Ако се узме једна фиксирана вредност за горњих 30 битова $key2[n]$, применом израза 3.1 добија се $key2[n{-}1]$. Може се узети да је $MSB(key1list[n]) = 0$, јер та вредност не утиче на битове на позицијама 10-15 вредности $key2[n{-}1]$ која се добија. Локална променљива $bitovi\_6$ узима добијене битове на позицијама 10-15 вредности $key2[n{-}1]$ (битови на позицијама 0-9 и 16-31 добијене вредности $key2[n{-}1]$ се постављају на нуле и врши се померање удесно за 10 места). Вредност променљиве $bitovi\_6$ се користи као друга димензија при коришћењу тродимензионог низа ima_jos_key2, као и табеле $broj\_povezanih\_key2$. За сваку конкретну вредност $key2[n]$ проверава се да ли се може добити од неке вредности $key2[n{-}1]$ (и ако може, од колико различитих вредности). Тај број се чита из $broj\_povezanih\_key2[key3list[n{-}1]][bitovi\_6]$. Ако постоји неки $key2[n{-}1]$ из ког се може добити $key2[n]$, онда је та вредност $key2[n]$ "перспективна" и убацује се у низ $key2n$. Истовремено се врши и подешавање битова на позицијама 0 и 1. Они се добијају из могућих вредности $key2[n{-}1]$ на основу израза $((key2tab[key3list[n{-}1]][ima\_jos\_key2[key3list[n{-}1]][bitovi\_6][k]] \oplus invcrc32tab[prvi\_key2 \gg 24]) \gg 8)\&3$ (детаљно објашњење следи у делу који описује редукцију броја кандидата за $key2[n]$). Повећава се (за један) број елемената низа $key2n$ (та информација се чува у глобалној променљиви $broj\_key2n$). Кад се испитају све могуће вредности за горњих 30 битова $key2[n]$, исписује се колико је комплетних "перспективних" вредности $key2[n]$ пронађено, тј. колико елемената има низ $key2n$.

За напад је потребно знати све могуће вредности $key2[12]$. Ако је познато тачно 13 бајтова отвореног текста, све могуће вредности се већ налазе у $key2n$. Ако је познато више од 13 бајтова отвореног текста, врши се редукција броја потенцијалних вредности $key2[12]$ и у том случају програм брже завршава са радом.

Редукција се врши у функцији $redukcija\_key2n(i)$ на основу $n{-}13$ познатих

бајтова отвореног текста. Ако су у низу $key2n$ садржани кандидати за $key2[i]$, за сваки елемент низа $key2n$ на основу $key3[i-1]$ и $key3[i-2]$ израчунава се $key2[i-1]$. Вредност $key2[i-1]$ се добија у три корака:

а) $key2[i-1]$(битови 8-31) $= (key2[i] \ll 8) \oplus invcrc32tab[MSB(key2[i])]$;

б) $key2[i-1]$(битови 2-7) = битови 2-7 свих вредности $key2tab[key3[i-1]][k]$ који дају $key3[i-1]$;

ц) $key2[i-1]$(битови 0-1) = битови 8-9 свих вредности $(key2tab[key3[i-2]][k] \oplus invcrc32tab[MSB(key2[i-1])])$ који дају $key3[i-2]$.

За једну фиксирану вредност $key2[i]$, битови 8-31 у првом кораку (а) добијају се директно на основу 3.1 и то на јединствен начин. Вредност $MSB(key1[i])$ не утиче на битове на позицијама 8-31, па се узима да је његова вредност једнака нули.

Битови 2-7 у кораку (б) могу имати више различитих вредности. Број тих могућности је

$$broj\_povezanih\_key2[key3list[i-1]][(key2\_i\_minus\_1 \,\&\, 0xfc00) \gg 10],$$

где $key2\_i\_minus\_1$ представља вредност $key2[i-1]$ добијену у кораку (а). Операцијом $(key2\_i\_minus\_1 \,\&\, 0xfc00) \gg 10$ постиже се да су сви битови, осим битова на позицијама 10-15, једнаки нули (они су једнозначно добијени у кораку (а)). Померањем за десет места удесно постиже се да се тих шест битова налазе на позицијама 0-5. Они представљају другу координату за низ $broj\_povezanih\_key2$. У петљи се проналазе све могуће комбинације за битове на позицијама 2-7 вредности $key2[i-1]$. У свакој итерацији (може их бити од нула до највише пет) узима се једна по једна вредност из $key2tab[key3list[i-1]]$ која има исте битове на позицијама 10-15 као $key2[i-1]$ (битови 8-31 ове вредности добијене су у претходном кораку). На већ добијене битове 8-31 додају се сви могући повољни битови 2-7.

У кораку (ц) на основу познатих битова на позицијама 2-31 вредности $key2[i-1]$, на основу 3.1, добијају се битови на позицијама 10-31 вредности $key2[i-2]$ (као и у првом делу, претпоставља се да је $MSB(key1list[i-1]) = 0$, јер не утиче на битове на позицијама 8-31). Трансформацијом израза 3.1 добија се

$key2[i-2] = (key2[i-1] \ll 8) \oplus invcrc32tab[MSB(key2[i-1])] \oplus MSB(key1[i-1])$,
односно

$(key2[i-1] \ll 8) = key2[i-2] \oplus invcrc32tab[MSB(key2[i-1])] \oplus MSB(key1[i-1])$,
или

$key2[i-1] = (key2[i-2] \oplus invcrc32tab[MSB(key2[i-1])] \oplus MSB(key1[i-1])) \gg 8$.

У овом изразу непознати су само битови на позицијама нула и један вредности $key2[i-1]$. Пошто се врши померање удесно за осам места, довољно је пронаћи вредности на позицијама осам и девет израза $key2[i-2]) \oplus invcrc32tab[MSB(key2[i-1])]$ (вредност $MSB(key1[i-1])$ не утиче на битове осам и девет, па се изостављa). Вредност $ima\_jos\_key2[key3list[i-2]][key2\_i\_minus\_2][s]$ (где $key2\_i\_minus\_2$ представља битове на позицијама 10-15 добијене вредности

$key2[i{-}2]$, а s редни број итерације) даје редне бројеве елемената табеле $key2tab$ (у врсти $key3list[i{-}2]$) који дају одговарајући $key2[i{-}1]$. Познати су битови 2-15 вредности $key2tab[key3list[i{-}2]][ima\_jos\_key2[key3list[i{-}2]][key2\_i\_minus\_2][s]]$ - они одговарају могућностима за $key2[i-2]$. Вредност $invcrc32tab[MSB(key2[i{-}1])]$ је позната, па су познати битови 2-15 вредности израза $key2[i{-}2] \oplus invcrc32tab[MSB(key2[i{-}1])]$. Померањем за осам места удесно добијају се битови на позицијама 0-7 израза $(key2[i{-}2] \oplus invcrc32tab[MSB(key2[i{-}1])] \oplus MSB(key1[i{-}1])) \gg 8$. Овим поступком добијени су сви могући, одговарајући битови на позицијама нула и један вредности $key2[i{-}1]$, а самим тим добијена је и комплетна вредност $key2[i{-}1]$.

Све добијене, комплетне вредности $key2[i{-}1]$ уносе се у низ $key2n\_1$. Овај низ (због сигурности) мора бити веће димензије од 2^{22} јер се за сваку познату вредност $key2[i]$ добија у просеку једна одговарајућа вредност $key2[i{-}1]$, али само у просеку. За неке вредности $key2[i]$ не добија се ни једна вредност $key2[i{-}1]$, док се за неке друге вредности $key2[i]$ добија чак и по пет одговарајућих вредности $key2[i{-}1]$. Зато број елемената низа $key2n\_1$ (који се чува у глобалној променљиви $broj\_key2n\_1$) може бити већи или мањи од броја елемената низа $key2n$.

У добијеном низу $key2n\_1$, који садржи могуће вредности $key2[i{-}1]$, има поновљених вредности. Зато се позива функција $sortiranje()$ која врши сортирање низа $key2n\_1$ применом quicksort алгоритма. Елементи низа $key2n\_1$ преписују се у низ $key2n$, при чему се сви елементи низа $key2n\_1$ који имају исте вредности преписују само једном (одбацују се дупликати). Тим поступком се смањује број потенцијалних кандидата за $key2[i{-}1]$ у односу на број кандидата за $key2[i]$.

Наведени корак редукције примењује се $argv[3]{-}13$ пута и добијају се сви потенцијални кандидати за $key2[12]$. Редукцијом се значајно смањује број кандидата за $key2[12]$ (видети табелу на страни 28).

Кад су добијени сви могући кандидати за $key2[12]$, прелази се на формирање потенцијалних листа вредности $key2list[i], i = 0, \ldots, 12$. У свакој итерацији петље позива се функција $formiraj\_listu\_key2(key2n[j])$, која добија као агумент једну по једну могућу вредност $key2[12]$, $j = 0, \ldots, broj\_key2n - 1$ (из низа $key2n$). Формира се једна по једна потенцијална листа вредности $key2list[i], i = 2, \ldots, 12$. У функцији $formiraj\_listu\_key2()$ се поставља вредност $key2list[12]$ и позива се рекурзивна функција $key2\_rekurzija()$ у којој се у сваком кораку рекурзије формира један по један елемент $key2list[i]$.

Функција $key2\_rekurzija(i)$ на основу познатих $key2list[i]$, $key3list[i{-}1]$ и $key3list[i{-}2]$, израчунава $key2list[i{-}1]$, а такође се добија и $MSB(key1list[i])$. $key2list[i-1]$ се добија у три корака на исти начин као код редукције броја вредности $key2[n]$ (описано на претходној страни). Ако се добије да је једна вредност $key2list[i]$ повезана са више од једне $key2list[i{-}1]$, а те вредности су исте (из више различитих елемената једне исте врсте табеле $key2tab$ се могу добити исте вредности $key2list[i{-}1]$), те двоструке вредности се занемарују и одмах се прелази на тражење следећег елемента у листи $key2list[i]$. У овој функцији се такође из две узастопне, комплетно познате вредности $key2list[i]$ и $key2list[i{-}1]$, израчунава бајт највеће тежине вредности $key1list[i]$ ($MSB(key1list[i])$). По-

лазећи од 3.1 добија се

$$MSB(key1list[i]) = (key2[i] \ll 8) \oplus invcrc32tab[MSB(key2list[i])] \oplus key2[i{-}1].$$

Када су добијене $key2list[i{-}1]$ и $MSB(key1list[i])$, рекурзивно се позива функција $key2\_rekurzija(i{-}1)$ која проналази $key2list[i{-}2]$ и $MSB(key1list[i{-}1])$. Ако је формирана комплетна листа $key2list[2] - key2list[12]$, излази се из рекурзије и позива функција $formiraj\_listu\_key1()$ која формира све могуће листе $key1list[i]$.

4.2.2 Одређивање могућих листи key1

На основу познатих $MSB(key1list[3]), \ldots, MSB(key1list[12])$ формирају се све могуће листе вредности $key1list[i]$, $i = 4, \ldots, 12$.

Функција $formiraj\_listu\_key1()$ у петљи узима све могуће вредности за непозната 24 бита на позицијама 0-23 (битови 24-31 су познати) вредности $key1list[12]$. Не дају све 2^{24} могућности одговарајућу вредност $key1list[11]$, већ само њих 2^{16}. Вредности $key1list[12]$ и $key1list[11]$ су повезане изразом 3.3. Ако је позната конкретна вредност $key1list[12]$, израчунава се вредност $key1\_n$: $key1n\_1 = (key1list[12] - 1) * 3645876429$, где 3645876429 представља 134775813^{-1}. Добијена је вредност $key1list[11] + LSB(key0list[12]) = key1\_n$. Ако се подударају вредности бајтова највеће тежине вредности $key1n\_1$ и $key1list[11]$ или се поклапају бајтови највеће тежине вредности $key1n\_1{-}255$ и $key1list[11]$, наставља се даље и тражи се комплетна вредност $key1list[11]$ позивом функције $key1\_rekurzija(12)$, а у супротном се прелази на испитивање следеће могуће вредност $key1list[12]$. Бајтови највеће тежине се разликују највише за један, па се зато испитују само две варијанте ($key1n\_1$ и $key1n\_1{-}255$).

Рекурзивна функција $key1\_rekurzija(i)$ коришћењем израза 3.3, а на основу познате комплетне вредности $key1list[i]$ и познатих $MSB(key1list[i{-}1])$ и $MSB(key1list[i{-}2])$ израчунава комплетну вредност $key1list[i{-}1]$. Ако је позната комплетна вредност $key1list[i]$, може се добити комплетна вредност $(key1list[i] - 1) * 3645876429$. Бројеви $key1list[i{-}1]$ и $key1list[i{-}1] + LSB(key0list[i])$ се могу разликовати једино за неку вредност из интервала $0 - 255$. Пошто је то бајт најмање тежине, када се та разлика пренесе до бајта највеће тежине, они се разликују највише за један. Зато се проверава подударање бајта највеће тежине вредности добијене помоћу $(key1list[i] - 1) * 3645876429$ са бајтом највеће тежине вредности $key1list[i{-}1]$ и $key1list[i{-}1]+255$. Ако се бајтови највеће тежине поклапају, прелази се на израчунавање комплетне вредности $key1list[i{-}1]$. Локална променљива $key1\_i\_minus\_1$ добија вредност $(key1list[i] - 1) * 3645876429$, а та вредност је једнака $key1list[i{-}1] + LSB(key0list[i])$. Тачна вредност $key1list[i{-}1]$ узима вредност из интервала $[key1\_i\_minus\_1 - (key1\_i\_minus\_1 - 255)]$. У петљи се узимају све могуће вредности из тог интервала и утврђује се која од њих даје одговарајућу вредност $MSB(key1[i{-}2])$, која је већ позната; за то се користи израз

$$key1list[i{-}2] + LSB(key0list[i{-}1]) = (key1list[i{-}1] - 1) * 3645876429.$$

За сваку од 256 могућих вредности $key1list[i{-}1]$, израчунава се $(key1list[i{-}1] - 1) * 3645876429$, а затим се упоређује добијени бајт највеће тежине са $MSB(key1list[i{-}2])$ и $MSB(key1list[i{-}2] + 255)$. Када се добије поклапање, пронађено је коју тачно вредност узима $key1list[i{-}1]$, а истовремено је пронађена вредност бајта најмање тежине вредности $key0list[i]$: $LSB(key0list[i]) = (key1[i] - 1) * 3645876429 - key1list[i{-}1]$.

Када је пронађена комплетна вредност $key1list[i{-}1]$, рекурзивно се позива функција $key1\_rekurzija(i{-}1)$ која израчунава комплетну вредност $key1list[i{-}2]$ и $LSB(key0list[i{-}1])$. Кад се формирају све комплетне вредности листе $key1list[i]$, $i = 4, \ldots, 12$, излази се из рекурзије и позива се функција $formiraj\_listu\_key0()$.

Претходно је показано (стр. 26) да, ако је позната једна конкретна вредност $key1list[i]$, у просеку се добија само једна одговарајућа вредност $key1list[i{-}1]$. Овде је битно уочити да реч **у просеку** не значи увек и само једну. Ово је узрок грешке у програму (која се јако тешко открива) због које програм у једном малом броју случајева не проналази интерни кључ (иако се чини да испитује све могуће случајеве). При испитивању коју конкретну вредност из интервала $[key1\_i\_minus\_1 - (key1\_i\_minus\_1{-}255)]$ узима $key2list[i{-}1]$, у петљи се не сме ставити команда $break$ (као показатељ да, ако је пронађена права вредност, треба изаћи из петље). Тестирањем је показано да $key2list[i{-}1]$ може узети не само једну, него две различите вредности из задатог интервала, тако да све једнакости и везе између променљивих и даље буду задовољене. Показано је (такође тестирањем) да $key2list[i{-}1]$ не може узети више од две вредности из датог интервала.

Претходна анализа показује је да се за једну конкретну листу вредности $key2list[i]$, $i = 2, \ldots, 12$ добија у просеку 2^{16} могућих листи вредности $key1list[j]$, $j = 4, \ldots, 12$, али само у просеку. Може их бити и више, али не много више (у примерима се добија за 10-15% више).

4.2.3 Одређивање листе key0 и интерног кључа

Наредни корак је позив функције $formiraj\_listu\_key0()$ која на основу познатих вредности $LSB(key0list[j])$, $j = 5, \ldots, 12$, формира листу комплетних вредности $key0list[i]$, $i = 4, \ldots, 12$, затим проверава да ли је то листа која се заиста формира при шифровању архиве и ако је то тачно, комплетира листу свих вредности $key0list[i]$, $key1list[i]$ и $key2list[i], i = 0, \ldots, 12$, проналази интерни кључ, штампа одговарајућу поруку и зауставља даљу претрагу.

На основу 3.5 добија се
$key0list[i] \gg 8 = key0list[i{+}1] \oplus crc32tab[LSB(key0list[i]) \oplus nesifr\_podaci[i]]$.
За вредности $key0list[12]$ и $key0list[11]$ познати су битови на позицијама 0-7, а за вредност $crc32tab[LSB(key0list[11]) \oplus nesifr\_podaci[11]]$ познати су сви битови (0-31), па се добијају битови на позицијама 0-7 за $key0list[11] \gg 8$, односно битови на позицијама 8-15 вредности $key0list[11]$. Како су већ познати битови на позицијема 0-7, закључује се да су познати битови на позицијама 0-15 вредности $key0list[11]$.

Ако се горњи израз примени на $key0list[11]$ и $key0list[10]$, добијају се познати битови на позицијама 0-15 израза $key0list[10] \gg 8$, односно битови на

позицијама 8-23 вредности $key0list[10]$. Како су битови на позицијама 0-7 већ познати, укупно су познати битови на позицијама 0-23 вредности $key0list[10]$. Аналогним поступком се добија комплетна вредност $key0list[9]$ (битови на позицијама 0-31).

Ако је позната комплетна вредност $key0list[9]$, на основу израза $key0list[i{-}1] = (key0list[i] \gg 8) \oplus crc32tab[LSB(key0list[i]) \oplus nesifr\_podaci[i]]$ добијају се вредности $key0list[10]$, $key0list[11]$ и $key0list[12]$, а на основу израза $key0list[i-1] = (key0list[i] \ll 8) \oplus invcrc32tab[MSB(key0list[i])] \oplus nesifr\_podaci[i{-}1]$ добијају се вредности $key0list[i]$, $i = 4, \ldots, 8$, чији се бајтови најмање тежине упоређују са бајтовима најмање тежине већ познатих вредности ($LSB(key0list[i])$, $i = 4, \ldots, 8$). Ако се поклапају свих пет бајтова, пронађена је листа вредности која се користи у процесу шифровању фајла, а у супротном се прелази на испитивање следеће могуће листе вредности.

Ако је пронађена листа вредности $key0list[i]$, $i = 4, \ldots, 12$, која се користи у шифровању фајла, прелази се на одређивање интерног кључа. На основу 3.6 налазе се вредности $key0list[i]$, $key1list[i]$, $key2list[i]$, $i = 0, \ldots, 3$. Пошто је на почетак шифрованог фајла додато шифарско заглавље, у петљи (12 итерација) се чита један по један бајт (из променљиве $random\_bajtovi$), дешифрује се и примењују се формуле 3.6. Тиме је добијен интерни кључ. Та вредност се исписује и прекида се даља претрага.

Ако је пронађен интерни кључ, за прекид претраге користи се глобална променљива $nadjena\_kompletna\_lista$. Њена вредност је на почетку функције $main$ постављена на нулу, а кад се пронађе интерни кључ, њена вредност се поставља на јединицу. При испитивању могућих комбинација у функцијама zip_crack, $key2\_rekurzija$, $formiraj\_listu\_key1$ и $key1\_rekurzija$ проверава се да ли је пронађен интерни кључ, тј. да ли је $nadjena\_kompletna\_lista == 1$. Ако је то тачно, прекида се свако даље испитивање могућих комбинација.

4.2.4 Одређивање лозинке

Када је пронађен интерни кључ, корисник одређује да ли да програм настави са даљим извршавањем (притиком на тастер y) и да пронађе лозинку којом је шифрован фајл или да прекине даље извршавање програма (притиском на било који други тастер).

Ако корисник жели да пронађе лозинку којом је шифрован фајл, позива се функција $pronadji\_lozinku()$ која на основу познате вредности интерног кључа проналази лозинку. Лозинка се чува у глобалној променљиви $lozinka[\,]$. Претпоставља се да лозинка нема више од 13 знакова. Променљива $duzina\_lozinke$ чува дужину лозинке. У функцији се проверавају лозинке дужине $0, 1, 2, \ldots, 13$.

Ако је интерни кључ једнак почетним вредностима: ($key0list[0]$, $key1list[0]$, $key2list[0]$)=($0x12345678$, $0x23456789$, $0x34567890$), пронађена је лозинка дужине нула, тј. није унета никаква лозинка.

Полазећи од 3.5добија се
$key0list[i] = (key0list[i{+}1] \ll 8) \oplus invcrc32tab[MSB(key0list[i{+}1])] \oplus char$. На основу познате вредности $key0list[0]$, тако се добија вредност $key0\_i\_minus1$ ($= key0_{-1}$). Овде се може ставити $char = 0x00$, јер $char$ не утиче на горња три бајта

$key0\_i\_minus1$. Ако се горња три бајта $key0\_i\_minus1$ поклапају са горња три бајта вредности $0x12345678$, добијена је лозинка чија је дужина један. Лозинка је одређена изразом $lozinka[0] = (key0\_i\_minus1 \oplus 0x12345678)\ \&\ 0xff$. Ако се горња формула примени још једном, добија се $key0\_i\_minus2$ $(= key0_{-2})$. Ако се горња два бајта вредности $key0\_i\_minus2$ подударају са горња два бајта вредности $0x12345678$, добијена је лозинка дужине два. Лозинка је одређена изразом $lozinka[i] = ((key0\_i\_minus2 \oplus 0x12345678) \gg (i*8))\ \&\ 0xff, i = 0, 1$. Аналогним поступком добијају се лозинке дужина три и четири. У сваком од ових случајева проверава се да ли је добијена права лозинка. Врши се први корак шифровања (стр. 23) и проверава се да ли се коришћењем добијене лозинке заиста добија интерни кључ. У том случају програм исписује добијену лозинку и завршава са радом, а у супротном прелази на испитивање лозинки дужине пет.

При испитивању лозинки дужина већих од четири користе се низови $key1\_sifra[\,]$ и $key0\_sifra[\,]$, глобалне променљиве. Они садрже потенцијалне листе вредности $key1_i$ и $key0_i$ које се формирају у првом кораку шифровања, при комбиновању иницијалних вредности $key0$, $key1$ и $key2$ са добијеном лозинком.

Најпре се испитује се да ли је лозинка дужине пет. Применом израза 3.1 добија се $key2\_i\_minus1$ $(= key2_{-1})$. Поставља се последњи елемент низа $key1\_sifra[6]$ на $key1list[0]$. Четири пута се примењује 3.1 на $key2\_i\_minus1$ и добија се вредност која представља $key2_{-5}$. Упоређује се добијена вредност са $0x34567890$ и добијају се потенцијалне вредности бајтова највеће тежине $key1_{-4}, \ldots, key1_{-1}$ (примењује се аналоган поступак, као при добијању $key0_{-i}$ у тражењу лозинки дужине четири и мање). Добијени бајтови се постављају у низ $key1\_sifra$ на позиције 2–5. Позива се функција $key1\_sifra5\_rekurzija(i)$ која је веома слична функцији $key1\_rekurzija$. Она на основу познате комплетне вредности $key1\_sifra[i]$ и познатих $MSB(key1\_sifra[i–1])$ и $MSB(key1\_sifra[i–2])$, израчунава комплетну вредност $key1\_sifra[i–1]$. Примењује се исти поступак као у функцији $key1\_rekurzija$ (стр. 42). Овим поступком налазе се све могуће комплетне вредности $key1\_sifra[3]$, $key1\_sifra[4]$ и $key1\_sifra[5]$ (вредност $key1\_sifra[6]$ је већ позната). Добијају се и $LSB(key0\_sifra[4])$ и $LSB(key0\_sifra[5])$. Када су све ове вредности пронађене, излази се из рекурзије. У петљи се одређује последњи знак лозинке. Узимају се све могуће вредности (има их 256) и проверава се који знак (коришћењем формуле 3.5) повезује $LSB(key0\_sifra[5])$ и $key0list[0]$. Преостала четири знака лозинке одређују се на исти начин као код тражења лозинки дужине четири.

Ако није добијена лозинка дужине пет, прелази се на испитивање лозинки дужине шест. Коришћењем првог и другог израза из 3.6 добијају се вредности $key2\_i\_minus1$ $(= key2_{-1})$, $key1\_i\_minus1$ $(= key1_{-1})$ и $key2\_i\_minus2$ $(= key2_{-2})$. Поставља се последњи елемент низа $key1\_sifra[6]$ на $key1\_i\_minus1$, а затим се примењује исти поступак као код одређивања лозинке дужине пет, добија вредност која представља $key2_{-6}$ и потенцијалне вредности бајтова највеће тежине $key1_{-5}, \ldots, key1_{-2}$. Добијени бајтови постављају се у низ $key1\_sifra$ на позиције $2-5$. Позива се функција $key1\_sifra6\_rekurzija(i)$ која (исто као у функцији $key1\_sifra5\_rekurzija$) налази све могуће комплетне вредности $key1\_sifra[3]$, $key1\_sifra[4]$ и $key1\_sifra[5]$, као и $LSB(key0\_sifra[4])$

и $LSB(key0\_sifra[5])$. Последња два знака лозинке добијају се коришћењем израза (изведеног из израза $key0_{i+1} = crc32(key0_i, char)$, који је саставни део функције *update_keys*): $key0[i-1] = (key0[i] \ll 8) \oplus invcrc32tab[MSB(key0[i])] \oplus lozinka[i-1]$, тј. $lozinka[i-1] = (key0[i] \ll 8) \oplus invcrc32tab[MSB(key0[i])] \oplus key0[i-1]$. Како $lozinka[i-1]$ представља један знак (8 битова), $(key0[i] \ll 8)$ не утиче на резултат, па се може изоставити. Зато израз добија облик:

$$lozinka[i-1] = (invcrc32tab[MSB(key0[i])] \oplus key0[i-1]) \;\&\; 0xff.$$

Приликом прве употребе овог израза $key0[i]$ је комплетно познат ($= key0list[0]$), а за $key0[i-1]$ је познат бајт најмање тежине ($= LSB(key0\_sifra[6])$), па се може одредити $lozinka[5]$. На основу те вредности (коришћењем последњег израза из 3.6) се одређује $key0\_i\_minus1$. $lozinka[4]$ се одређује коришћењем истог израза као и $lozinka[5]$. Преостала четири знака лозинке одређују се на исти начин као код испитивања лозинки дужине четири. Када је пронађена потенцијална лозинка, врши се тестирање да би се проверило да ли је то права лозинка. Врши се први корак шифровања и проверава се да ли се коришћењем добијене лозинке заиста добија интерни кључ. У том случају програм исписује добијену лозинку и завршава са радом, а у супротном прелази на испитивање следеће могуће лозинке дужине шест. Ако су испитане све могуће лозинке дужине шест, а није нађена права, прелази се на испитивање лозинки дужине седам.

За лозинке чија је дужина већа од шест, првих $duzina\_lozinke-6$ знакова постављају се на све могуће начине (за сваки од тих знакова постоји 256 могућности), а последњих шест знакова налазе се на исти начин као код испитивања лозинки дужине шест. Код лозинки дужине седам до десет, за постављање првих произвољних знакова довољно је користи четворобајтну променљиву ($unsigned\ int$) $pom1$. Она узима једну од 2^{32} различитих вредности, а то је довољно да покрије све могуће комбинације четири знака. За лозинке дужине веће од десет, користи се још једна ($unsigned\ int$) променљива $pom2$. Истовременим коришћењем променљивих $pom1$ и $pom2$ добијају се све могуће комбинације за осам знакова. Када се фиксира првих $duzina\_lozinke-6$ знакова, позива се функција $pronadji\_dugacku\_lozinku(duzina\_lozinke)$ која одређује преосталих шест знакова и проверава да ли је добијена одговарајућа лозинка. Ова функција врши први корак шифровања за првих $duzina\_lozinke-6$ знакова и добијене вредности $key0_i, key1_i, key2_i$, $i = duzina\_lozinke-6$, поставља за иницијалне, уместо ($0x12345678$, $0x23456879$, $0x34567890$). Затим се користи аналоган поступак као при одређивању лозинке дужине шест.

Ако је пронађена одговарајућа лозинка, за прекид даље претраге користи се глобална променљива $pronadjena\_lozinka$. Њена вредност је на почетку функције $pronadji\_lozinku$ постављена на нулу, а кад се пронађе лозинка, њена вредност се поставља на јединицу. При испитивању свих могућих лозинки у функцијама $pronadji\_lozinku$, $key1\_sifra5\_rekurzija$, $key1\_sifra6\_rekurzija$ и $pronadji\_dugacku\_lozinku$ проверава се да ли је пронађена лозинка, тј. да ли је $pronadjena\_lozinka == 1$. Ако је то тачно, прекида се свако даље испитивање могућих комбинација, исписује се лозинка и програм завршава са радом.

4.2.5 Анализа ефикасности

Време које је потребно за одређивање интерног кључа зависи од броја познатих бајтова отвореног текста, а време потребно за одређивање лозинке зависи од дужине лозинке. Ова два времена су независна, па се укупно време рада програма добија њиховим сабирањем.

У следећој табели приказана су времена потребна за одређивање интерног кључа (резултати су добијени тестирањем програма). Приказана су времена потребна за испитивање свих могућих потенцијалних листи ($key0_i, key1_i, key2_i$). Ако је пронађен интерни кључ, програм прекида даљу претрагу, па време извршавања програма може бити и мање. Број могућих вредности $key2_{12}$ чува се у $broj\_key2n$. Од једне конкретне вредности $key2_{12}$ у просеку се добија једна комплетна листа $key2_i, i = 0, \ldots, 12$, али само у просеку. Број комплетних листи $key2_i$ чува се у $broj\_listi\_key2n$. Програм у просеку испитује 32-35 потенцијалних листи $key2_i$ у секунди, па време испитивања свих могућих кандидата зависи од броја кандидата за листу $key2_i$.

Број познатих бајтова отвореног текста	$broj\_key2n$	$broj\_listi\_key2n$	Време потребно за испитивање свих могућих кандидата
13	4194304	4794465	101 дан
14	2626561	2762671	57 дана
15	1928744	2027882	42 дана
16	1496894	1570764	33 дана
17	1160283	1216259	25 дана
20	769309	807267	17 дана
23	589408	619336	13 дана
33	316770	333764	7 дана
43	222706	234280	5 дана
63	134863	134965	2 дана и 19 сати
113	68926	73017	1 дан и 12 сати
183	42617	45267	23 сата
343	21681	23355	12 сати
503	15894	17518	9 сати
823	10455	11566	6 сати
1013	9367	10491	5 сати и 30 минута
2013	3690	4044	2 сата

Резултати из горње табеле добијени су тестирањем програма са различитим бројем познатих бајтова отвореног текста једног истог фајла (због упоређивања). Програм је комплетно тестиран са $33, 43, \ldots, 2013$ познатих бајтова отвореног текста, док је за мањи број познатих бајтова отвореног текста вршено само тестирање вредности променљиве $broj\_listi\_key2n$, а на основу ње је процењено време потребно за испитивање свих могућих кандидата. Времена приказана у табели представљају времена потребна за испитивање свих могућих кандидата, док се у просеку резултат добија за двоструко краће време. Ако се програм тестира на некој другој архиви, добијају се друге вредности. Те вредности не

одступају пуно (± 10–15%) од вредности приказаних у табели.

Теоретски, интерни кључ се може одредити и са мање од 13 познатих бајтова отвореног текста. Ако је познато 12 бајтова отвореног текста, онда се тринаести бајт може претпоставити на 256 начина. У том случају, време потребно за испитивање свих могућих случајева се повећава 256 пута, тј. потребно је око 70 година за испитивање свих могућности. Због тога је потребно знати бар 13 бајтова отвореног текста, да би програм завршио испитивање свих могућих кандидата у неком разумном времену.

Време потребно за одређивање лозинке зависи од њене дужине. За лозинке чија је дужине мања од седам, време потребно за одређивање лозинке је занемарљиво у односу на време потребно за одређивање интерног кључа. За лозинке дужине седам и веће, узимају се све могуће вредности за првих $duzina\_lozinke$–6 знакова. Сваки знак има 256 могућих вредности. Ако се дужина лозинке повећа за један, време потребно за одређивање лозинке се повећава 256 пута.

Дужина лозинке	Потребно време	Дужина лозинке	Потребно време
7	$7,8 * 10^{-5}\ sec.$	11	3,5 дана
8	$0,02\ sec.$	12	2,5 године
9	$5\ sec.$	13	628 година
10	$20\ min.$	14	160768 година

Вредности из претходне табеле добијене су тестирањем програма за лозинке дужине до 11. За лозинке већих дужина, време потребно за одређивање лозинке је процењено. Резултати из горње табеле представљају времена потребна за испитивање свих могућих комбинација знакова лозинке (у најгорем могућем случају). У тренутку кад се одреди лозинка, прекида се даља претрага и програм завршава са радом. У том случају је потребно мање времена за добијање лозинке.

Поглавље 5

Закључци и правци даљег рада

У овом раду представљена је реализација напада описаног у [1] на шифроване .ZIP архиве. Рад потврђује закључак да је шифровање коришћењем интерног кључа од 96 бита веома слабо и да га не треба користити за заштиту важних података.

Развијени програм може се даље усавршавати у правцу реализације специјализованих напада на шифрате појединих типова фајлова, уз избегавање потребе за познавањем дела отвореног текста.

Пронађени су начини за дешифровање фајлова који су шифровани старим алгоритмима, али су и осмишљени нови, бољи алгоритми за шифровање. Програмери се, са једне стране, труде да пронађу слабости алгоритама који се данас користе за шифровање, а са друге стране, да осмисле још боље алгоритме за шифровање, које ће бити још теже разбити. Као и увек у животу, некад надјача једна, некад друга струја, некад се чини да је неки алгоритам савршен, да нема слабости, а некад изгледа да не постоји ниједан добар и поуздан алгоритам. Оваква конкуренција увек ће постојати.

Литература

[1] E. Biham, *How to decrypt or even substitute DES-encrypted messages in* 2^{38} *steps,* Information Processing Letters, 84, 2002.

[2] E. Biham, P. Kocher, *A known plaintext attack on the PKZIP stream cipher,* Fast Software Encryption '94, Lecture Notes in Computer Science, vol. 1008, Springer-Verlag, Berlin Germany, 1994.

[3] F. Buccholz, *The structure of a PKZIP file,* доступно на https://users.cs.jmu.edu/buchhofp/forensics/formats/pkzip.html, 2002.

[4] T. Kohno, *Analysis of the WinZip encryption method* , University of California, San Diego USA, 2004.

[5] PKWARE Inc, *APPNOTE.txt – .ZIP File Format Specification* , version 6.3.2, 2007.

[6] M. Stay, *ZIP attacks with reduced known plaintext* , Fast Software Encryption 2001, Lecture Notes in Computer Science, vol. 2355, 124-134, Springer, Berlin, 2001.

[7] M. Stevens, E. Flanders, *Yet another Plaintext Attack on ZIP's Encryption Scheme,* доступно на http://www.securiteam.com/securitynews/5LP0A0096O.html, 2003.

[8] J. S. Vitter, *Design and analysis of dynamic Huffman codes,* Journal of the Association for Computing Machinery, 1987.

[9] http://www.pkware.com

[10] http://en.wikipedia.org