How we recovered $XXX,000 of Bitcoin from an encrypted zip file

Michael Stay, PhD
CTO, Pyrofex Corp.
DEF CON 2020

Products

PRODUCTS

- Password Recovery Toolkits
- **Password Recovery Modules**
- NT & Novell Access Utilities
- Distributed Network Attack
- Forensic Toolkit
- SecureClean
- SecureClean Administrator
- CleanDrive

PASSWORD RECOVERY TOOLKIT MODULES

Lost Your Password? Get It Back.

Two ways to recover your lost password.

1. Personal Software Package

AccessData has a wide variety of individual password breaking modules that can help you recover lost passwords for almost every product in the industry. If you need to recover lost passwords for several different programs, please see our password recover tool kits.

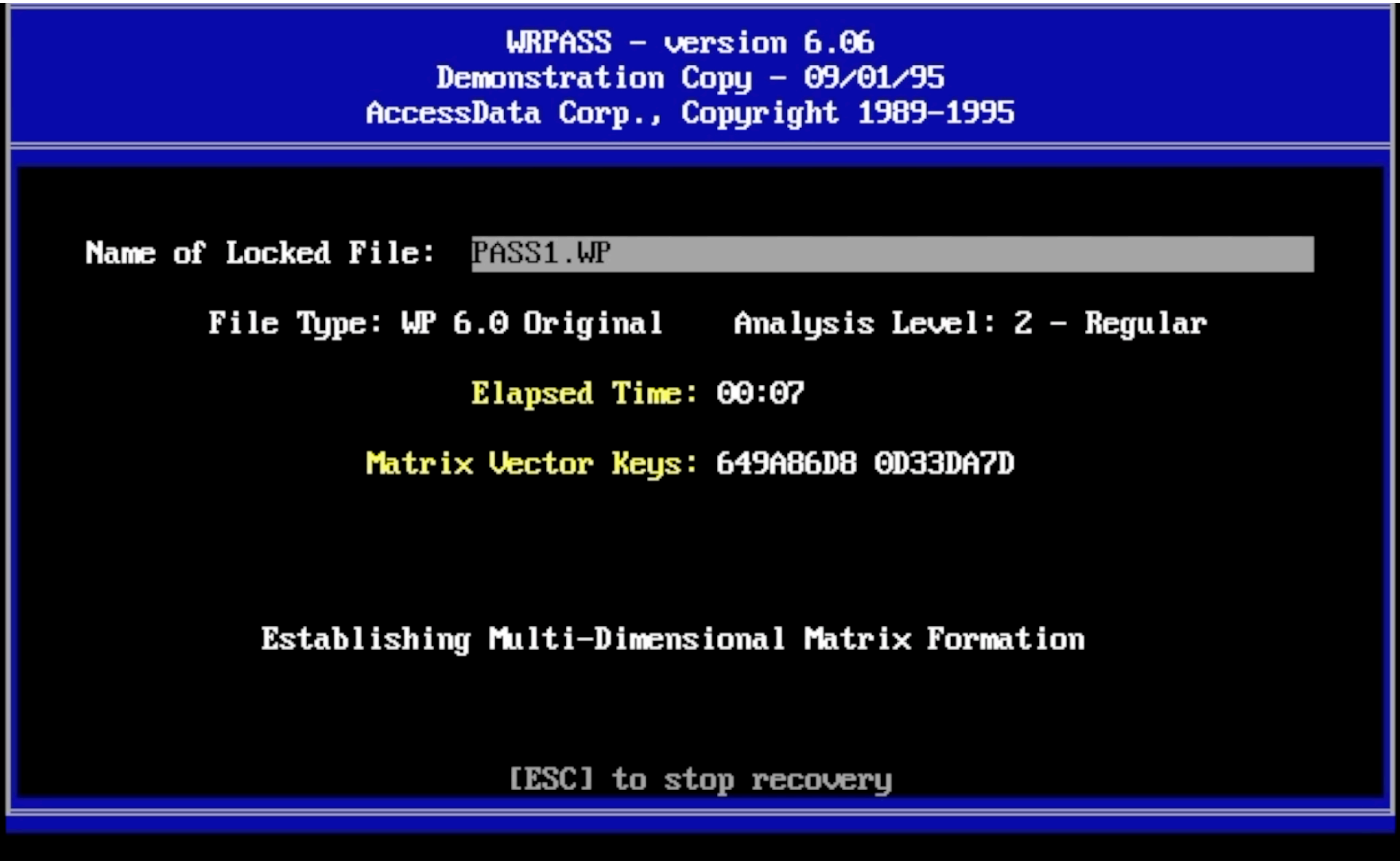
Individual Password Breaker Modules:

Access
ACT!
Ami Pro
Approach
Ascend
BestCrypt
DataPerfect
Excel
FoxBase
Lotus 1-2-3
MS Money
Organizer
Outlook
Paradox
Pro Write
WinZip & Generic Zippers
Q&A;
Quattro Pro
QuickBooks
Quicken
Scheduler+
Symphony
VersaCheck
Word
WordPerfect
Word Pro

Utilities That Bypass Network Administrator Passwords:
Windows NT & Novell

Free Lost Password Breakers:
If you know how to use a hex editor, here's how to break Quicken 3.0 and Money 2.0.


Microsoft Word
File Edit View Insert Format Tools Table Window Help
Microsoft
This copy of Microsoft Word is licensed to:
User
Company
Microsoft
Word
for Windows 95
Version 7.0
Copyright© 1983-1995
Microsoft Corporation
This program is protected by U.S and International copyright laws as described in Help About.
Start
Microsoft Word
5:40 AM


WRPASS - version 6.06
Demonstration Copy - 09/01/95
AccessData Corp., Copyright 1989-1995
Name of Locked File: PASS1.WP
File Type: WP 6.0 Original
Analysis Level: 2 - Regular
Elapsed Time: 00:07
Matrix Vector Keys: 649A86D8 0D33DA7D
Establishing Multi-Dimensional Matrix Formation
[ESC] to stop recovery


Microsoft Word - Document1
File Edit View Ir
Normal
100%
About Microsoft Word
Microsoft® Word 97
Copyright © 1983-1996 Microsoft Corporation
International CorrectSpell™© 1993 and International Hyphenator© 1991 by INSO Corporation. All rights reserved.
Thesaurus© 1984-1996 Soft-Art, Inc.® All rights reserved.
Certain templates developed for Microsoft Corporation by Impressa Systems, Santa Rosa, California.
Compare Versions ©1993 Advanced Software, Inc. All rights reserved.
This product is licensed to:
User
Company
Product ID: 22222-OEM-2222222-22222
Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.
OK
System Info...
Tech Support...
Page Sec 1
At 1" Ln 1 Col 1
REC TRK EXT OVR WPH
Start
My Computer
Microsoft Word - Doc...
7:52 AM

File: APPNOTE.TXT - .ZIP File Format Specification
Version: 6.3.9
Status: FINAL - replaces version 6.3.8
Revised: July 15, 2020

1.0 Introduction

1.1 Purpose

1.1.1 This specification is intended to define a cross-platform, interoperable file storage and transfer format. Since its first publication in 1989, PKWARE, Inc. ("PKWARE") has remained committed to ensuring the interoperability of the .ZIP file format through periodic publication and maintenance of this specification. We trust that all .ZIP compatible vendors and application developers that use and benefit from this format will share and support this commitment to interoperability.

1.2 Scope

1.2.1 ZIP is one of the most widely used compressed file formats. It is universally used to aggregate, compress, and encrypt files into a single interoperable container. No specific use or application need is defined by this format and no specific implementation guidance is provided. This document provides details on the storage format for creating ZIP files. Information is provided on the records and fields that describe what a ZIP file is.

1.3 Trademarks

1.3.1 PKWARE, PKZIP, Smartcrypt, SecureZIP, and PKSFX are registered trademarks of PKWARE, Inc. in the United States and elsewhere. PKPatchMaker, Deflate64, and ZIP64 are trademarks of PKWARE, Inc. Other marks referenced within this document appear for identification purposes only and are the property of their respective owners.

1.4 Permitted Use

Info-ZIP

LATEST RELEASES: ***Zip 3.00*** was released on 7 July 2008. ***WiZ 5.03*** was released on 11 March 2005. ***UnZip 6.0*** was released on 29 April 2009. ***MacZip 1.06*** was released on 22 February 2001. See the Zip, UnZip and WiZ pages for current status and download locations.

In addition, a new set of **discussion forums** was set up in October 2007. These replace the older QuickTopic forum, which was overrun by spam. (The spam postings have since been deleted, but further posts to the old forum are permanently disabled.)

About Info-ZIP

Info-ZIP is a diverse, Internet-based workgroup of about 20 primary authors and over one hundred beta-testers, formed in 1990 as a mailing list hosted by Keith Petersen on the original SimTel site at the White Sands Missile Range in New Mexico.

Info-ZIP's purpose is to provide free, portable, high-quality versions of the **Zip** and **UnZip** compressor-archiver utilities that are compatible with the DOS-based **PKZIP** by PKWARE, Inc.

Info-ZIP supports hardware from microcomputers all the way up to Cray supercomputers, running on almost all versions of Unix, VMS, OS/2, Windows 9x/NT/etc. (a.k.a. Win32), Windows 3.x, Windows CE, MS-DOS, AmigaDOS, Atari TOS, Acorn RISC OS, BeOS, Mac OS, SMS/QDOS, MVS and OS/390 OE, VM/CMS, FlexOS, Tandem NSK and Human68K (Japanese). There is also some (old) support for LynxOS, TOPS-20, AOS/VS and Novell NLMs. Shared libraries (DLLs) are available for Unix, OS/2, Win32 and Win16, and graphical interfaces are available for Win32, Win16, WinCE and Mac OS.

Info-ZIP code has been incorporated into a number of third-party products as well, both commercial and freeware. Some of the more interesting ones (well, historically speaking) include the use of **UnZip** code in the *unzip.dll* distributed with IBM's **OS/2 Warp BonusPak** and **WebExplorer**, as part of the reinstallation code for the IBM Aptivas preloaded with **OS/2 Warp**, and as part of IBM's Infoprint product. Sun used Info-ZIP's self-extractor to distribute the NT version of their **HotJava** browser, Novell uses UnZip for **NetWare 6** installation, and SAP includes it in **Business One**. Various Windows products such as **WinZip** and the **DynaZIP** DLLs incorporate Info-ZIP code, too. And let us not forget **Pretty Good Privacy** (PGP), an excellent encryption program that uses Info-ZIP code as a first step in encrypting files. Info-ZIP's primary compression engine has also been spun off into the free **zlib** compression library, used in **Netscape/Mozilla/Firefox**, the **Linux** kernel, **Windows**, **Java**, virtually all **PNG**-supporting software, and countless other products.

Info-ZIP can be reached by a web-based form, but you'll have to read our **Frequently Asked Questions** page to find out how. Our two primary web sites are hosted by our very own Hunter Goatley and by the most excellent SourceForge. Secondary distribution sites are hosted by the Comprehensive TeX Archive Network.

A Known Plaintext Attack on the PKZIP Stream Cipher

Eli Biham* Paul C. Kocher**

Abstract. The PKZIP program is one of the more widely used archive/compression programs on personal computers. It also has many compatible variants on other computers, and is used by most BBS's and ftp sites to compress their archives. PKZIP provides a stream cipher which allows users to scramble files with variable length keys (passwords).
In this paper we describe a known plaintext attack on this cipher, which can find the internal representation of the key within a few hours on a personal computer using a few hundred bytes of known plaintext. In many cases, the actual user keys can also be found from the internal representation. We conclude that the PKZIP cipher is weak, and should not be used to protect valuable data.

1 Introduction

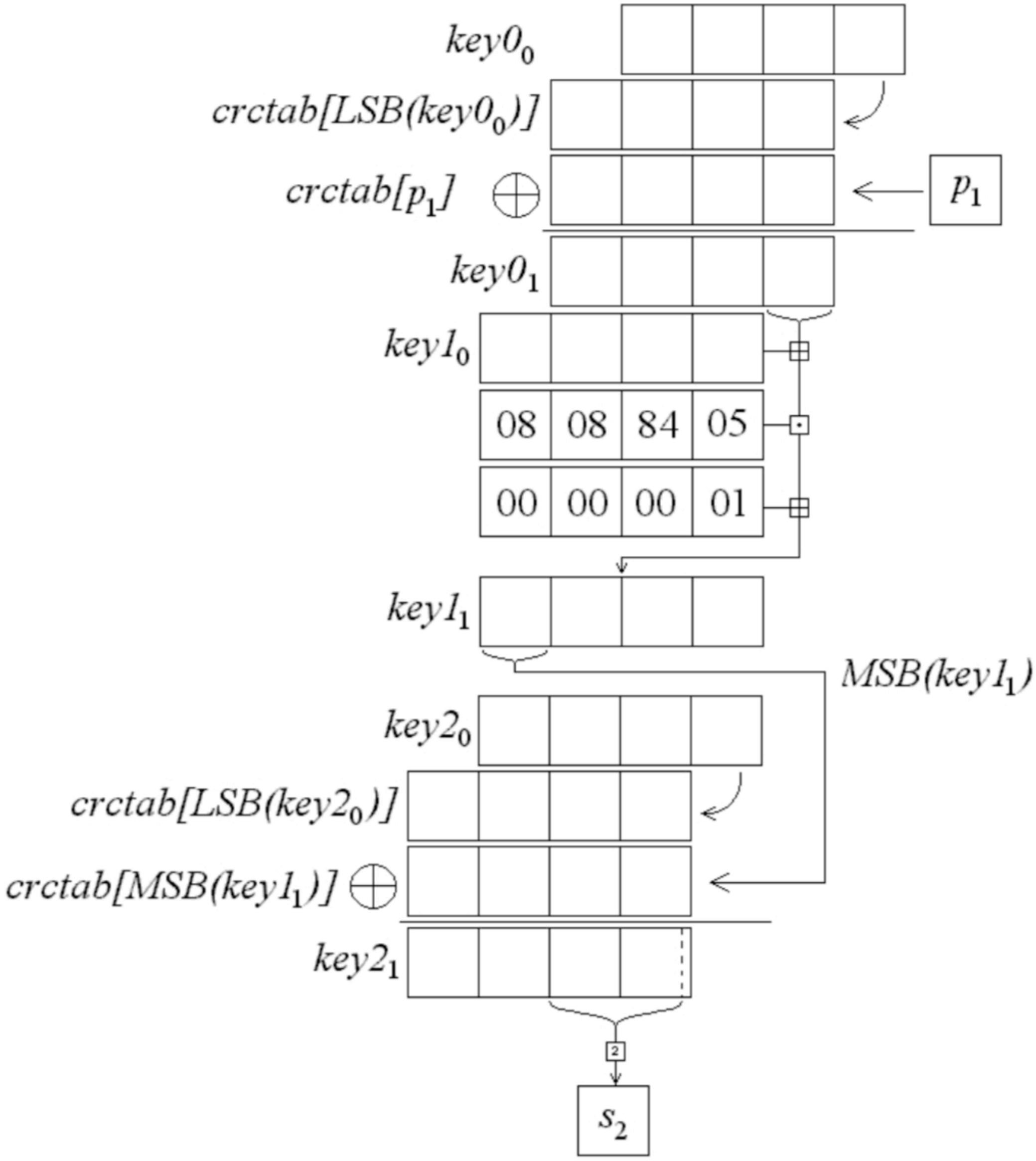
The PKZIP program is one of the more widely used archive/compression programs on personal computers. It also has many compatible variants on other computers (such as Infozip's zip/unzip), and is used by most BBS's and ftp sites to compress their archives. PKZIP provides a stream cipher which allows users to scramble the archived files under variable length keys (passwords). This stream cipher was designed by Roger Schlafly.

In this paper we describe a known plaintext attack on the PKZIP stream cipher which takes a few hours on a personal computer and requires about 13–40 (compressed) known plaintext bytes, or the first 30–200 uncompressed bytes, when the file is compressed. The attack primarily finds the 96-bit internal representation of the key, which suffices to decrypt the whole file and any other file encrypted under the same key. Later, the original key can be constructed. This attack was used to find the key of the PKZIP contest.

The analysis in this paper holds to both versions of PKZIP: version 1.10 and version 2.04g. The ciphers used in the two versions differ in minor details, which does not affect the analysis.

The structure of this paper is as follows: Section 2 describes PKZIP and the PKZIP stream cipher. The attack is described in Section 3, and a summary of the results is given in Section 4.

* Computer Science Department, Technion - Israel Institute of Technology, Haifa 32000, Israel
** Independent cryptographic consultant, 7700 N.W. Ridgewood Dr., Corvallis, OR 97330, USA


$key0_0$
$crctab[LSB(key0_0)]$
$crctab[p_1]$
p_1
$key0_1$
$key1_0$
08 08 84 05
00 00 00 01
$key1_1$
$MSB(key1_1)$
$key2_0$
$crctab[LSB(key2_0)]$
$crctab[MSB(key1_1)]$
$key2_1$
2
s_2

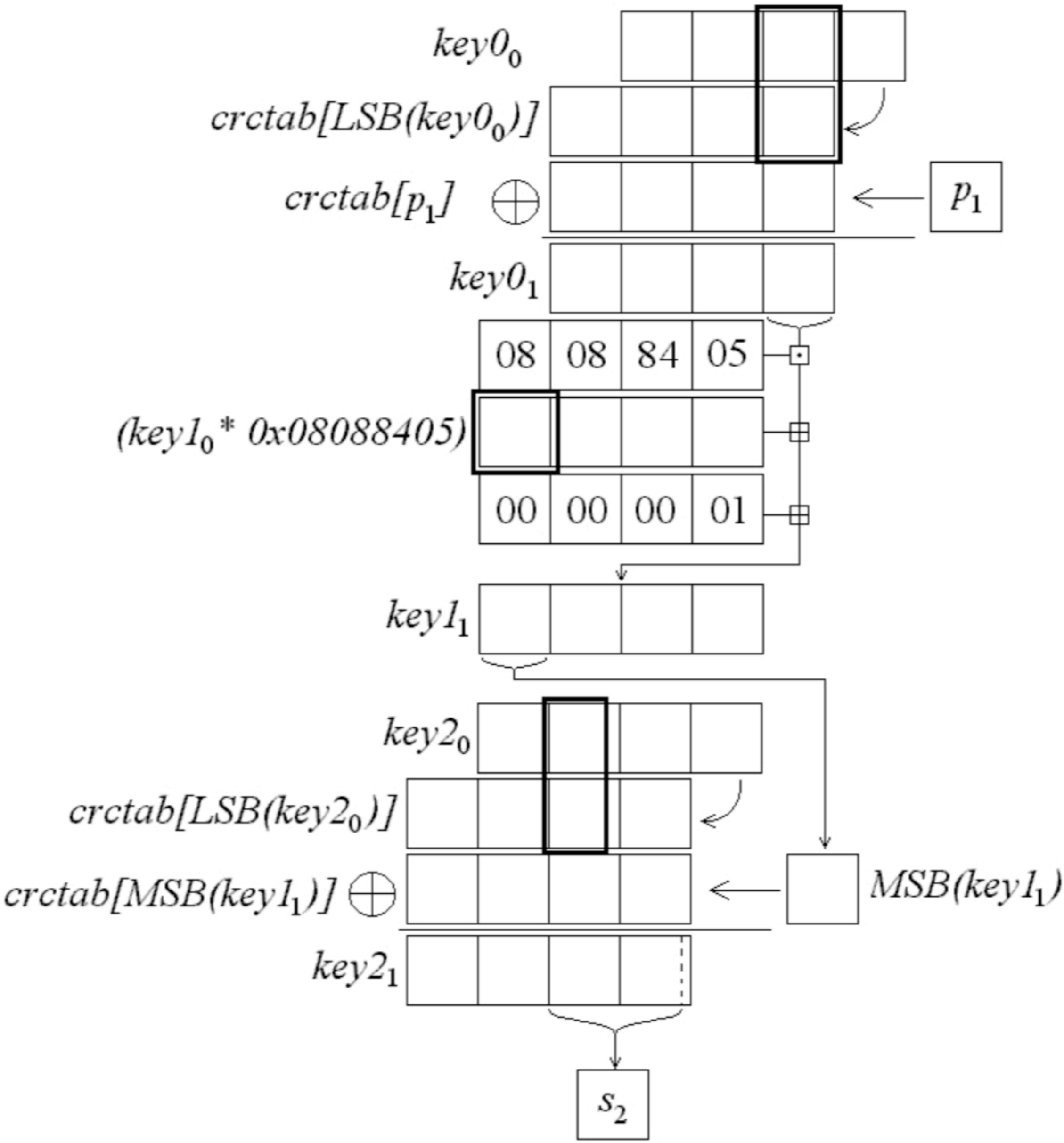
```
122  /***********************************************************************
123   * Return the next byte in the pseudo-random sequence
124   */
125  int decrypt_byte(__G)
126      __GDEF
127  {
128      unsigned temp;  /* POTENTIAL BUG:  temp*(temp^1) may overflow in an
129                       * unpredictable manner on 16-bit systems; not a problem
130                       * with any known compiler so far, though */
131
132      temp = ((unsigned)GLOBAL(keys[2]) & 0xffff) | 2;
133      return (int)(((temp * (temp ^ 1)) >> 8) & 0xff);
134  }
135
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	…
	rand()										CRC32		plaintext				

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	…
x =	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	c0	c1	p0	p1	p2	p3	…

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	…
x =	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	c0	c1	p0	p1	p2	p3	…
password	s0	s1x	s2x	s3x	s4x	s5x	s6x	s7x	s8x	s9x							
y =	r0 ^ s0	r1 ^ s1x	r2 ^ s2x	r3 ^ s3x	r4 ^ s4x	r5 ^ s5x	r6 ^ s6x	r7 ^ s7x	r8 ^ s8x	r9 ^ s9x							

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	…
x =	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	c0	c1	p0	p1	p2	p3	…
password	s0	s1x	s2x	s3x	s4x	s5x	s6x	s7x	s8x	s9x							
y =	r0 ^ s0	r1 ^ s1x	r2 ^ s2x	r3 ^ s3x	r4 ^ s4x	r5 ^ s5x	r6 ^ s6x	r7 ^ s7x	r8 ^ s8x	r9 ^ s9x							
password	s0	s1y	s2y	s3y	s4y	s5y	s6y	s7y	s8y	s9y	sAy	sBy	sCy	sDy	sEy	sFy	…
h =	r0	r1 ^ s1x ^ s1y	r2 ^ s2x ^ s2y	r3 ^ s3x ^ s3y	r4 ^ s4x ^ s4y	r5 ^ s5x ^ s5y	r6 ^ s6x ^ s6y	r7 ^ s7x ^ s7y	r8 ^ s8x ^ s8y	r9 ^ s9x ^ s9y	c0 ^ sAy	c1 ^ sBy	p0 ^ sCy	p1 ^ sDy	p2 ^ sEy	p3 ^ sFy	…


$key0_0$
$crctab[LSB(key0_0)]$
$crctab[p_1]$
p_1
$key0_1$
08 08 84 05
$(key1_0 * 0x08088405)$
00 00 00 01
$key1_1$
$key2_0$
$crctab[LSB(key2_0)]$
$crctab[MSB(key1_1)]$
$MSB(key1_1)$
$key2_1$
s_2

ZIP Attacks with Reduced Known Plaintext

Michael Stay

AccessData Corporation
2500 N. University Ave. Ste. 200
Provo, UT 84606
staym@accessdata.com

Abstract. Biham and Kocher demonstrated that the PKZIP stream cipher was weak and presented an attack requiring thirteen bytes of plaintext. The *deflate* algorithm "zippers" now use to compress the plaintext before encryption makes it difficult to get known plaintext. We consider the problem of reducing the amount of known plaintext by finding other ways to filter key guesses. In most cases we can reduce the amount of known plaintext from the archived file to two or three bytes, depending on the zipper used and the number of files in the archive. For the most popular zippers on the Internet, there is a fast attack that does not require any information about the files in the archive; instead, it gets doubly-encrypted plaintext by exploiting a weakness in the pseudorandom-number generator.

1 Introduction

PKZIP is a compression / archival program created by Phil Katz. Katz had the foresight to document his file format completely in the file APPNOTE.TXT, distributed with every copy of PKZIP; there are now literally hundreds of "zipper" programs available, and the ZIP file format has become a *de facto* standard on the Internet.

In [BK94] Biham and Kocher demonstrated that the PKZIP stream cipher was weak and presented an attack requiring thirteen bytes of plaintext. Eight bytes of the plaintext must be contiguous, and all of the bytes must be the text that was encrypted, which is usually compressed data. [K92] shows that the compression method used at the time, *implode*, produces many predictable bytes suitable for mounting the attack.

Most zippers available today implement only one of the compression methods defined in APPNOTE.TXT, called *deflate*. *Deflate* uses Huffman coding followed by a variant of Lempel-Ziv. Once the dictionary reaches a certain size, the process starts over. Since the Huffman codes for any of the data depend on a great deal of surrounding data, one is forced to guess the plaintext unless one has the original data. The difficulty of getting known plaintext was one reason Phil Zimmerman decided to use *deflate* in PGP [PGP98]. Practically speaking, if one has enough of the original file to get the thirteen bytes of plaintext required for the attack in [BK94], one has enough to break the encryption almost instantly.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 125–134, 2002.

[redacted]

Mobile • 36m ago

[redacted] • 1:18 PM

Hi, Mike!
I read your work about zip known plaintext attacks. I forgot the password from the zip archive with important information and am looking for possible ways besides hashcat bruteforce. Maybe you follow the current status of this topic? If we find the password successfully, I will thank you;)

Mike Stay • 1:18 PM

Write a message...

GIF

Press Enter to Send

[redacted]

Mobile • 5h ago

[redacted] [redacted] • 4:46 PM

> Compare work done this year to find hash collisions in SHA-1, a 128-bit hash function, which requires 2^64 hashes and cost around $100K of specialized hardware and electricity costs.

I can say that I could spend so much on this archive. In any case, this is less than trying out a password from a bunch of characters or brute force all 3 zip keys.

Write a message...

GIF

Press Enter to Send

CRC32(key00, 0)				
MSB(key10 * 0x08088405**n)				
key20				

Stage:
Previous:
Current:
Total:
After filtering:

CRC32(key00, 0)				1
MSB(key10 * 0x08088405**n)				1
key20		1	1	1

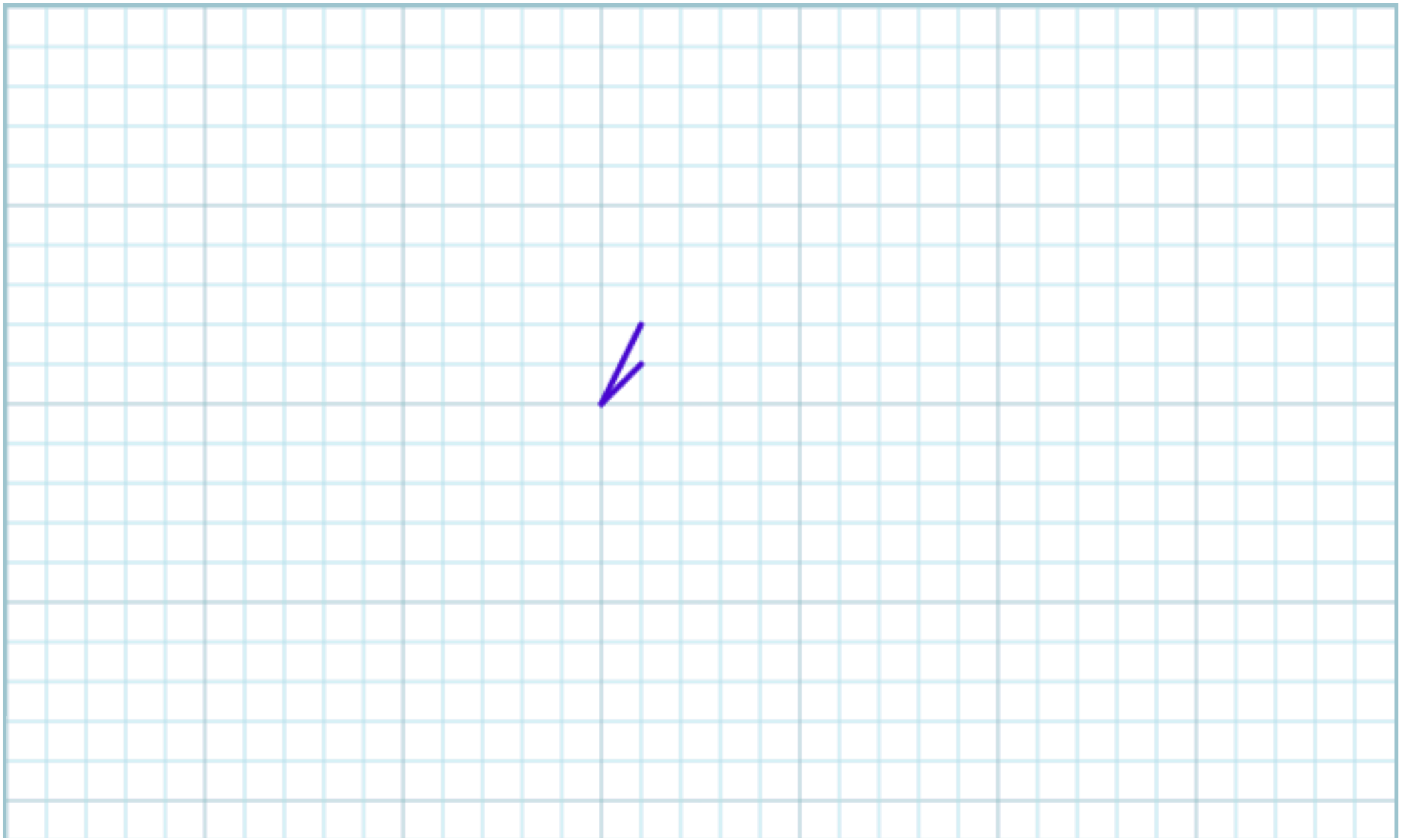
Stage: 1
Previous: 0
Current: 40 + 4 carry bits = 44
Total: 44
After filtering: 44 - 16 = 28

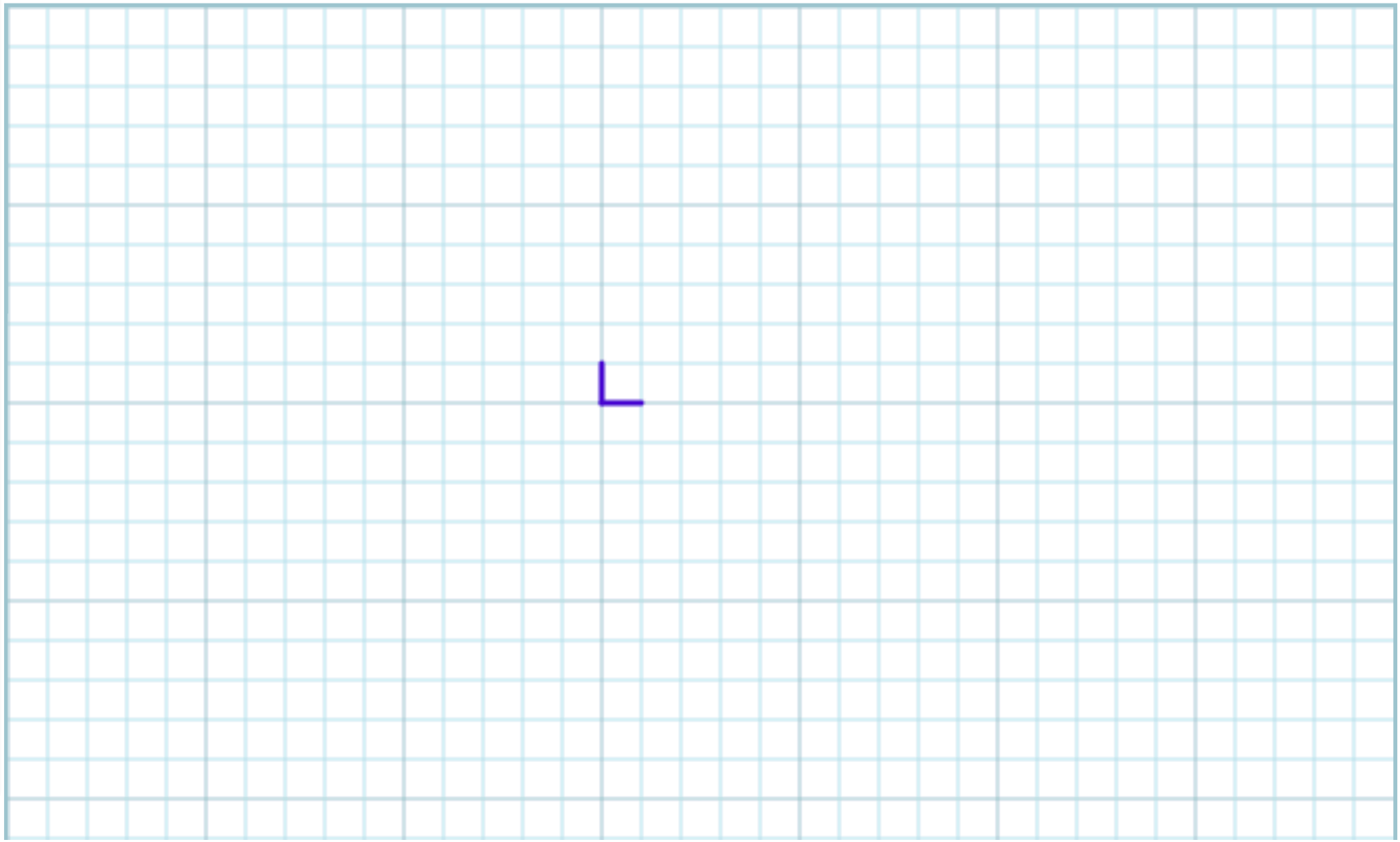
CRC32(key00, 0)			2	1
MSB(key10 * 0x08088405**n)			2	1
key20	2	1	1	1

Stage: 2
Previous: 28
Current: 24 + 4 carry bits = 28
Total: 28 + 28 = 56
After filtering: 56 - 16 = 40

CRC32(key00, 0)	4	3	2	1
MSB(key10 * 0x08088405**n)	4	3	2	1
key20	2	1	1	1

Stage: 4
Previous: 44
Current: 16 + 4 carry bits = 20
Total: 44 + 20 = 64
After filtering: 64 - 16 = 48





Home
Questions
Tags
Users
Unanswered

Breaking Truncated Linear Congruential Generator with known parameters

Ask Question

Asked 2 years, 3 months ago Active 2 years, 3 months ago Viewed 328 times

2

There is an elaborate discussion on the breaking of TLCG on the link below, where they show how to break the generator with known parameters given the most significant bits. Problem with LLL reduction on truncated LCG schemes

I tried to apply the same principles when given the least significant bits but with no success. On the paper by Frieze et al they discuss it briefly and mention substituting *x = 2s0*x(1) + x(2)* that helps a little bit but I cant figure out what the value of s0 is supposed to be. Is the anyone who can help.?

cryptanalysis lattice-crypto

share improve this question follow

edited Apr 28 '18 at 12:28
Paul Uszak
1

asked Apr 28 '18 at 10:57
Norman W
21 2

Is that substitution formula correct mathematics? – Paul Uszak Apr 28 '18 at 12:31

1 Could you state the givens in the problem at hand? Is the modulus prime, a power of two, other? – fgrieu ♦ Apr 28 '18 at 12:55

add a comment

1 Answer

Active Oldest Votes

1

According to the paper Freize et atll, for the most significant bit case which is discussed at length in the link,the modulas M must be odd, the addend I assume can be any number between 1 and the modulus, and the incriment must be zero. So given a*x(i) + b ~ mod M M is odd 0 > a < M b = 0 Because both a and M are known n we obtain high order bits y(i). Then x = y + z Where y is the high order bits n z is the lower order bits.

Lx ~ 0 mod M Taking B the reduced basis of yields. Bx ~ 0 mod M Substituting x = y + z gives B x + By ~ 0 mod M which yields the equation Bx + By = km for an unknown vector k of integers.

From this point its easy to find the lower order bits z, (see the link above).

In the case where we are give the lower significant bits z instead of the higher bits y, the paper suggests substituting x = 2(s0)* y + z. And they talk about finding the inverse of 2(s0) mod M which is guaranteed to exist since M is odd. But that's where I get completely lost.

share improve this answer follow

answered Apr 28 '18 at 17:08
Norman Bonda
11 2

The Overflow Blog
Podcast 256: You down with GPT-3? Yeah you know me!
Podcast – 25 Years of Java: the past to the present

Featured on Meta
Improved experience for users with review suspensions
CEO Blog: Some exciting news about fundraising
Could Wikipedia's articles be considered as a reference for CryptoSE posts?

Linked
8 Problem with LLL reduction on truncated LCG schemes

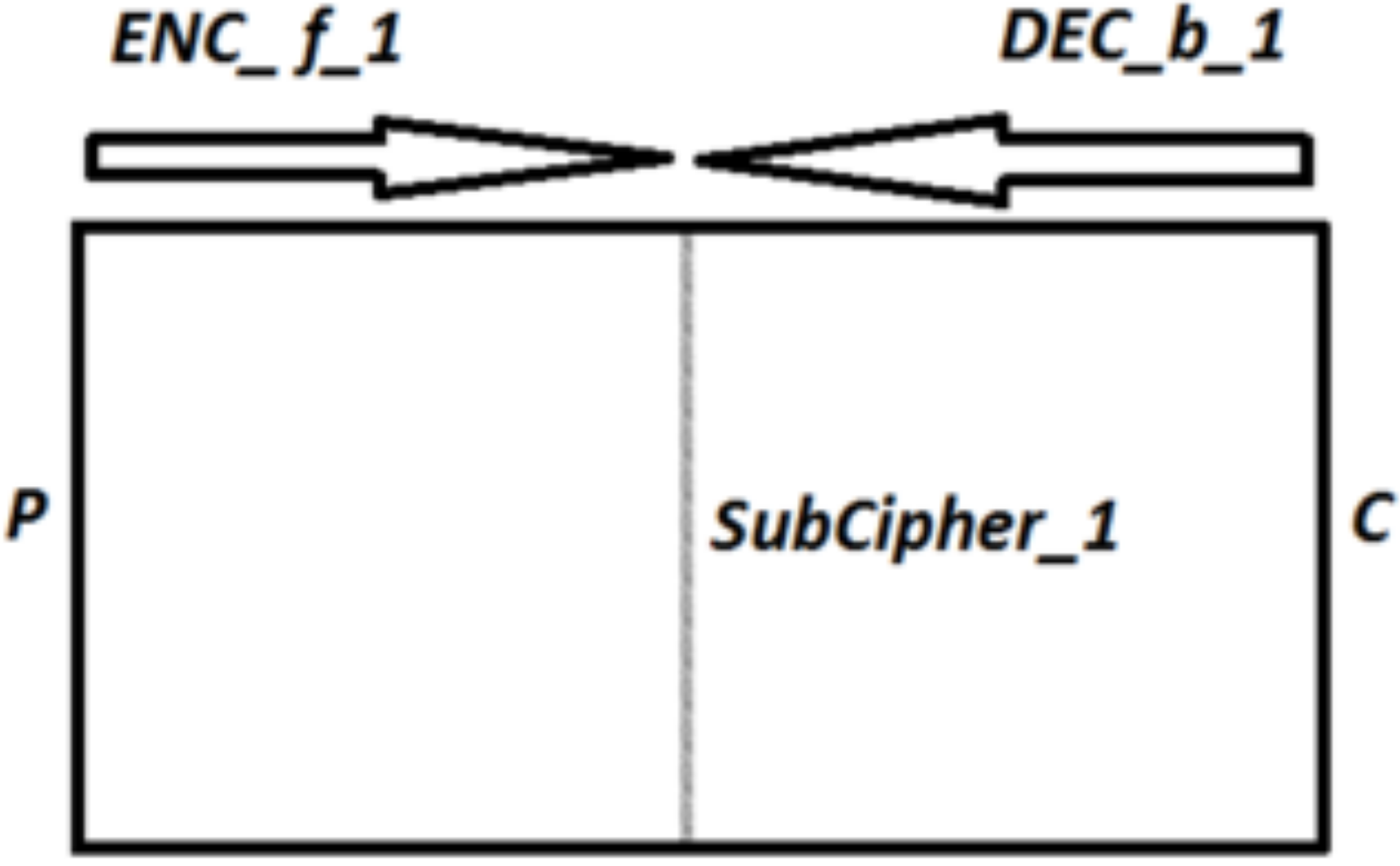
Related

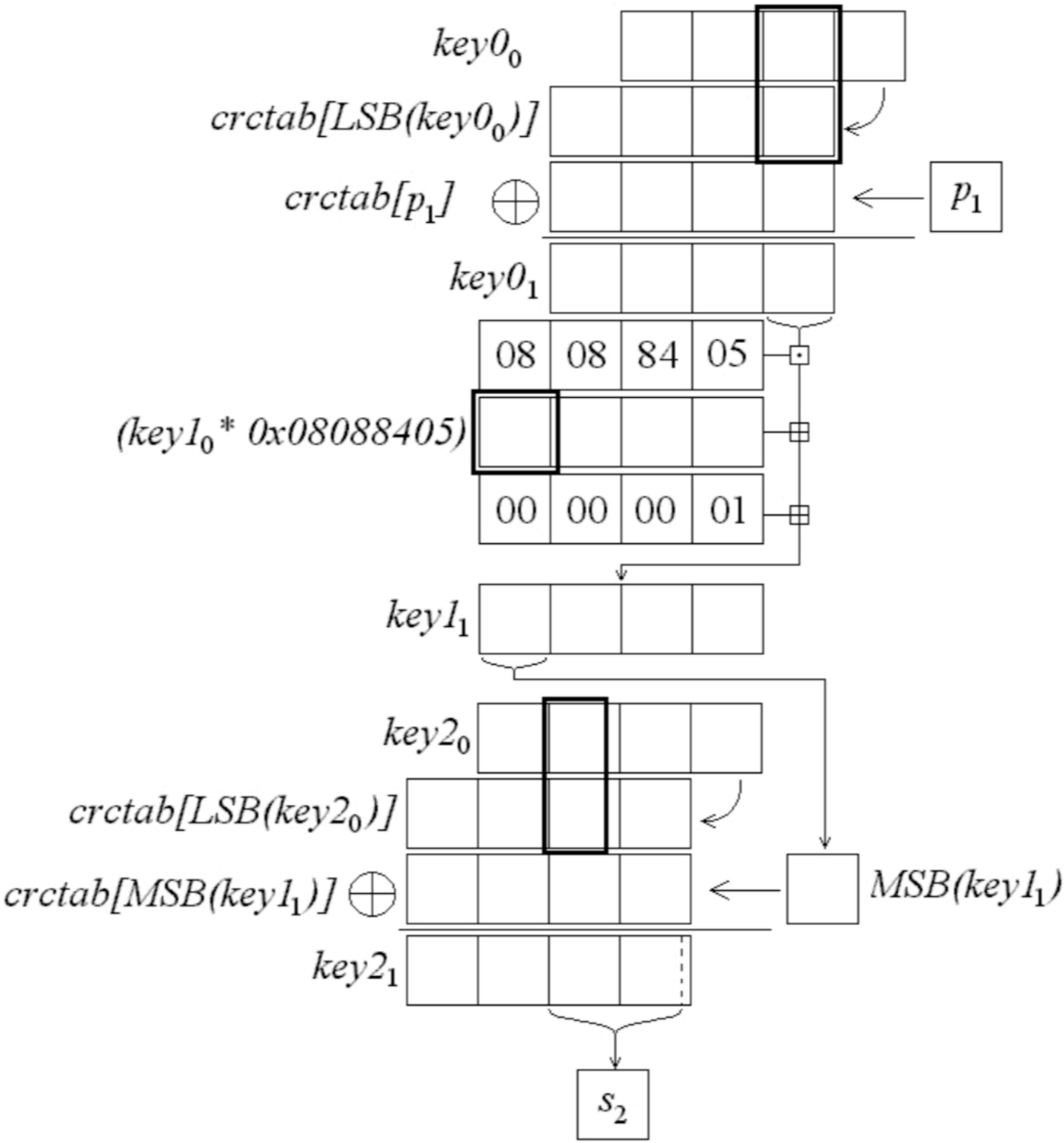
Hot Network Questions
If teachers account for 30% of variance of student achievement, can a teacher have 30% increase in achievement by teaching better?
Why not always take off with 10 degree flaps in a Cessna 172?
I just paid off my car - should I release my Credit Union's lien on the car?
How to plot these figures?
Tips on making a best and final offer
Is it possible to assign an integer value to a positional parameter in zsh?
What is the use of dual VFOs?

```
M = 2^32
c = 0x08088405
L = matrix([
    [  M,  0,  0,  0],
    [c^1, -1,  0,  0],
    [c^2,  0, -1,  0],
    [c^3,  0,  0, -1]
])
B = L.LLL()
size = 4

k10 = randint(0, M)
ks = [ c^(n + 1) * k10 % M for n in range(size) ]
print "ks: "
print map(hex, ks)
msbs = [(k & 0xff000000) for k in ks]
secret = [ks[i] - msbs[i] for i in range(size)]
w1 = B * vector(msbs)
w2 = vector([ round(RR(w) / M) * M - w for w in w1 ])
guess = list(B.solve_right(w2))
print "guess: "
# print [hex(Integer(guess[i])) for i in range(size)]
print guess

print "diff from msb + guess: "
# print [hex(Integer(ks[i] - msbs[i] - guess[i])) for i in
range(size)]
print vector(ks) - vector(msbs) - vector(guess)
```


ENC_ f_1
DEC_b_1
P
SubCipher_1
C


$key0_0$
$crctab[LSB(key0_0)]$
$crctab[p_1]$
p_1
$key0_1$
08 08 84 05
$(key1_0 * 0x08088405)$
00 00 00 01
$key1_1$
$key2_0$
$crctab[LSB(key2_0)]$
$crctab[MSB(key1_1)]$
$MSB(key1_1)$
$key2_1$
s_2

```
    }
}

void write_stage1_candidate_file(FILE *f,
                                 const vector<stage1_candidate> &candidates,
                                 const size_t start_idx, const size_t num) {
    fprintf(stderr,
            "write_stage1_candidate_file: writing %ld candidates "
            "out of %ld to file starting at index %ld.\n",
            num, candidates.size(), start_idx);

    write_word(f, num);
    auto end_idx = start_idx + num;
    for (size_t i = start_idx; i < end_idx; ++i) {
        write_stage1_candidate(f, candidates[i]);
    }
}

// info: the info about the archive to attack
// table: vector<vector<stage1a>> table(0x01000000)
void mitm_stage1a(archive_info &info, vector<vector<stage1a>> &table,
                  correct_guess *c) {
    // STAGE 1
    //
    // Guess s0, chunk2, chunk3 and carry bits.
    uint8_t xf0 = info.file[0].x[0];
    uint8_t xf1 = info.file[1].x[0];
    uint32_t extra(0);

    for (uint16_t s0 = 0; s0 < 0x100; ++s0) {
        fprintf(stderr, "%02x ", s0);
        if ((s0 & 0xf) == 0xf) {
            fprintf(stderr, "\n");
        }
        for (uint16_t chunk2 = 0; chunk2 < 0x100; ++chunk2) {
            for (uint16_t chunk3 = 0; chunk3 < 0x100; ++chunk3) {
                for (uint8_t carries = 0; carries < 0x10; ++carries) {
                    if (nullptr != c && s0 == c->sx[0][0] &&
                        chunk2 == c->chunk2 && chunk3 == c->chunk3 &&
                        carries == (c->carries >> 12)) {
                        fprintf(stderr, "On correct guess.\n");
                    }
                    uint8_t carryxf0 = carries & 1;
                    uint8_t carryyf0 = (carries >> 1) & 1;
                    uint8_t carryxf1 = (carries >> 2) & 1;
                    uint8_t carryyf1 = (carries >> 3) & 1;
                    uint32_t upper = 0x01000000;  // exclusive
                    uint32_t lower = 0x00000000;  // inclusive
```

```
uint32_t k0crc = chunk2;
uint32_t extra = 0;
uint8_t msbxf0 =
    first_half_step(xf0, false, chunk3, carryxf0, k0crc,
                    extra, upper, lower);
uint8_t yf0 = xf0 ^ s0;
k0crc = chunk2;
extra = 0;
uint8_t msbyf0 =
    first_half_step(yf0, false, chunk3, carryyf0, k0crc,
                    extra, upper, lower);
if (upper < lower) {
    if (nullptr != c && s0 == c->sx[0][0] &&
        chunk2 == c->chunk2 && chunk3 == c->chunk3 &&
        carries == (c->carries >> 12)) {
        fprintf(stderr,
                "Failed to get correct guess: s0 = %02x, "
                "chunk2 = %02x, "
                "chunk3 = "
                "%02x, carries = %x\n",
                s0, chunk2, chunk3, carries);
    }
    continue;
}
k0crc = chunk2;
extra = 0;
uint8_t msbxf1 =
    first_half_step(xf1, false, chunk3, carryxf1, k0crc,
                    extra, upper, lower);
if (upper < lower) {
    if (nullptr != c && s0 == c->sx[0][0] &&
        chunk2 == c->chunk2 && chunk3 == c->chunk3 &&
        carries == (c->carries >> 12)) {
        fprintf(stderr,
                "Failed to get correct guess: s0 = %02x, "
                "chunk2 = %02x, "
                "chunk3 = "
                "%02x, carries = %x\n",
                s0, chunk2, chunk3, carries);
    }
    continue;
}
uint8_t yf1 = xf1 ^ s0;
k0crc = chunk2;
extra = 0;
uint8_t msbyf1 =
    first_half_step(yf1, false, chunk3, carryyf1, k0crc,
```

```
                                    extra, upper, lower);
                if (upper < lower) {
                    if (nullptr != c && s0 == c->sx[0][0] &&
                        chunk2 == c->chunk2 && chunk3 == c->chunk3 &&
                        carries == (c->carries >> 12)) {
                        fprintf(stderr,
                                "Failed to get correct guess: s0 = %02x, "
                                "chunk2 = %02x, "
                                "chunk3 = "
                                "%02x, carries = %x\n",
                                s0, chunk2, chunk3, carries);
                    }
                    continue;
                }
                uint32_t mk = toMapKey(msbxf0, msbyf0, msbxf1, msbyf1);
                if (nullptr != c && s0 == c->sx[0][0] &&
                    chunk2 == c->chunk2 && chunk3 == c->chunk3 &&
                    carries == (c->carries >> 12)) {
                    fprintf(stderr,
                            "MSBs: %02x, %02x, %02x, %02x, Mapkey: %08x, "
                            "carries: %x, "
                            "c.carries: %04x\n",
                            msbxf0, msbyf0, msbxf1, msbyf1, mk, carries,
                            c->carries);
                }
                stage1a candidate = {uint8_t(s0), uint8_t(chunk2),
                                     uint8_t(chunk3), carries, msbxf0};
                table[mk].push_back(candidate);
            }
        }
    }
}

// info: the info about the archive to attack
// table: the output of mitm_stage1a
// candidates: an empty vector
void mitm_stage1b(const archive_info &info,
                  const vector<vector<stage1a>> &table,
                  vector<stage1_candidate> &candidates, const correct_guess *c,
                  size_t *correct_candidate_index) {
    // Second half of MITM for stage 1
    bool found_correct = false;
    for (uint16_t s1xf0 = 0; s1xf0 < 0x100; ++s1xf0) {
        for (uint8_t prefix = 0; prefix < 0x40; ++prefix) {
            uint16_t pxf0(preimages[s1xf0][prefix]);
            if (nullptr != c && s1xf0 == c->sx[0][1]) {
                fprintf(stderr, "s1xf0: %02x, prefix: %04x    ", s1xf0, pxf0);
```

```
            if ((prefix & 3) == 3) {
                fprintf(stderr, "\n");
            }
        }
        vector<uint8_t> firsts(0);
        uint8_t s1yf0 = s1xf0 ^ info.file[0].x[1] ^ info.file[0].h[1];
        second_half_step(pxf0, s1yf0, firsts);
        if (!firsts.size()) {
            continue;
        }
        for (uint16_t s1xf1 = 0; s1xf1 < 0x100; ++s1xf1) {
            vector<uint8_t> seconds(0);
            second_half_step(pxf0, s1xf1, seconds);
            if (!seconds.size()) {
                continue;
            }
            vector<uint8_t> thirds(0);
            uint8_t s1yf1 = s1xf1 ^ info.file[1].x[1] ^ info.file[1].h[1];
            second_half_step(pxf0, s1yf1, thirds);
            if (!thirds.size()) {
                continue;
            }
            for (auto f : firsts) {
                for (auto s : seconds) {
                    for (auto t : thirds) {
                        uint32_t mapkey(f | (s << 8) | (t << 16));
                        for (stage1a candidate : table[mapkey]) {
                            stage1_candidate g;
                            g.chunk2 = candidate.chunk2;
                            g.chunk3 = candidate.chunk3;
                            g.cb1 = candidate.cb;
                            g.m1 =
                                (candidate.msbk11xf0 * 0x01010101) ^ mapkey;

                            // Get ~4 possible solutions for lo24(k20) =
                            // chunks 1 and 4
                            //       A  B  C  D   k20
                            // ^  E  F  G  H      crc32tab[D]
                            //    ----------
                            //    I  J  K  L      crck20
                            // ^  M  N  O  P      crc32tab[msbk11xf0]
                            //    ----------
                            //    Q  R  S  T      (pxf0 << 2) matches k21xf0

                            // Starting at the bottom, derive 15..2 of KL
                            // from 15..2 of ST and OP
                            uint16_t crck20 =
                                ((pxf0 << 2) ^
```

```
                 crc32tab[candidate.msbk11xf0]) &
                0xfffc;

            // Now starting at the top, iterate over 64
            // possibilities for 15..2 of CD
            for (uint8_t i = 0; i < 64; ++i) {
                uint32_t maybek20 =
                    (preimages[candidate.s0][i] << 2);
                // and 4 possibilities for low two bits of D
                for (uint8_t lo = 0; lo < 4; ++lo) {
                    // CD
                    maybek20 = (maybek20 & 0xfffc) | lo;
                    // L' = C ^ H
                    uint8_t match =
                        (maybek20 >> 8) ^
                        crc32tab[maybek20 & 0xff];
                    // If upper six bits of L == upper six
                    // of L' then we have a candidate
                    if ((match & 0xfc) == (crck20 & 0xfc)) {
                        // KL ^ GH = BC.  (B = BC >> 8) &
                        // 0xff.
                        uint8_t b =
                            ((crck20 ^
                              crc32tab[maybek20 & 0xff]) >>
                             8) &
                            0xff;

                        if (g.k20_count >= g.MAX_K20S) {
                            fprintf(stderr,
                                    "Not enough space for "
                                    "k20 candidate in "
                                    "stage1_candidate.\n");
                            abort();
                        }

                        // BCD = (B << 16) | CD
                        g.maybek20[g.k20_count] =
                            (b << 16) | maybek20;
                        g.k20_count += 1;
                    }
                }
            }

            if (0 == g.k20_count) {
                continue;
            }

            candidates.push_back(g);
```

```
                                if (nullptr != c && s1xf0 == c->sx[0][1] &&
                                    s1xf1 == c->sx[1][1] &&
                                    candidate.s0 == c->sx[0][0] &&
                                    candidate.chunk2 == c->chunk2 &&
                                    candidate.chunk3 == c->chunk3 &&
                                    candidate.cb == (c->carries >> 12)) {
                                    found_correct = true;
                                    fprintf(stderr,
                                            "Correct candidates index = %lx\n",
                                            candidates.size() - 1);
                                    if (nullptr != correct_candidate_index) {
                                        *correct_candidate_index =
                                            candidates.size() - 1;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

    if (c != nullptr && !found_correct) {
        fprintf(stderr,
                "Failed to use correct guess: s1xf0 = %02x, s1xf1 = %02x\n",
                c->sx[0][1], c->sx[1][1]);
    }
    fprintf(stderr, "Stage 1 candidates.size() == %04lx\n", candidates.size());
}

};  // namespace mitm_stage1
```

```
__global__ void gpu_stage3_kernel(const gpu_stage2_candidate *candidates,
                                  keys *results,
                                  const archive_info* archive,
                                  const uint32_t stage2_candidate_count,
                                  const mitm::correct_guess& c) {
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if (i < stage2_candidate_count) {
        keys result = {0, 0, 0};
        stage3::gpu_stage3_internal(*archive, candidates[i], &result, &c);
        results[i].crck00 = result.crck00;
        results[i].k10 = result.k10;
        results[i].k20 = result.k20;
    }
}
```

breakzip

Project ID: 14482548

12

Utilities for cracking encrypted zip files that use weak encryption.

Merge branch 'cuda-fail-fast' into 'master'

Mike Stay authored 4 months ago

`f2e30c52`

Name	Last commit	Last update
cmake/Modules	Add a cmake module for tcmalloc and use it by default in the build.	6 months ago
doc	Initial commit.	10 months ago
scripts	Fix the gce install script.	5 months ago
src	Merge branch 'cuda-fail-fast' into 'master'	4 months ago
third-party	Fix zip build	5 months ago
.clang-format	Tweaks to clang format.	6 months ago
.gitignore	Remove some more files autotools should generate.	5 months ago
CMakeLists.txt	Add -DDEBUG flag to debug builds.	5 months ago
Copyright.txt	recoverseed: Recover srand initial seed.	9 months ago
LICENSE	recoverseed: Recover srand initial seed.	9 months ago
README	Initial commit.	10 months ago
README.md	Finalize cleanup of build.sh/clean.sh in check directory. Fixup READ...	5 months ago
breakzip.dox	Initial commit.	10 months ago
build.sh	Remove broken clang formatting from build.sh	6 months ago
clean.sh	Initial commit.	10 months ago
contributors.txt	Initial commit.	10 months ago
format.sh	Add format.sh script.	5 months ago
run_stage1.sh	Rejigger scripts.	5 months ago
run_stage2.sh	Tweaks.	5 months ago
run_stage3.sh	Remove head -n from stage3 script.	5 months ago
split_stage2.sh	Add stage3 scripts.	5 months ago
test.zip	Modify zipcloak and crypt.c to print seeds so we can test recoverseed.	9 months ago
test_encrypted.zip	Modify zipcloak and crypt.c to print seeds so we can test recoverseed.	9 months ago

README.md

Welcome to breakzip! This is a collection of open source utilities for working with Zip files and cracking (hopefully) their encryption key. This project is written and maintained by:

- Mike Stay (stay@pyrofex.net)