Skema Enkripsi Pada WinZip dan Serangannya

Andresta RamadhanSetiawan
(13503030)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
Email : if13030@students.if.itb.ac.id

Abstrak
Winzip merupakan aplikasi kompresi yang sangat popular pada system operasi Windows. Selain melakukan kompresi, untuk menjaga kerahasiaan arsip, Winzip menyediakan fitur enkripsi. Versi terbaru yang ada di situs resminya yaitu Winzip 10 menggunakan enkripsi AES untuk melindungi arsip rahasia pengguna. Untuk mendukung enkripsi pada versi Winzip yang lama (sebelum versi 9), pada versi terbaru ini juga masih disertakan enkripsi tradisional ZIP 2.0. Pada makalah ini akan dijelaskan bagaimana mengenai format berkas zip, bagaimana skema enkripsi yang digunakan oelh WinZip untuk melindungi berkas-berkas rahasia. Skema Enkripsi WinZip rentan terhadap serangan seperti *known-plaintext attack*, *random seed attack*. Serangan menggunakan *bruteforce* atau *dictionary attack* juga cukup berbahaya jika password yang digunakan pendek, dalam waktu yang singkat password dapat diperoleh. Hal ini yang biasanya digunakan program *password recovery* untuk mendapatkan *password*.

Kata Kunci : *WinZIP, Known Plaintext Attack, plaintext, chipertext.*

1. Pendahuluan

ZIP format adalah salat satu dari sekian banyak format yang paling banyak digunakan oleh program kompresi dan pengarsipan data pada berbagai macam sistem operasi. Satu atau lebih berkas (file) yang telah dikompres dapat dimuat pada satu berkas zip (zip file). Zip file menjaga berkas-berkas yang saling berhubungan tetap menyatu sehingga akan mudah untuk disebarluaskan baik itu lewat email, mengunduh berkas dari internet selain itu kemampuannya untuk mengkompresi data membuat penyimpanan berkas zip lebih efisien. Berikut ini adalah beberapa keuntungan menyimpan berkas dalam format zip

Memudahkan pendistribusian data lewat internet.
Pendistribuasian data lewat internet akan lebih mudah dan cepat, terutama data yang terdiri lebih dari satu arsip. Hanya dengan unduh untuk mengambil satu atau lebih berkas yang dibutuhkan selain itu prosesnya pun akan lebih cepat karena semua berkas yang diarsipkan menjadi berkas zip telah terkompresi sehingga lebih kecil ukurannya dari berkas aslinya.

Penghematan ruang penyimpanan. Seperti telah disebutkan, berkas yang diarsipkan menjadi berkas zip akan dikompresi sehingga ukurannya lebih kecil. Hal ini bermanfaat untuk menghemat ruang penyimpanan pada diska. Jika anda memiliki berkas yang berukuran besar dan jarang digunakan arsipkan saja menjadi berkas zip dan unzip hanya jika berkas tersebut akan digunakan.

Menjaga berkas sekelompok berkas tetap utuh. Ketika mengirimkan berkas-berkas dalam satu berkas zip tentunya akan lebih menguntungkan, karena kemungkinan berkas 'tercecer' atau hilang sangat kecil.

2. Berkas .ZIP

Sejarah
Berkas format ZIP diciptakan oleh Phil Katz, pendiri PKWARE, setelah perselisihan mengenai hak paten merek dagang ARC (kependekan dari archieve) dan ekstensi nama berkas .arc antara

PKWare dengan System Enhancement Associates (SCA).

Produk pengarsipan PKWare yang pertama adalah PKARC menggunakan kode sumber yang dipublikasikan oleh SCA. Kode sumber SCA yang ditulis dalam bahasa C diubah menggunakan bahasa asembly (mesin) sehingga lebih cepat prosesnya. PKArc juga menggunakan ekstensi nama berkas yang sama dengan yang digunakan SCA yaitu ".arc". Oleh karean itu, SCA menuntut royalti atas penggunaan merek dagang dan kode sumbernya. Akan tetapi PKWare menolak. SCA memenangkan tuntutan di pengadilan atas pelanggaran merek dagang oleh PKWare. Akhirnya Katz merilis ulang produknya dengan mengganti nama menjadi PKPAK.

Walaupun sudah mengganti nama menjadi PKPAK, Katz tetap harus membayar lisensi atas peggunaan kode sumber SEA. Karena Katz tetap menolak untuk membayar royalti akhirnya ia membuat format berkas pengarsipan sendiri, yang sampai sekarang terkenal diseluruh dunia sebagai format ZIP. Format ZIP yang didesain Katz lebih kokoh dan tahan terhadap korupsi data dari pada format ARC karena katolag data yang "redundan". Selain itu, ZIP juga lebih fleksibel karena memungkinkan adanya tambahan algoritma untuk kompresi data dan juga tempat unuk pengembangan lebih jauh. Ketika perangkat lunak PKZIP dirilis, banyak orang yang mulai meninggalkan format ARC karena dinilai lebih lambat dan tidak efektif performanya dalam melakukan kompresi data.

Katz mempublikasikan dokumen teknis format berkas ZIP bersamaan dengan dikeluarkannya versi pertama perangkat lunak pengarsipan PKZIP pada bulan Januari 1989. Penggunaan nama ZIP (yang berarti cepat) diusulkan oleh Robert Mahoney. Mereka menginginkan produknya lebih cepat dari pada ARC dan format kompresi yang lainnya.

Ketika antarmuka berbasis grafis (graphical user interface) mulai banyak digunakan pada sistem operasi, mulai banyak program kompresi dan pengarsipan bebasi grafis. Kebanyakan program-program tersebut menggunakan format ZIP. Pada akhir tahun 1990 telah banyak aplikasi pengaturan berkas (file manager) yang mengintegrasikan format ZIP bahkan Microsoft bahkan mengintegrasikan format ZIP pada sistem operasi Windowsnya. Hal ini membuat format ZIP semakin dikenal dan digemari sehingga dapat dikatakan ZIP telah menjadi standar untuk kompresi dan pengarsipan berkas. Hampir semua program pengarsipan dan kompresi berkas saat ini mendukung format ZIP. Salah satu program dari sekian banyak yang ada dan banyak digunakan, dan yang akan dibahas dalam makalah ini, adalah WinZip.

Informasi Teknis

Spesifakasi format ZIP mengindikasikan bahwa berkas-berkas dapat disimpan tanpa kompresi atau dengan menggunakan variasi alogoritma kompresi lainnya. Pada kenyetaannya, format ZIP yang digunakan menggunakan algoritma DEFLATEnya Katz, kecuali ketika berkas yang ditambahkan pada arsip telah dikompresi terlebih dahulu atau berkas tersebut akan rusak jika dikompresi.

Untuk masalah keamanan berkas yang diarsipkan, ZIP mendukung penggunaan kata sandi untuk mengenkripsi menggunakan enkripsi kunci simetri. Beberapa fitur baru ditambahkan seperti metode enkripsi dan kompresi yang lebih mutahir. Tapi fitur ini tidak didukung oleh semua program pengarsipan dan kompresi yang ada dipasaran. Salah satu program yang mendukung enkripsi adalah WinZip, yang akan dibahasah pada makalah ini.

3. WinZip

WinZip adalah sebuah program pengarsipan dan kompresi berkas yang cukup banyak digunakan dan berjalan pada sistem operasi Windows. Dikembangkan oleh WinZip Computing, dulunya Nico Mak Computing, WinZip menggunakan format PKZIPnya PKWARE dengan dukungan tambahan terhadap format kompresi lainnya.

WinZip mulai memasarkan produknya pada tahun 1990an sebagai antarmuka grafis untuk PKZIP. Pada sekitar tahun 1996, pencipta WinZip menggabungkan kode kompresi yang berasal dari proyek Info-ZIP, kemudian menghilangkan kebutuhan akan eksekusi PKZIP untuk disertakan pada produk WinZip. Pada tanggal 2 Mei 2006, Corel Corporation mengakuisisi WinZip Computing.

Yang membuat WinZip banyak digunakan orang adalah karena fitur-fitur yang ditawarkannya, antara lain:

- mengkompresi dan melakukan "unzipping" format arsip PKZIP.
- integrasi terhadap shell Windows
- mengenkripsi berkas rahasia menggunakan enkripsi kunci simetri
- menulis arsip ZIP langsung ke CD maupun DVD
- versi 9.0 mengimplementasikan format PKZIP versi 64-bit yang mendukung lebih dari 65.535 berkas dalam satu arsip zip dan ukuran arsip ZIP yang lebih dari 4 Gigabyte
- versi 10.0 mendukung bzip2 dan algoritma kompresi PPMd yang menghasilkan berkas arsip yang lebih kecil.
- dapat bekerja dengan arsip ARC, ARJ, dan LHA.
- mendukung format kompresi TAR, Z, GZ, TAZ, dan TGZ yang biasa digunakan pada lingkungan UNIX.

Dari fitur-fitur diatas, yang menjadi sorotan adalah masalah keamanan yang ditawarkan oleh WinZip. Enkripsi yang disediakan WinZip harus dapat menjamin pengguna yang memiliki kunci saja yang dapat mengakses berkas ZIP yang telah dienkripsi.

Winzip untuk versi 9.0 keatas mendukung 128 dan 256 bit kunci enkripsi AES yang tentunya menyediakan tingkat keamanan yang lebih dibandingkan metode enkripsi Zip 2.0 yang digunakan pada WinZip sebelum versi 9.0. AES enkripsi pada WinZip yang telah disertifikasi FIPS-197 menggunakan algoritma kriptografi Rijndael. National Institute of Standards and Technology (NIST) mengumumkan bahwa AES merupakan teknik enkripsi yang dapat digunakan oleh pemerintah, kalangan bisnis maupun perorangan karena telah terbukti kuat cepat dan efisien dalam pengoprasiaannya.

Ada beberapa kemudahan yang ditawarkan WinZip pada teknologi enkripsi terbarunya yang memudahkan pengguna untuk melakukan enkripsi. Berkas-berkas yang telah menjadi berkas Zip dapat dienkripsi, ini berbeda dengan versi sebelumnya yang hanya dapat menenkripsi berkas pada saat proses pembentukan berkas Zip.

4. Enkripsi Pada WinZip

Traditional PKWARE Encryption

Enkripsi ZIP 2.0 berdasarkan pada enkripsi tradisional PKWARE. Enkripsi ini sangat lemah berdasarkan standar keamanan saat ini sehingga penggunaannya hanya untuk keperluan keamanan yang rendah, tidak direkomendasikan untuk melakukan enkripsi pada berkas yang sangat rahasia. Alasan enkripsi ZIP 2.0 ini masih disertakan adalah untuk mendukung kompatibilitas terhadap aplikasi ZIP yang dahulu yang mungkin masih digunakan sampai saat ini.

Proses enkripsi dilakukan pada aliran data yang telah terkompresi. Berkas yang terenkripsi harus didekripsi terlebih dahulu sebelum diekstraksi menjadi berkas-berkas asli.

Setiap berkas yang dienkripsi memiliki 12 byte tambahan yang disimpan pada awal area data yang isinya mendefinisikan "header" enkripsi untuk berkas tersebut. Header enkripsi tersenbut diisi dengan bilangan acak (random) dan kemudian dienkripsi menggunakan 3 buah kunci masing-masing 32-bit. Ketiga kunci tersebut diinisialisasi oleh kata kunci yang diisikan. Setelah tiap byte data dienkripsi, kunci-kunci tersebut kemudian diperbarui menggunakan teknik pembangkitan bilangan pseudo-random yang dikombinasikan dengan algoritma CRC-32

Berikut ini adalah langkah-langkah secara umum yang dilakukan untuk mendekripsi berkas ZIP :

1. Inisialisasi tiga buah kunci 32-bit dengan kata kunci.
2. Baca dan dekripsi 12 byte header enkripsi, kemudian inisialisasi kunci enkripsi
3. Baca dan dekripsi aliran data yang terkompres menggunakan kunci enkripsi yang didapat dari langkah dua

berikut ini adalah algorima dari tiap langkah diatas.

1. Menginisialisasi kunci enkripsi

```
Key(0) <- 305419896
Key(1) <- 591751049
Key(2) <- 878082192

loop for i <- 0 to length(password)-1
   update_keys(password(i))
end loop

update_keys(char):
Key(0) <- crc32(key(0),char)
Key(1) <- Key(1) + (Key(0) & 000000ffH)
Key(1) <- Key(1) * 134775813 + 1
Key(2) <- crc32(key(2),key(1) >> 24)
end update_keys
```

Dimana crc32(old_chr,char) adalah sebuah fungsi yang menerima masukan nilai CRC dan sebuah karakter, mengembalikan nilai CRC yang baru menggunakan algoritma CRC-32.

2. Dekripsi header yang terenkripsi
   Tujuan langkah ini adalah untuk menginisialisasi kunci enkripsi, berdasarkan data acak, untuk membuat serangan dengan plainteks pada data menjadi tidak efektif.

   Baca 12-byte header enkripsi pada buffer, dari buffer ke-0 hingga ke-11

```
loop for i <- 0 to 11
    C     <-     buffer(i)     ^
decrypt_byte()
    update_keys(C)
    buffer(i) <- C
end loop
```

   algoritma decrypt_byte() adalah sebagai berikut :

```
unsigned char decrypt_byte()
    local unsigned short temp
    temp <- Key(2) | 2
    decrypt_byte   <-   (temp   *
(temp ^ 1)) >> 8
end decrypt_byte
```

   Setelah header didekripsi, satu atau dua byte terakhir pada buffer seharusnya byte urutan atas pada CRC untuk berkas yang didekripsi. Satu byte CRC check digunakan untuk mengetes apakah kata kunci yang diberikan benar atau salah.

3. Dekripsi aliran data yang terkompresi

   algoritma untuk mendekripsi aliran data yang terkompresi adalah sebagai berikut :

```
loop until done
    read a character into C
    Temp <- C ^ decrypt_byte()
    update_keys(temp)
    output Temp
end loop
```

AE-2

enkripsi AE-2 (AES Encrption versi 2) yang mulai diterapkan pada WinZip 9.0 merupakan perbaikan dari AE-1.

Untuk melakukan enkripsi dan dekripsi AES , WinZip menggunakan fungsi AES yang ditulis oleh Dr. Brian Gladman. Fungsi enkripsi Dr. Gladman dapat digunakan pada berbagai macam sistem operasi dan dapat di link secara statis pada aplikasi yang membutuhkan tanpa membutuhkan library apapun.

5. ZIP File Format

Sama seperti enkripsi ZIP 2.0 AES enkripsi membutuhkan tempat untuk tambahan data tapi bedanya nilai CRCnya adalah 0. Format dasar berkas ZIP tidak berubah.

Untuk berkas yang dienkripsi, bit 0 dari "general purpose bit flags" harus diisi 1 pada tiap header lokal berkas yang dienkripsi dengan AES. Keberadaan berkas yang dienkripsi dengan AES pada berkas ZIP ditandai dengan kode metode kompresi yang baru (99 desimal) pada header lokal berkas dan central directory entry, digunakan bersamaan dengan tempat data tambahan AES. Kode sesungguhnya untuk metode kompresi disimpan pada tempat data tambahan AES. lihat tabel 1.

Enkripsi berkas menggunakan metode AE-2, nilai format standar ZIP tidak digunakan dan 0 harus diisikan pada kolom ini. Sebagai gantinya, korupsi data akan dideteksi oleh kolom kode autentifikasi (authentication code). Sedangkan pada enkripsi berkas menggunakan metode AE-1, nilai standar ZIP CRC tetap disertakan. Untuk membedakan penggunaan AE-1 atau AE-2 kolom data tambahan diisi dengan 0x0001 untuk AE-1 dan 0x0002 untuk AE-2.

Berkas yang dienkripsi menggunakan enkripsi AES akan memiliki kolom data tambahan diasosiasikan dengan berkas tersebut. Data tambahan ini disimpan pada header lokal dan central directory entry untuk berkas tsb. ID header data tambahan untuk enkripsi AES adalah 0x901. Panjang data ini adalah 11 byte dengan alokasi 7 byte ditambah 2 byte untuk ID header dan 2 byte untuk ukuran data. Oleh karena itu, data tambahan untuk tiap berkas adalah 22 byte, 11 disimpan pada header pusat dan 11 lagi pada header lokal berkas.
Format data tambahan untuk AES adalah sebagai berikut :

Offset	Size (bytes)	Content
0	2	Extra field header ID (0x9901)
2	2	Data size (currently 7, but subject to possible increase in the future)
4	2	Integer version number specific to the zip vendor
6	2	2-character vendor ID
8	1	Integer mode value indicating AES encryption strength
9	2	The actual compression method used to compress the file

Keterangan

data size:	saat ini nilainya diisi 7, tetapi spesifikas AE-2 memungkinkan untuk mengubah data ini untuk tambahan kolom data.
Vendor ID:	kolom data tambahan AES didesain untuk dapat diperluas sehingga vendor lain dapat menambahakan variasi mereka pada enkripsi non-ZIP 2.0
Vendor version:	versi vendor untuk AE-2 adalah 0x0002. WinZip yang mendukung AE-2 harus juga dapat memproses berkas yang denkripsi dengan format AE-1, dimana vendor versinya adalah 0x0001. Penanganan nilai CRC merupakan hal yang membedakan antara AE-1 dengan AE-2.
Encryption strength:	nilai mode untuk AE-2 adalah

Value	Strength
0x01	128-bit encryption key
0x02	192-bit encryption key
0x03	256-bit encryption key

	spesifikasi AE-2 hanya mendukung 128-, 192-, 256 bit kunci enkripsi. Walaupun pada kenyataannya WinZip tidak mendukung penggunaan kunci 192-bit untuk melakukan enkripsi. Panjang kunci selain itu tidak diperbolehkan.
Compression method:	metode kompresi adalah salah satu data yang disimpan pada header lokal maupun pusat berkas tersebut. Metode kompresi 99 desimal digunakan untuk memberi tanda bahwa ada berkas yang dienkripsi menggunakan AES.

Encrypted file storage format

data tambahan yang dibutuhkan untuk melakukan dekripsi disimpan pada berkas yang dienkripsi tersebut, bukan pada header. Format yang disimpan pada berkas adalah sebagai berikut :

Size (bytes)	Content
Variable	Salt value
2	Password verification value
Variable	Encrypted file data
10	Authentication code

catatan bahwa nilai kolom "compressed size" dari header berkas lokal dan central directory entry adalah ukuran total dari item yang ada diatas yaitu ukuran total dari nilai "salt", kata kunci, data terenkripsi, dan kode otentifikasi.

[salt value]

salt atau salt value adalah nilai acak atau acak-semu ragkaian byte yang dikombinasikan dengan kata kunci enkripsi (encryption password) untuk menciptakan kunci enkripsi (encyrption key)dan kunci otentikasi (authentication key). Nilai salt dihasilkan oleh aplikasi enkripsi kemudian disimpan tanpa terenkripsi bersama berkas data.

Aplikasi kriptografi yang baik harus menghasilkan nilai salt yang berbeda untuk tiap berkas yang dienkripsi walaupun dengan kata kunci (password) yang sama. Jika kata kunci yang sama digunakan pada berkas yang dienkripsi menghasilkan nilai salt yang sama maka akan memudahkan penyerang untuk mendapatkan plainteks semua berkas jika satu saja plainteks telah diketahui. Ingat bahwa nilai salt tidaklah dienkripsi. Oleh karena itu nilai salt untuk tiap berkas harus unik.

Pada implementasi enkripsi menggunakan AE-2 ukuran nilai saltberdasarkan panjang kunci enkripsi yang digunakan, lihat tabel 2.

Key size	Salt size
128 bits	8 bytes
192 bits	12 bytes
256 bits	16 bytes

Dr. Gladman menyediakan fungsi generator bilangan acak-semu pada berkas PRNG.C dan PRNG.H Berikut ini adalah cara mendapatkan salt value menggunakan fungsi dari Dr. Gladman.

1. Kita sebelumnya harus membuat fungsi entropi sebagai inisialisasi generator bilangan acak. Berikut ini adalah fungsi entropy yang diunakan :

   ```
   int entropy(unsigned char buf[],
   unsigned int len)
   ```

2. Inisialisasi generator dengan memanggil prng_init() dengan parameter masukan hasil dari fungsi entropi pada langkah satu dan sebuah instance dari sturktur prng_ctx.

   ```
   prng_ctx ctx;
   prng_init(entropy_fun, &ctx);
   ```

3. Untuk mendapatkan rangkaian byte acak, gunakan fungsi prng_rand(). Fungsi ini akan menghasilkan 16 byte acak, yang akan digunakn sebagai salt value untuk enkripsi AES 256-bit.

   ```
   unsigned char buffer[16];
   prng_rand(buffer,
   sizeof(buffer), &ctx);
   ```

4. Jika selesai menggunakan generator, tutup dengan memanggil fungsi prng_end

   ```
   prng_end(&ctx);
   ```

Fungsi entropi pada langkah satu adalah sebagai berikut :

```
int entropy(unsigned char buf[], unsigned int len)
{
unsigned __int64 pentium_tsc[1];
unsigned int i;
// Note: check QueryPerformanceFrequency() first to
// ensure that QueryPerformanceCounter() will work.
QueryPerformanceCounter((LARGE_INTEGER *)pentium_tsc);
for (i = 0; i < 8 && i < len; ++i)
buf[i] = ((unsigned char*)pentium_tsc)[i];
return i;
}
```

Isi dari PNRG.H adalah sebagai berikut :

```
#ifndef _PRNG_H
#define _PRNG_H

#include "sha1.h"

#define  PRNG_POOL_LEN        256        /*
minimum random pool size             */
#define  PRNG_MIN_MIX         20      /* min
initial pool mixing iterations   */

/* ensure that pool length is a multiple
of the SHA1 digest size        */

#define PRNG_POOL_SIZE  (SHA1_DIGEST_SIZE
*   (1   +   (PRNG_POOL_LEN   -   1)   /
SHA1_DIGEST_SIZE))

#if defined(__cplusplus)
extern "C"
{
#endif

/* A function for providing entropy is a
parameter in the prng_init()   */
/* call.  This function has the following
form and returns a maximum    */
/* of 'len' bytes of pseudo random data
in the buffer 'buf'.  It can    */
/* return less than 'len' bytes but will
be repeatedly called for more  */
/*      data      in      this      case.
*/

typedef  int  (*prng_entropy_fn)(unsigned
char buf[], unsigned int len);

typedef struct
{   unsigned char   rbuf[PRNG_POOL_SIZE];
/* the random pool          */
    unsigned char   obuf[PRNG_POOL_SIZE];
/* pool output buffer       */
    unsigned    int                  pos;
/* output buffer position   */
    prng_entropy_fn             entropy;
/* entropy function pointer */
} prng_ctx;

/* initialise the random stream generator
*/
void    prng_init(prng_entropy_fn    fun,
prng_ctx ctx[1]);

/* obtain random bytes from the generator
*/
void   prng_rand(unsigned   char   data[],
unsigned int data_len, prng_ctx ctx[1]);
```

```
/*  close  the  random  stream  generator
*/
void prng_end(prng_ctx ctx[1]);

#if defined(__cplusplus)
}
#endif

#endif
```

Isi dari PNRG.C adalah sebagai berikut :

```
#include <memory.h>
#include "prng.h"

#if defined(__cplusplus)
extern "C"
{
#endif

/* mix a random data pool using the SHA1
compression function (as   */
/*  suggested  by  Peter  Gutmann  in  his
paper on random pools)         */

static void prng_mix(unsigned char buf[])
{   unsigned int    i, len;
    sha1_ctx        ctx[1];

    /*lint  -e{663}    unusual  array  to
pointer conversion */
    for(i = 0; i < PRNG_POOL_SIZE; i +=
SHA1_DIGEST_SIZE)
    {
        /*  copy  digest  size  pool  block
into SHA1 hash block */
        memcpy(ctx->hash, buf + (i ? i :
PRNG_POOL_SIZE)
                            -
SHA1_DIGEST_SIZE, SHA1_DIGEST_SIZE);

        /*  copy  data  from  pool  into  the
SHA1 data buffer    */
        len = PRNG_POOL_SIZE - i;
        memcpy(ctx->wbuf, buf + i, (len >
SHA1_BLOCK_SIZE   ?   SHA1_BLOCK_SIZE   :
len));

        if(len < SHA1_BLOCK_SIZE)
            memcpy(((char*)ctx->wbuf)    +
len, buf, SHA1_BLOCK_SIZE - len);

        /*   compress   using   the   SHA1
compression function     */
        sha1_compile(ctx);

        /*  put  digest  size  block  back
into the random pool  */
        memcpy(buf    +    i,    ctx->hash,
SHA1_DIGEST_SIZE);
    }
}

/* refresh the output buffer and update
the random pool by adding   */
/*       entropy       and       remixing
*/

static void update_pool(prng_ctx ctx[1])
{   unsigned int    i = 0;

    /* transfer random pool data to the
output buffer   */
    memcpy(ctx->obuf,           ctx->rbuf,
PRNG_POOL_SIZE);

    /* enter entropy data into the pool
*/
    while(i < PRNG_POOL_SIZE)
        i  +=  ctx->entropy(ctx->rbuf  +  i,
PRNG_POOL_SIZE - i);

    /* invert and xor the original pool
data into the pool  */
    for(i = 0; i < PRNG_POOL_SIZE; ++i)
        ctx->rbuf[i] ^= ~ctx->obuf[i];

    /* mix the pool and the output buffer
*/
    prng_mix(ctx->rbuf);
    prng_mix(ctx->obuf);
}

void    prng_init(prng_entropy_fn    fun,
prng_ctx ctx[1])
{   int i;

    /* clear the buffers and the counter
in the context     */
    memset(ctx, 0, sizeof(prng_ctx));

    /*  set  the  pointer  to  the  entropy
collection function   */
    ctx->entropy = fun;

    /*  initialise  the  random  data  pool
*/
    update_pool(ctx);

    /* mix the pool a minimum number of
times               */
    for(i = 0; i < PRNG_MIN_MIX; ++i)
        prng_mix(ctx->rbuf);

    /* update the pool to prime the pool
output buffer      */
    update_pool(ctx);
}

/* provide random bytes from the random
data pool   */

void   prng_rand(unsigned   char   data[],
unsigned int data_len, prng_ctx ctx[1])
{   unsigned char   *rp = data;
    unsigned int    len, pos = ctx->pos;

    while(data_len)
    {
        /* transfer 'data_len' bytes (or
the number of bytes remaining  */
        /*  the  pool  output  buffer  if
less) into the output             */
        len  =  (data_len  <  PRNG_POOL_SIZE
- pos ? data_len : PRNG_POOL_SIZE - pos);
        memcpy(rp, ctx->obuf + pos, len);
        rp  +=  len;             /* update
ouput buffer position pointer     */
        pos  +=  len;            /* update
pool output buffer pointer        */
        data_len -= len;    /* update the
remaining data count          */
```

```
        /* refresh the random pool if
necessary */
        if(pos == PRNG_POOL_SIZE)
        {
            update_pool(ctx); pos = 0;
        }
    }

    ctx->pos = pos;
}

void prng_end(prng_ctx ctx[1])
{
    /* ensure the data in the context is
destroyed  */
    memset(ctx, 0, sizeof(prng_ctx));
}

#if defined(__cplusplus)
}
#endif
```

Password verification value
Nilai dari dua byte ini dihasilkan sebagai bagian dari proses yang menghasilkan kunci enkripsi dan dekripsi dari kata kunci (password). Ketika melakukan enkripsi, nilai verifikasi (verification value) diperoleh dari kata kunci enkripsi (encryption password) dan disimpan bersama berkas yang terenkripsi. Sebelum melakukan dekripsi, nilai verifikasi (verification value) dapat dihasilkan dari kata kunci dekripsi (decryption password) dan kemudian dibandingkan dengan nilai verifikasi yang tersimpan pada berkas terenkripsi yang dihasilkan pada proses enkripsi. Nilai verifikasi ini akan mendeteksi kata kunci yang salah (kata kunci dekripsi tidak sama dengan enkripsi) dengan cepat tetapi tidak untuk semua kata kunci yang salah, hanya pendeteksian dini saja. Ada kemungkinan 1 dari 65536 bahwa kata kunci yang salah tetap menghasilkan nilai verifikasi yang sama. Oleh karena itu nilai verifikasi yang cocok tidak dijadikan dasar untuk mengindikasikan kata kunci yang benar. Nilai verifikasi ini disimpan tanpa terenkripsi.
Information on how to obtain the password verification value from Dr. Gladman's encryption library can be found on the coding tips page.

Encrypted file data
Enkripsi diterapkan hanya pada isi dari berkas. Dilakukan setelah proses kompresi dan tidak pada data lain yang berasosiasi. Berkas data dienkripsi byte demi byte menggunakan algoritma enkripsi AES yang dioperasikan pada mode "CTR", ini berarti panjang data yang terenkripsi sama dengan sebelum enkripsi.
Walaupun data yang dienkripsi adalah byte demi byte tapi pada funsi enkripsi dan dekripsi dikenal dalam bentuk blok. Besar ukuran blok yang digunakan untuk enkripsi dan dekripsi harus sama. Berdasarkan spesifikasi AE-2, ukuran blok harus 16 byte (walaupun byte terakhir mungkin akan lebih kecil).

Authentication code
Kode otentikasi ini menyediakan pengecekan pada isi berkas yang terenkripsi tidak berubah baik dengan sengaja atau tidak sengaja sejak pertama kali dienkripsi. Jadi kode otentikasi ini bekerja seperti CRC yang mengecek keutuhan data setelah berkas dikompres dan dienkripsi. Seperti telah disebutkan sebelumnya, CRC pada AE-2 tidak lagi digunakan karena rentan terhadap berbagai serangan sehingga digantikan dengan kode otentikasi.
Kode otentikasi dihasilkan dari keluaran proses enkripsi. Kode AES Dr. Gladman menyediakan servis ini. Kode otentikasi tersebut tidak dienkripsi dan disimpan mengikuti byte terakhir data yang dienkripsi.

Non-files and zero-length files
Untuk mengurangi ukuran berkas zip, WinZip Inc. merekomendasikan bahwa selain berkas seperti direktori atau folder tidak perlu dienkripsi. Tetapi berkas dengan ukuran 0 harus dienkripsi, karena jika terdapat berkas yang dienkripsi dan tidak dalam satu berkas zip akan berbahaya dan mumungkinkan penyerang untuk membongkar enkripsi, hal ini akan dijelaskan kemudian.
Jika berkas dengan panjang 0 dienkripsi, bagian "Encrypted file data" pada penyimpanan akan bernilai 0 (lihat tabel 2), tetapi tamabahan data lainnya seperti "salt value", "password verification", dan "authentication code" harus tetap diisi baik pada area penyimpanan berkas maupun pada header lokal dan pusat.

"Mixed" Zip files”
Tidak ada aturan untuk mengenkripsi semua berkas yang berada dalam satu berkas zip ataupun mengenkripsi menggunakan kata kunci yang sama pada berkas yang dienkripsi. Kita dapat hanya mengenkripsi berkas yang penting saja terenkripsi sedangkan berkas lainnya tetap tidak terenkripsi atau tiap berkas menggunakan kata kunci yang berbeda. Akan tetapi adanya berkas yang terenkripsi dan tidak terenkripsi pada satu berkas zip menimbulkan resiko keamanan, hal ini akan dibahas lebih jauh pada makalah ini.

Key Generation
bilangan acak-semu yang dihasilkan pada enkripsi AE-2, sebagaimana diimplementasikan oleh Dr. Gladman, diperoleh dari fungsi PBKDF2

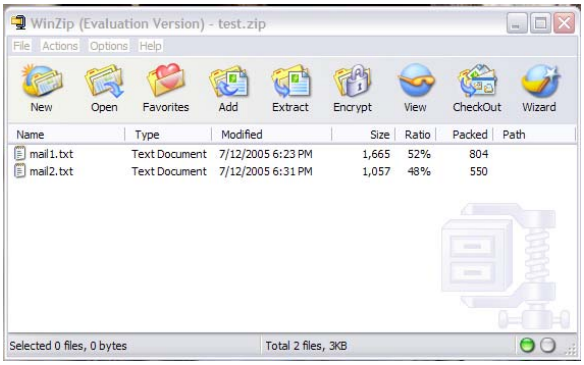
yang dideskripsikan pada RFC2898. fungsi PBKDF2(P, S, c, dkLen) menerima masukan sebagai berikut :

P : password (kata kunci) sebuah string oktet
S : nilai salt, juga sebuah string oktet
c : counter iterasi, sebuah bilangan integer positif
dkLen : panjang oktet string yang ingin dihasilkan, sebuah bilangan integer positif

Output dari fungsi ini adalah bilangan acak yang kemudian dibagi menjadi tiga bagian: n bit pertama dijadikan kunci enkripsi (encryption key), n bit selanjutnya dijadikan kunci otentifikasi (authentication key) dan 16 bit (2 byte) terakhir menjadi nilai verifikasi kata kunci (password verification value).

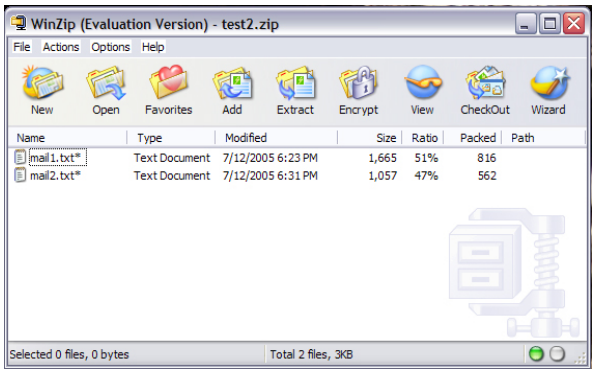
6. Enkripsi pada berkas yang akan diZIP

Untuk melakukan enkripsi pada berkas zip pertama kita buat arsip baru dengan memilih *new* kemudian tambahkan beberapa berkas yang akan di-zip. Pada gambar dibawah berkas zip terdiri dari mail1.txt dan mail2.txt. Berkas-berkas tersebut belum dienkripsi.



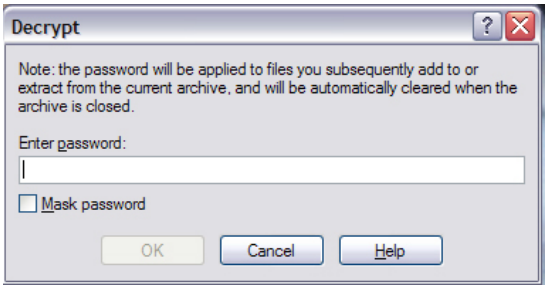
Untuk mengenkripsi semua berkas yang ditampilkan pada jendela WinZip pilih icon Encrypt (bergambar gembok) kemudian akan muncul *dialog box* Enccrypt seperti dibawah ini.

Isikan kata kunci kemudian pilih salah satu dari tiga metode enkripsi yang disediakan yaitu, Zip 2.0 compatible encryption (portable), 128-bits AES encryption (strong), dan 256-bits AES encryption (stronger). Zip 2.0 memang *compatible* dengan seri winzip sebelumnya (WinZip sebelum versi 9.0) dan program Zip lainnya tetapi memiliki kelemahan yaitu sangat rentan terhadap serangan. Sedangkan enkripsi AES merupakan enkripsi yang kuat.

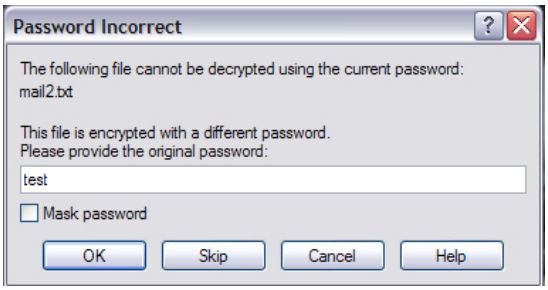


Perhatikan bahwa berkas yang dienkripsi akan diberi tanda asterisk (*) setelah nama berkas. Sebgai tambahan, WinZip tidak menenkripsi nama berkas, yand dienkripsi adalah hanya isi berkas saja.

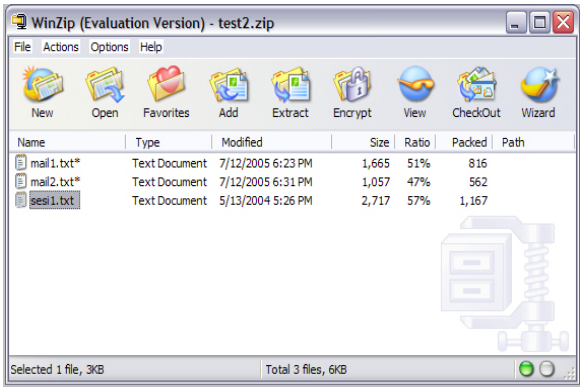
Jika kita akan mengekstraksi berkas yang telah dienkripsi, maka akan muncul *dialog box* Decrypt



Jika password yang dimasukan tidak sama dengan password ketika enkripsi maka akan munxul peringatan pada *dialog box* Decrypt seperti pada gambar dibawah ini.



Pada winzip versi 9.0 keatas dalam satu berkas zip dapat terdiri dari berkas terenkripsi dan tidak.



Gambar diatas memperlihatkan dua berkas terenkripsi yaitu mail1.txt, mail2.txt dan satu berkas sesi1.txt tidak dienkripsi pada satu berkas zip.

7. Serangan terhadap berkas ZIP

Walaupun WinZip telah dilengkapi dengan fitur enkripsi, akan tetapi masih terdapat beberapa celah untuk mendapatkan berkas plaintext yang berada pada berkas zip melalui beberapa serangan. Kita pernah mendengar program-program *password recovery* yang berda dipasaran, program tersebut berguna jika kita memproteksi berkas zip dengan *password* (kata kunci) kemudian ketika ingin membuka berkas-berkas yang berada berkas zip kita lupa *password* (kata kunci) yang digunakan. Dengan bantuan program tersebut kita dapat mendapatkan kembali berkas-berkas yang telah dikompresi dan dienkripsi menjadi berkas zip tanpa mengetahui *password* (kata kunci). Hal ini berarti keamanan enkripsi pada WinZip dapat dipecahkan, walaupun serangan menggunakan program *password recovery* belum tentu semuanya berhasil tergantung kekuatan kata kunci yang digunakan. Semakin panjang kata kunci maka semakin sulit atau hampir tidak mungkin untuk dipecahkan.

Brute Force Attack

Serangan BruteForce akan mencoba semua kemungkinan *password* kata kunci. Jika panjang kunci yang digunakan hanya lima karakter, dengan anggapan karakter yang digunakan biasanya adalah a..z A..Z 0..9 dan karakter lainnya seperti !@#$%^&*() maka kemungkinan kunci adalah 72^4 yaitu 26.873.856, kemungkinan kunci. Dengan kemampuan komputer sekarang hal *password* (kata kunci) akan mudah didapatkan.

Sebuah program *password recovery* yang di-*install* pada komputer dengan prosesor AMD Athlon 64 3000 (clock 1,8 GHz) dan memori sebesar 512 byte sanggup mencoba 12 juta kata kunci tiap detik untuk berkas zip yang dienkripsi dengan ZIP 2.0 encryption, sedangkan jika enkripsi yang digunakan AES 128-bit kecepatannya turun drastic menjadi hanya 600 *password* (kata kunci) per detik dan 300 *password* (kata kunci) per detik untuk AES 256-bit. Lihat table 3.

Enkripsi	Kecepatan(p/s)	Estimasi
ZIP 2.0	12 juta	3 detik
AES 128-bit	600	30 menit
AES 256-bit	300	60 menit

Akan tetapi jika kata kunci yang digunakan cukup panjang missal 10 karakter maka kemungkinan kata kunci menjad sangat besar yaitu 72^{10} = 3.743.906.242.624.487.424. membutuhkan waktu sekitar 10 tahun untuk mendapatkan kata kunci, itu pun jika dienkripsi menggunakan ZIP 2.0 yang sangat lemah jika dibandingkan AES.

Oleh karena itu serangn dengan *brute force* tidaklah efektif jika kata kunci tang digunakan

cukup panjang apalagi jika yang digunakan adalah enkripsi AES.

Dictionary Attack
Seperti serangan *brute force*, serangan *dictionary* mencoba kemungkinan kata kunci. Tetapi kemungkinan keta kunci yang digunakan lebih sedikit dari *bruteforce* karena semua kemungkinan kata kunci ada pada *dictionary*(kamus). Isi dari kamus itu sendiri adalah semua kemungkinan kata dalam bahasa tertentu, missal inggris. Serangan ini dilakukan karena ada biasanya orang menggunakan kata atau kombinasi kata pada bahasa tertentu agar kata kunci mudah diingat.

Table dibawah ini menunjukan waktu yang dibutuhkan untuk mendapatkan kata kunci menggunakan *Dictionary Attack*. Kata kunci yang digunakan adalah kata dalam bahasa inggris yaitu *mail*.

Enkripsi	Kecepatan(p/s)	Estimasi
ZIP 2.0	12 juta	1 detik
AES 128-bit	600	9 menit
AES 256-bit	300	17 menit

Kelemahan serangn ini adalah hanya bisa digunakan jika kata kunci yang digunakan tidak mengandung karakter selain huruf dan bukan merupakan *String* yang tidak bermakna artinya dalam bahasa tertentu. Selain itu bahasa yang digunakan untuk mengenkripsi harus sama dengan bahasa pada kamus yang digunakan. Dengan kata lain penyerang harus mengetahui bahasa yang digunakan

Known Plaintext Attack (KPA)
Known Plaintext Attack (KPA) adalah jenis serangan kriptanalis dimana penyerang memiliki contoh berkas baik itu plainteks dan juga chipernya yang berkoresponden (plainteks yang telah dienkripsi) setelah itu penyerang dapat bebas memperoleh informasi lebih jaun seperti kunci yang digunakan.
Berkas terkompresi (berkas zip) yang dienkripsi sangat lemah terhadap serangan seperti ini.

Arsip berkas zip yang ternkripsi sangat rentan terhadap serangan ini. Penyerang hanya membutuhkan satu berkas yang tidak terenkripsi untuk mendapatkan semua berkas. Program *password recovery* dapat dengan mudah mengkalkulasi kunci yang dibutuhkan untuk melakukan dekripsi. Untuk mendapatkan berkas yang tidak dienkripsi (plainteks) penyerang bisa mencarinya di internet atau dengan mencoba-coba mencari pada komputer korban. Bisa juga dengan membuat sendiri plainteks berdasarkan nama berkas yang dienkripsi karena walaupun isi dari berkas dienkripsi tetapi nama berkas tidak dienkripsi.

KPA pada stream cipher
Serangan Known Plainteks pertama kali dikemukakan oleh Eli Biham dan Paul C.Kocher [BK94] “Known Plaintext Attack on the PKZIP” Stream Cipher. kemudian oleh Mikel Stay “ZIP Atack with reduced Known-Plaintext”.

Sebelum dijelaskan bagaimana serangan ini bekerja terlebih dahulu dijelaskan keterangan symbol-simbol yang digunakan.

Bit 0	1	2	3	4	5	6	7

Tiap bit diberi nomor dari kanan ke kiri, bit ke-0 adalah bit pertama, bit ke-1 menempati tempat kedua, dan seterusnya.
Misalkan p_i adalah byte plainteks ke-i, i = 1,2,3,…,n. Misalkan s_i adalah byte stream ke-1 dan $key0_i$, $key1_i$, dan $key2_i$ adalah nilai dari *key0*, *key1*, dan *key2* setelah memproses p_i. Catatan bahwa S_1 adalah sebuah fungsi dari *header random* dan *password*. Bit 2 sampai 15 dari *key2* menentukan S_{i+1}.

Serangan dengan plainteks adalah sebagai berikut :

1. XOR *ciphertext* dan *plaintext* yang diketahui untuk mendapatkan byte stream s_1 sampai s_3
2. Tebak secara sembarang 22 bit dari $key2_{13}$.
3. Tebakan yang dikombinasikan dengan S_{13} sudah cukup untuk mengisi delapan atau lebih bit pada $key2_{13}$, total semuanya 30. S_{12} menyediakan informasi yang cukup untuk menurunkan 30 bit dari $key2_{12}$ dan nilai paling signifikan (MSB) dari $key1_{13}$. Secara umum, tiap byte stream memungkinkan kita untuk mengkalkulasi 30 bit $key2_{i-1}$ dan byte paling signifikan (MSB) dari $key1_i$.
4. Kemudian kita lanjutkan dengan menggunakan byte stream untuk membuat sebuah daftar dari byte paling signifikan (MSB) dari $key11_3$ sampai

key1$_8$.

5. Untuk setiap daftar, kita temukan 2^{16} kemungkinan untuk 24 bit rendah (*low bit*) dari key113 sampai key19 dengan mengkalkulasikan byte rendah (*low byte*) dari *(key1*$_i$*+LSB(key0*$_{i+1}$*))*, dengan begitu kita mendapatkan byte kanan tinggi (*right high byte*)dari *key1*$_{i+1}$.
6. Untuk setiap 2^{16} daftar dari key1 yang telah lengkap, turunkan byte rendah (*low byte*) dari *key0*$_{13}$ sampai *key0*$_{10}$.
7. Jika kita telah mendapatkan byte rendah (*low byte*) dari *key0*$_{10}$, *key0*$_{11}$, *key0*$_{12}$, and *key0*$_{13}$, Kita dapat menggunakan informasi dari byte pada plainteks untuk membalikan funsi CRC, karena fungsi tersebut linier, dan menemukan *internal state* secara lengkap pada satu titik selama enkripsi.
8. Jika *internal state* sudah ditemukan semuanya kita dapat mendekripsi *ciphertext* sesuai dengan p_1 sampai p_5 dan membuang semua kunci yang salah.

Kita perlu total 13 byte dari *plaintext* yang diketahui, delapan untuk serangan dan 5 untuk menyaring 2^{38} kunci yang ada.

Random Seed Attack

Seperti dijelaskan sebelumnya, skema enkripsi ZIP 2.0 sangat lemah. Salah satu serangannya adalah dengan *known-plaintext attack*. Disini akan dijelaskan cara lain untuk menyerang skema enkripsi tersebut dengan melakukan *reverse engineering* pada IBDL32.ddl-nya WinZip.

Sebagaimana kita ketahui mendapatkan plainteks yang valid untuk melakukan serangan *known plaintekx* [BK94] cukup sulit. Kesulitan mendapatkan plainteks karena plainteks tersebut dienkripsi dan pengubahan sedikit saja data terenkripsi tersebut akan menghasilkan plainteks yang kacau. Jadi tanpa mengetahui plainteks kita tidak dapat melakukan apa-apa.

Berikut ini adalah skema enkripsi ZIP 2.0

```
void update_keys(char p) {
        key0=crc32(key0, p);
        key1=key1 + (key0 & 0xff);
        key1=key1 * 134775813 + 1;
        key2=crc32(key2, key1>>24);
}

char decrypt_byte(char b) {
        unsigned short tmp;
        tmp=key2 | 2;
        return(tmp * (tmp^1)>>8);
}

        To inizializate the keys:

--
        key0=305419896;
        key1=591751049;
        key2=878082192;

        for(i=0 ; i<strlen(pass) ; i++) {
                update_keys(pass[i]);
        }
--

        To code a byte of data:

--
        tmp=byte^decrypt_byte(byte);
        update_keys(tmp);
--
```

Sebagai tambahan, 12 byte acak ditambahkan pada data terkompresi dengan tujuan mempersulit serangan. Keduabelas byte ini sangat penting untuk menginisialisasi kode stream. Karena jika kita mengetahui state dari kode stream kita dapat membalikannya sampai kita mendapat awal dari stream tersebut.

Untuk menghasilkan 12 byte acak kebanyakan progam pengarsipan menggunakan fungsi acak, tetapi ada juga yang membuat fungsi sendiri seperti halnya WinZip.

Jika kita melakukan *reverse engineering* terhadap fungsi random WinZip maka akan diperoleh kode sebagai berikut :

```
46c5a0     push    ebp
46c5a1     mov     ebp, esp
46c5a3     mov     eax, [IBDL32.dll:Seed]
46c5a8     mov     edx, eax
46c5aa     add     edx, edx
46c5ac     add     edx, eax
46c5ae     shl     edx, 2
46c5b1     add     edx, eax
46c5b3     mov     ecx, edx
46c5b5     shl     ecx, 4
46c5b8     add     ecx, eax
46c5ba     mov     edx, ecx
46c5bc     shl     edx, 8
46c5bf     sub     edx, eax
46c5c1     shl     edx, 2
46c5c4     lea     ecx, [eax+edx]
46c5c7     mov     [IBDL32.dll:Seed], ecx
46c5cd     mov     eax, [IBDL32.dll:Seed]
46c5c2     sar     eax, 10h
46c5c5     mov     ecx, eax
46c5c7     and     ch, 7fh
46c5ca     movzx   edx, cx
46c5cd     mov     eax, edx
46c5cf     ret
```

Kode diatas adalah dalam bahasa assembly dan terlihat sangat membingungkan. Akan tetapi setelah dianalisis dapat dihasilkan kode dalam bahasa C yang isinya seperti berikut ini :

```
unsigned short rand()
{
        seed=0x343fD * seed;
        return ((seed >> 16)&0x7fff);
}
```

Sedangkan implementasi funsi *random* yang normal adalah sebagai berikut :

```
unsigned short rand()
{
        seed=0x343fD * seed + 0x269ec3;
        return ((seed >> 16)&0x7fff);
}
```

Sepertinya orang yang membuat kode *rand()* pada WinZip lupa menambahkan 0x269ec3 pada seed. Hal ini menyebabkan masalah keamanan karena kemungkinan rangkaian seed menurun dari 2^(12*8) menjadi 2^(3*8). Sekarang akan diperlihatkan bagaimana penyerangan dapat dilakukan.

Pengurangan rangkaian seed membuat penyerangan denagan *burteforce* akan lebih mudah dan kemudian menebak state dari kode stream (*key0*, *key1*, *key2*).

Kita dapat kembali pada state awal menggunakan formula yang diambil dari kode sumber *pkcrack* (stage2.c baris 175)

```
/* The equation from section 3.3 is used twice here:
* (1) keyl_{n-1} + LSB(key0_n) = rhs = (keyl_n - 1) * INVCONST
* and
* (2) keyl_{n-2} + LSB(key0_{n-1}) = (keyl_{n-1} - 1) * INVCONST
*
* At this point we know keyl_n, MSB(keyl_{n-1}) and MSB(keyl_{n-2}).
*
* From (2) follows
* MSB(keyl_{n-2}) = MSB((keyl_{n-1} - 1) * INVCONST - LSB(key0_{n-1}))
* Inserting (1) yields
* MSB(keyl_{n-2}) = MSB((rhs - 1)*INVCONST -
*                       LSB(key0_n)*INVCONST - LSB(key0_{n-1}))
* which means that either
* (a) MSB(keyl_{n-2}) = MSB((rhs - 1)*INVCONST) -
*                       MSB(LSB(key0_n)*INVCONST - LSB(key0_{n-1}))
* or
* (b) MSB(keyl_{n-2}) = MSB((rhs - 1)*INVCONST) -
*                       MSB(LSB(key0_n)*INVCONST - LSB(key0_{n-1})) - 1
*
* It can easily be verified that for any two bytes b1, b2:
* MSB( b1*INVCONST + b2 ) = MSB( b1*INVCONST )
* (simple exhaustive test on 2^16 combinations)
*
* We have computed diff = MSB((rhs - 1)*INVCONST) - MSB(keyl_{n-2}).
* Now all we have to do is find values for key0_n so that
* (following from (1))
* MSB(keyl_{n-1}) = MSB(rhs-LSB(key0_n))
* and (following from (a) and (b)) either
* diff = MSB(LSB(key0_n)*INVCONST)
* or
* diff = MSB(LSB(key0_n)*INVCONST) + 1
*
* Candidate values are selected using the precomputed lookup table mTab2.
*/
```

Percobaan untuk eksploitasi ini dilakukan oleh Mike Stevens *mikestevens@softhome.net* dan Elisa Flanders *eflanders@springfield.com* pada berkas ZIP yang terdiri dari 3 berkas, jumlah byte plaintext yang dikerahui adalah 12 * 3 = 36 byte dengan waktu kuran dari dua jam pada komputer Pentium 500MHz dan 128MB memori.

Serangan pada berkas ZIP yang terdiri dari 3 berkas atau kurang masih mungkin dapat dilakukan karena pada WinZip, dua byte CRC paling penting disimpan dengan 12 angka acak.

Percobaan dilakukan dengan terhadap berkas zip *tocrack.zip* yang terdiri dari tiga berkas dan kata kunci yang digunakan untuk enkripsi adalah “&THPOL101%ISLAME@|1” yang terdiri dari 19 digit. Teknik *bruteforce* tidak mungkin dapat dilakukan untuk mendapatkan kata kunci karena akan memakan waktu yang sangat lama. Akan tetapi yang dilakukan adalah melakukan *bruteforce* pada 3 kunci yang dihasilkan dari *password*.

```
[root@localhost]$ ./zipproof -p tocrack.zip
[*] generating posible secuences..  DONE.
[*] reducing number of posible Key2.. DONE.
[*] Bruteforcing:
        [-] Key2 => 0x6a54f21e
[*] Generating initial keys:
        [-] Key0 => 0xaca8571c
        [-] Key1 => 0x439e8759
        [-] Key2 => 0x508d8f22
[*] Tryng to get password.. Not found!
[E] Is not posible to find a password for these keys, you can use
    findkeys tool (from pkcrack) to get it.
[root@localhost]$
```

Mendapatkan password yang digunakan tidak mungkin dilakukan karena terlalu panjang, tetapi dengan 3 kunci yang didaptkan sudah cukum untuk melakukan ekstraksi berkas-berkas yang ada dalam berkas zip.

```
[root@localhost]$ ./zipdecrypt 0xaca8571c 0x439e8759 0x508d8f22 tocrack2.zip
Decrypting file1.txt (bb6c9531638e70d425ebe60b)... OK!
Decrypting file2.txt (0f57542dbf0e18517a5cee0b)... OK!
Decrypting file3.txt (2d2bc008f19607809298f90b)... OK!
```

10. Kesimpulan

Sebenarnya masih banyak celah keamanan lainnya pada skema enkripsi WinZip Tadayoshi Kohno dalam "*Attacking and Repairing the WinZip Encryption Scheme*" mejelaskan berbagai macam serangan terhadap skema winzip seperti kemungkinan adanya *man-in –the-middle attack* saat pengiriman berkas zip.

Setiap enkripsi yang digunakan tidak ada yang benar-banar aman 100% dari serangan, yang dapat dilakukan adalah memperlambat proses untuk mendapatkan plaintext dari ciphertextnya.

Skema enkripsi WinZip cukup rentan terhadap serangan oleh karena itu diharapakan menggunakan program enkripsi dari pihak ketiga jika ingin melakukan enkripsi pada berkas zip.

Referensi

[1] Advance Archive Password Recovery : Guaranteed WinZip attack, http://www.elcomsoft.com/help/archpr/guaranteed_winzip_attack.html

[2] Advance Archive Password Recovery : Introduction, http://www.elcomsoft.com/help/archpr/introduction.html

[3] Advance Archive Password Recovery : Known plaintext attack (ZIP), http://www.elcomsoft.com/help/archpr/known_plaintext_attack_(zip).html

[4] Advance Archive Password Recovery : Passwords and encryption, http://www.elcomsoft.com/help/archpr/password_encrypted_file.html

[5] project page, Dr. Gladman's web site http://www.winzip.com/gladman.cgi

[6] Biham, Eli. and Paul Kocher. "A Known Plaintext Attack on the PKZIP Stream Cipher." Fast Software Encryption 2, Proceedings of the Leuven Workshop, LNCS 1008, December 1994.

[7] Biham, Eli. How to decrypt or even substitute DES-encrypted messages in 228 steps.

[8] Kocher, Paul. ZIPCRACK 2.00 Documentation. 1992. http://www.bokler.com/bokler/zipcrack.txt

[9] M. Stay. ZIP attacks with reduced known plaintext. In M. Matsui, editor, Fast Software Encryption 2001, volume 2355 of Lecture Notes in Computer Science

[10] PKWARE. APPNOTE.TXT - .ZIP File Format Specification, http://www.pkware.com/products/enterprise/white_papers/appnote.txt.

[11] PKWARE, Homepage, http://www.pkware.com/

[12] PKWARE, Inc., General Format of a ZIP File, technical note, included in PKZIP (file appnote.txt)

[13] WinZip Computing, Inc. AES encryption information:Encryption specification AE-2, http://www.winzip.com/aes_info.htm

[14] WinZip Computing, Inc. Advanced Encryption Technology

[15] WinZip Computing, Inc. AES Coding Tips for Developers, available at http://www.winzip.com/aes_tips.htm.

[16] WinZip Computing, Inc. Homepage,Available at http://www.winzip.com

[17] WinZip Computing, Inc. What's new in WinZip 1.0 http://www.winzip.com/whatsnew10.htm