# PKZIP

# PKZIP

- Phil Katz's ZIP program
- Katz invented zip file format
  - ca 1989
- Before that, Katz created PKARC utility
  - ARC compression was patented by SEA, Inc.
  - SEA successfully sued Katz
- Katz then invented zip
  - ZIP was much better than SEA's ARC
  - He started his own company, PKWare
- Katz died of alcohol abuse at age 37 in 2000

# PKZIP

- **PKZIP** compresses files using zip
- Optionally, it encrypts compressed file
  - Uses a homemade stream cipher
  - PKZIP cipher due to Roger Schlafly
  - Schlafly has PhD in math (Berkeley, 1980)
- PKZIP cipher is susceptible to attack
  - Attack is nontrivial, has significant work factor, lots of memory required, etc.

# PKZIP Cipher

- Generates 1 byte of keystream per step
- 96 bit internal state
  - State: 32-bit words, which we label X,Y,Z
  - Initial state derived from a password
- Of course, password guessing is possible
  - We do not consider password guessing here
- Cipher design seems somewhat ad hoc
  - No clear design principles
  - Uses shifts, arithmetic operations, CRC, etc.

# PKZIP Encryption

- Given
  - Current state: X, Y, Z (32-bit words)
  - p = byte of plaintext to encrypt
  - Note: upper case for 32-bit words, lower case bytes
- Then the algorithm is…

  k = getKeystreamByte(Z)

  $c = p \oplus k$

  update(X, Y, Z, p)
- Next, we define getKeystreamByte, update

# PKZIP getKeystreamByte

- Let "v" be binary OR
- Define $\langle X \rangle_{i\ldots j}$ as bits i thru j (inclusive) of X
  - As usual, bits numbered left-to-right from 0
- Shift X by n bits to right: X >> n
- Then…

getKeystreamByte(Z)

$$t = \langle Z \vee 3 \rangle_{16\ldots31}$$

$$k = \langle (t \cdot (t \oplus 1)) >> 8 \rangle_{24\ldots31}$$

return(k)

end getKeystreamByte

# PKZIP update

- Given current state X, Y, Z and p

  update(X, Y, Z, p)

  $X = \mathrm{CRC}(X, p)$

  $Y = (Y + \langle X \rangle_{24\ldots31}) \cdot 134775813 + 1 \pmod{2^{32}}$

  $Z = \mathrm{CRC}(Z, \langle Y \rangle_{0\ldots7})$

  end update
- CRC function defined on next slide

# PKZIP CRC

- Let X be 32-bit word, b a byte

```
CRC(X, b)
    X = X ⊕ b
    for i = 0 to 7
        if X is odd
            X = (X >> 1) ⊕ 0xedb88320
        else
            X = (X >> 1)
        end if
    next i
    return(X)
end CRC
```

# CRCTable and CRCinverse

- For efficiency, define CRCtable so that

  $$\mathrm{CRC}(X,b) = \langle X\rangle_{0\ldots23} \oplus \mathrm{CRCtable}[\langle X\rangle_{24\ldots31} \oplus b]$$

- Inverse table, CRCinverse, exists in the following sense:
- If $B = \langle A\rangle_{0\ldots23} \oplus \mathrm{CRCtable}[\langle A\rangle_{24\ldots31} \oplus b]$
- Then $A = (B \ll 8) \oplus \mathrm{CRCinverse}[\langle B\rangle_{0\ldots7}] \oplus b$
- Inverse table is useful in attack

# Lists

- Let $(X_i,Y_i,Z_i)$ be internal state used to generate $i^{th}$ keystream byte
- Let $k_i$ be the $i^{th}$ keystream byte
- Let $p_i$ be $i^{th}$ plaintext byte
- Define "X-list" to be $X_0,X_1,\ldots X_n$
  - Note that n+1 elements in this list
- Similar definition for k-list, p-list, etc.

# Outline of PKZIP Attack

- Assume k-list and p-list are known
  - This is a **known plaintext** attack
- Want to find state $(X_i,Y_i,Z_i)$ for some i
  - Then all keystream bytes are known
- Executive summary of the attack
  1. Use k-list to find a set of **Z-lists**
  2. For each Z-list, find multiple **Y-lists**
  3. For each Y-list, use p-list to obtain one **X-list**
  4. True X-list is among X-lists in 3. Find X-list using p-list. From X-list, obtain state and keystream
- Details of steps 1 thru 4 on following slides

# Step 1: Z-lists

- Assume keystream bytes $k_0, k_1, \ldots, k_n$ known
- Keystream byte $k_i$ computed as

  $k_i = \langle (t \cdot (t \oplus 1)) >> 8 \rangle_{24\ldots31}$

  Where $t = \langle Z_i \vee 3 \rangle_{16\ldots31}$
- Given $k_n$, there are 64 possible $t$
  - Due to the "$\vee$ 3"
- This gives 64 putative $\langle Z_n \rangle_{16\ldots29}$
- Similarly, we find 64 putative $\langle Z_{n-1} \rangle_{16\ldots29}$

# Step 1: Z-lists

- Have 64 putative $\langle Z_n \rangle_{16\ldots29}$ and $\langle Z_{n-1} \rangle_{16\ldots29}$
- Implies there are $2^{22}$ putative $\langle Z_n \rangle_{0\ldots29}$
- By update we have $Z_n = \mathrm{CRC}(Z_{n-1}, \langle Y \rangle_{0\ldots7})$
- By CRC inversion formula

$$Z_{n-1} = (Z_n \ll 8) \oplus \mathrm{CRCinverse}[\langle Z_n \rangle_{0\ldots7}] \oplus \langle Y_n \rangle_{0\ldots7}$$

- For each of $2^{22}$ putative $\langle Z_n \rangle_{0\ldots29}$
  - Know bits 0 thru 21 on RHS, bits 16 to 29 on LHS
  - For correct $Z_n$ and $Z_{n-1}$, bits 16 thru 21 must agree
  - Since 6 bits, 1/64 chance of a random match
  - Since 64 $Z_{n-1}$, for each $Z_n$ expect 1 matching $Z_{n-1}$
  - Since there are $2^{22}$ $Z_{n-1}$ we obtain $2^{22}$ $Z_{n-1}$

# Step 1: Z-lists

- Repeat for $\langle Z_{n-2} \rangle_{0\ldots29}$ then $\langle Z_{n-3} \rangle_{0\ldots29}$ etc.
- **Bottom Line**
  - We obtain about $2^{22}$ Z-lists
  - Each of the form $\langle Z_i \rangle_{0\ldots29}$, for $i = 1,2,\ldots,n$
- Possible to extend each of these to "full" $Z_i$
  - That is, $Z_i$ bits 0 thru 31, not just bits 0 thru 29
  - We omit details here (see text)
- We have $2^{22}$ Z-lists, $\langle Z_i \rangle_{0\ldots29}$, for $i = 1,2,\ldots,n$

# Step 1 Refinement

- Possible to reduce number of Z-lists
- Requires additional known plaintext
- Reduces overall work factor
- For example
  - 28 more bytes, we can reduce number of Z-lists (and overall work) by a factor of $2^4$
  - 1000 additional bytes can reduce number of lists to a range by $2^{11}$ to $2^{14}$
- We ignore refinement, so $2^{22}$ Z-lists

# Step 2: Y-lists

- We have about $2^{22}$ putative Z-lists
  - Each consisting of putative $Z_1, Z_2, \ldots, Z_n$
- We use these to find consistent Y-lists
- From update, we can write CRC inverse as

$$\langle Y_i \rangle_{0\ldots7} = Z_{i-1} \oplus (Z_i << 8) \oplus \text{CRCinverse}[\langle Z_i \rangle_{0\ldots7}]$$

- For each Z-list, have $\langle Y_2 \rangle_{0\ldots7}, \langle Y_3 \rangle_{0\ldots7}, \ldots, \langle Y_n \rangle_{0\ldots7}$
- How to find remaining 24 bits of each $Y_i$ ?
  - This is a bit tricky…

# Step 2: Y-lists

- From update we have
  $Y_i = (Y_{i-1} + \langle X_i \rangle_{24\ldots31}) \cdot 134775813 + 1 \pmod{2^{32}}$
- Rewrite this as
  $(Y_i - 1) \cdot C = Y_{i-1} + \langle X_i \rangle_{24\ldots31}$
- Where $C = 134775813^{-1} \pmod{2^{32}}$
- Then with very high probability
  $\langle (Y_i - 1) \cdot C \rangle_{0\ldots7} = \langle Y_{i-1} \rangle_{0\ldots7}$
- Letting $i = n$, we have
  $\langle (Y_n - 1) \cdot C \rangle_{0\ldots7} = \langle Y_{n-1} \rangle_{0\ldots7}$

# Step 2: Y-lists

- We have $\langle(Y_n - 1) \cdot C\rangle_{0\ldots7} = \langle Y_{n-1}\rangle_{0\ldots7}$
  - Where both $\langle Y_n\rangle_{0\ldots7}$ and $\langle Y_{n-1}\rangle_{0\ldots7}$ known
- Test all $2^{24}$ choices for $\langle Y_n\rangle_{8\ldots31}$
  - For each, compute $\langle(Y_n - 1) \cdot C\rangle_{0\ldots7}$
  - And compare to known $\langle Y_{n-1}\rangle_{0\ldots7}$
  - Probability of a match is $1/2^8$
- **Bottom line**: Obtain $2^{16}$ $Y_n$ per Z-list
- Since $2^{22}$ Z-lists, we have $2^{38}$ $Y_n$

# Step 2: Y-lists

- We have

$$(Y_n - 1) \cdot C = Y_{n-1} + \langle X_n \rangle_{24\ldots31}$$

- Rewrite as

$$Y_{n-1} = (Y_n - 1) \cdot C - \langle X_n \rangle_{24\ldots31}$$

- Let $a = \langle X_n \rangle_{24\ldots31}$
- Then

$$Y_{n-1} = (Y_n - 1) \cdot C - a$$

- For some unknown byte a

# Step 2: Y-lists

- We have $Y_{n-1} = (Y_n - 1) \cdot C - a$
  - For some unknown byte a
- For each $Y_n$, compute $Y_{n-1}$ for all possible a
  - Test whether $\langle (Y_{n-1} - 1) \cdot C \rangle_{0\ldots7} = \langle Y_{n-2} \rangle_{0\ldots7}$
  - Recall that $\langle Y_{n-2} \rangle_{0\ldots7}$ is known
  - Try all 256 a, each has $1/2^8$ probability of match
  - Expect **one** $Y_{n-1}$ for each $Y_n$
- Can be made efficient using lookup tables
  - Given $\langle Y_{n-2} \rangle_{0\ldots7}$ lookup consistent $\langle Y_{n-1} \rangle_{0\ldots7}$

# Step 2: Y-lists

- Repeat for $Y_{n-2}, Y_{n-3}, \ldots, Y_3$
- **Bottom line**
  - Expect to obtain $2^{38}$ Y-lists
  - Each of the form $Y_3, Y_4, \ldots, Y_n$
- Remaining steps in the attack
  - Find X-lists (step 3)
  - Find correct X-list from set of X-lists (step 4)
  - Then some $(X_i, Y_i, Z_i)$ known and msg is broken!

# Step 3: X-lists

- We have about $2^{38}$ putative Y-lists
  - Each is of the form $Y_3, Y_4, \ldots, Y_n$
- How to find corresponding X-lists?
- Consider the formula

$$\langle X_i \rangle_{24\ldots31} = (Y_i - 1) \cdot C - Y_{i-1}$$

- Use this to obtain $\langle X_i \rangle_{24\ldots31}$ for $i = 4,5,\ldots,n$
- How to find remaining bits of each $X_i$ ?

# Step 3: X-lists

- From update function

  $X_i = \mathrm{CRC}(X_{i-1}, p_i)$
- Using CRC table,

  $X_i = \langle X_{i-1}\rangle_{0\ldots23} \oplus \mathrm{CRCtable}[\langle X_{i-1}\rangle_{24\ldots31} \oplus p_i]$
- Implications?
- If we know one complete $X_i$ and all $p_j$ then we can compute **all** (complete) $X_j$
  - CRC inverse allows us to find $X_{i-1}$ from $X_i$
- So how to find one complete $X_i$ ?

# Step 3: X-lists

- We know $\langle X_i \rangle_{24\ldots31}$ and $p_i$ for each i
- From update: $X_i = \langle X_{i-1} \rangle_{0\ldots23} \oplus \text{CRCtable}[\langle X_{i-1} \rangle_{24\ldots31} \oplus p_i]$
- This implies
  1. $\langle X_i \rangle_{0\ldots23} = X_{i+1} \oplus \text{CRCtable}[\langle X_i \rangle_{24\ldots31} \oplus p_{i+1}]$
  2. $\langle X_{i+1} \rangle_{0\ldots23} = X_{i+2} \oplus \text{CRCtable}[\langle X_{i+1} \rangle_{24\ldots31} \oplus p_{i+2}]$
  3. $\langle X_{i+2} \rangle_{0\ldots23} = X_{i+3} \oplus \text{CRCtable}[\langle X_{i+2} \rangle_{24\ldots31} \oplus p_{i+3}]$
- From $\langle X_{i+3} \rangle_{24\ldots31}$, $\langle X_{i+2} \rangle_{24\ldots31}$ and 3, get $\langle X_{i+2} \rangle_{16\ldots31}$
- From $\langle X_{i+2} \rangle_{16\ldots31}$, $\langle X_{i+1} \rangle_{24\ldots31}$ and 2, get $\langle X_{i+1} \rangle_{8\ldots31}$
- From $\langle X_{i+1} \rangle_{8\ldots31}$, $\langle X_i \rangle_{24\ldots31}$ and 1, get $X_i = \langle X_i \rangle_{0\ldots31}$

# Step 3: X-lists

- Using $X_i$ found on previous slide and

$$X_i = \langle X_{i-1} \rangle_{0\ldots23} \oplus \text{CRCtable}[\langle X_{i-1} \rangle_{24\ldots31} \oplus p_i]$$

- We can find the complete X-list
- Repeat this for each putative Y-list
  - Gives us about $2^{38}$ putative X-lists
- Correct X-list will (almost certainly) be among these $2^{38}$ X-lists
- How to select the "winning" X-list?

# Step 4: Correct X-lists

- How to select correct X-list?
- We can compute $\langle X_i \rangle_{24\ldots31}$ in two ways:

$$X_i = \langle X_{i-1} \rangle_{0\ldots23} \oplus \text{CRCtable}[\langle X_{i-1} \rangle_{24\ldots31} \oplus p_i]$$

$$\langle X_i \rangle_{24\ldots31} = (Y_i - 1) \cdot C - Y_{i-1}$$

- These two results must agree!
- Since testing 1 byte
  - Probability of random match about $1/2^8$
- We have about $2^{38}$ putative X-lists, so…
- About 5 such comparisons and we're done!

# Step 4: Recover Keystream

- Once we have found correct X-list
  - We know corresponding Y-list, Z-list
- For some $i < n$, we know state $(X_i, Y_i, Z_i)$
- From $(X_i, Y_i, Z_i)$ we generate $k_j$ for $j \geq i$
- We have the keystream and msg is broken
- Trudy wins again!

# How Much Plaintext?

- Trudy wants to minimize known plaintext
- Require plaintext bytes $p_0,p_1,\ldots,p_n$
  - So, how small can n be?
- Need $\langle X_i \rangle_{24\ldots31}$, $\langle X_{i+1} \rangle_{24\ldots31}$, $\langle X_{i+2} \rangle_{24\ldots31}$, $\langle X_{i+3} \rangle_{24\ldots31}$ to determine $X_i$
  - We can assume $i+3 = n$, so $n,n-1,n-2,n-3$ needed
- And we need five more $X_i$ to find true X-list
  - Can assume we use $i = n-4,n-5,n-6,n-7,n-8$
- **Cannot** use $X_i$, $i=0,1,2,3$, for any of the above
  - Since these are not found by the attack

# How Much Plaintext?

- Bottom line
  - Need 13 consecutive known plaintext bytes
  - Since 4 + 5 + 4 = 13 (from previous slide)
- Can reduce the work (step 1 refinement)
  - Requires more known plaintext
  - "Work" determined by number of lists
  - 28 additional known plaintext bytes reduces number of lists from $2^{38}$ to $2^{34}$
  - About 1000 additional plaintext bytes reduces number of lists to a range of $2^{27}$ to $2^{24}$

# Slightly Simplified Attack

- If we do **not** reduce number of lists (i.e., we do not implement step 1 refinement)
  - Then work is on order of $2^{38}$
- In this case, a simpler attack is possible
- "Simpler" means easier to program
  - We do not have to save large number of lists
  - Instead, we process each lists as generated

# Simplified Attack

- Suppose we have 13 known plaintexts
  - That is, $p_0, p_1, \ldots, p_{12}$
  - Then we know keystream bytes $k_0, k_1, \ldots, k_{12}$
- From step 1, we first do the following:

for i = 0 to 12

  Find all $\langle Z_i \rangle_{16\ldots29}$ consistent with $k_i$

next i

- Expect 64 $\langle Z_i \rangle_{16\ldots29}$ for each i
- Remainder of attack is on the next slide

# Simplified Attack

for each $\langle Z_{12} \rangle_{16\ldots29}$ // expect 64
    for each $\langle Z_{12} \rangle_{0\ldots15}$ // $2^{16}$ choices
        for i = 11,10,…,0
            Find $\langle Z_i \rangle_{16\ldots29}$ consistent with $\langle Z_{i+1} \rangle_{0\ldots29}$
            Extend $\langle Z_i \rangle_{16\ldots29}$ to $\langle Z_i \rangle_{0\ldots29}$
        next i
        Complete to Z-list (solve for bits 30 and 31)
        Solve for $\langle Y_i \rangle_{0\ldots7}$
        Solve for all bits of Y-lists // expect $2^{16}$ lists
        for each Y-list
            Solve for $\langle X_i \rangle_{24\ldots31}$
            Solve for $X_9$ using $\langle X_9 \rangle_{24\ldots31}, \langle X_{10} \rangle_{24\ldots31}, \langle X_{11} \rangle_{24\ldots31}, \langle X_{12} \rangle_{24\ldots31}$
            Solve for X-list
            if $\langle X_i \rangle_{24\ldots31}$ verified for i=8,7,6,5,4, return (X,Y,Z)-list
        next Y-list
    next $\langle Z_{12} \rangle_{0\ldots15}$
next $\langle Z_{12} \rangle_{16\ldots29}$

# PKZIP Conclusions

- PKZIP cipher is somewhat complex and difficult to analyze
- PKZIP cipher design
  - Appears to be ad hoc
  - Violated Kerckhoffs Principle
  - Mixed-mode arithmetic is interesting
- The bottom line…
  - PKZIP cipher is insecure
  - But **not** trivial to attack
  - An interesting and unusual cipher!