Basic Problem

Given: both parties already know the same secret
Goal: send a message confidentially

Any communication system that aims to guarantee confidentiality must solve this problem
One-Time Pad

**Key** is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
\[ \text{ciphertext} = \text{plaintext} \oplus \text{key} \]

Decrypt by bitwise XOR of ciphertext and key:
\[ \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \]

Cipher achieves **perfect secrecy** if and only if there are as many possible keys as possible plaintexts, and every key is equally likely (Claude Shannon)

Advantages of One-Time Pad

- Easy to compute
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute

- As secure as theoretically possible
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker’s computational resources
  - ...as long as the key sequence is truly random
    - True randomness is expensive to obtain in large quantities
  - ...as long as each key is same length as plaintext
    - But how does the sender communicate the key to receiver?
Problems with One-Time Pad

- Key must be as long as plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- Does not guarantee integrity
  - One-time pad only guarantees confidentiality
  - Attacker cannot recover plaintext, but can easily change it to something else
- Insecure if keys are reused
  - Attacker can obtain XOR of plaintexts

Stream Ciphers

- One-time pad
  \[ \text{Ciphertext}(\text{Key}, \text{Message}) = \text{Message} \oplus \text{Key} \]
  - Key must be a random bit sequence as long as message
- Idea: replace “random” with “pseudo-random”
  - Encrypt with pseudo-random number generator (PRNG)
  - PRNG takes a short, truly random secret seed (key) and expands it into a long “random-looking” sequence
    - E.g., 128-bit key into a $10^6$-bit pseudo-random sequence
  \[ \text{Ciphertext}(\text{Key}, \text{Message}) = \text{Message} \oplus \text{PRNG}(\text{Key}) \]
  - Message processed bit by bit, not in blocks
Properties of Stream Ciphers

- Usually very fast
  - Used where speed is important: WiFi, SSL, DVD

- Unlike one-time pad, stream ciphers do **not** provide perfect secrecy
  - Only as secure as the underlying PRNG
  - If used properly, can be as secure as block ciphers

- PRNG must be **unpredictable**
  - Given the stream of PRNG output (but not the seed!), it’s hard to predict what the next bit will be
    - If PRNG(unknown seed) = \(b_1...b_i\), then \(b_{i+1}\) is “0” with probability \(\frac{1}{2}\), “1” with probability \(\frac{1}{2}\)

Weaknesses of Stream Ciphers

- No integrity
  - Associativity & commutativity: \((X \oplus Y) \oplus Z = (X \oplus Z) \oplus Y\)
  - \((M_1 \oplus \text{PRNG(key)}) \oplus M_2 = (M_1 \oplus M_2) \oplus \text{PRNG(key)}\)

- Known-plaintext attack is very dangerous if keystream is ever repeated
  - Self-cancellation property of XOR: \(X \oplus X = 0\)
  - \((M_1 \oplus \text{PRNG(key)}) \oplus (M_2 \oplus \text{PRNG(key)}) = M_1 \oplus M_2\)
  - If attacker knows \(M_1\), then easily recovers \(M_2\)
    - Most plaintexts contain enough redundancy that knowledge of \(M_1\) or \(M_2\) is not even necessary to recover both from \(M_1 \oplus M_2\)
Stream Cipher Terminology

- Seed of pseudo-random generator often consists of initialization vector (IV) and key
  - IV is usually sent with the ciphertext
  - The key is a secret known only to the sender and the recipient, not sent with the ciphertext

- The pseudo-random bit stream produced by PRNG(IV, key) is referred to as keystream

- Encrypt message by XORing with keystream
  - ciphertext = message \oplus keystream

RC4

- Designed by Ron Rivest for RSA in 1987
- Simple, fast, widely used
  - SSL/TLS for Web security, WEP for wireless

Byte array S[256] contains a permutation of numbers from 0 to 255

i = j := 0

loop

  i := (i+1) mod 256
  j := (j+S[i]) mod 256
  swap(S[i], S[j])
  output (S[i]+S[j]) mod 256

end loop
Content Scrambling System (CSS)

- DVD encryption scheme from Matsushita and Toshiba

Each DVD is encrypted with a disk-specific 40-bit DISK KEY

Each player has its own PLAYER KEY (409 player manufacturers, each has its player key)

KEY DATA BLOCK contains disk key encrypted with 409 different player keys:
- Encrypt DiskKey
- Encrypt PlayerKey1 (DiskKey) ...
- Encrypt PlayerKey409 (DiskKey)

This helps attackers verify his guess of disk key

What happens if even a single player key is compromised?

Generic LFSR

- The register is seeded with an initial value.
- At each clock tick, the feedback function is evaluated using the input from the tapped bits. The result is shifted into the leftmost bit of the register. The rightmost bit is shifted into the output.
Linear Feedback Shift Register

- Pseudo-random bit stream
- Linear Feedback Shift Register (LFSR)
  - The LFSR is one popular technique for generating a pseudo-random bit stream. After the LFSR is seeded with a value, it can be clocked to generate a stream of bits.
  - Unfortunately, LFSRs aren’t truly random – they are periodic and will eventually repeat.
  - In general, the larger the LFSR, the greater its period. There period also depends on the particular configuration of the LFSR.
  - If the initial value of an LFSR is 0, it will produce only 0’s, this is sometimes called null cycling
CSS: LFSR Addition

first 2 bytes of key + 4th bit 1

LFSR-17
8 ticks
1 byte
Optional bit-wise inverter
Output byte

last 3 bytes of key + 4th bit 1

LFSR-25
8 ticks
1 byte
Optional bit-wise inverter
carry-out

Weakness #1: LFSR Cipher

➢ Brainless: disk key is 40 bit long
  • $2^{40}$ isn’t really very big – just brainlessly brute-force the keys

➢ With 6 Output Bytes $O(2^{16})$:
  • Guess the initial state of LFSR-17 $O(2^{16})$.
  • Clock out 4 bytes.
  • Use those 4 bytes to determine the corresponding 4 bytes of output from LFSR-25.
  • Use the LFSR-25 output to determine LFSR-25’s state.
  • Clock out 2 bytes on both LFSRs.
  • Verify these two bytes. Celebrate or guess again.
CSS: LFSR Addition

- First 2 bytes of key + 4th bit 1
  - LFSR-17
  - 8 ticks
  - 1 byte
  - Optional bit-wise inverter

- Last 3 bytes of key + 4th bit 1
  - LFSR-25
  - 8 ticks
  - 1 byte
  - Optional bit-wise inverter

- Carry-out from prior addition

+ 8-bit add

Output byte

8 ticks

9/01/2010 CS 686
Weakness #1: LFSR Cipher (Cont.)

- With 5 Output Bytes $O(2^{17})$:
  - Guess the initial state of LFSR-17 $O(2^{16})$
  - Clock out 3 bytes
  - Determine the corresponding output bytes from LFSR-25
  - This reveals all but the highest-order bit of LFSR-25
  - Try both possibilities: $O(2)$
    - Clock back 3 bytes
    - Select the setting where bit 4 is 1 (remember this is the initial case).
    - It is possible that both satisfy this – try both.
  - Clock out 2 bytes on both LFSRs.
  - Verify these two bytes. Celebrate or guess again.
Weakness #2: Mangled Output

- With Known ciphertext and plaintext
  - Guess $L_k$ (1 byte)
  - Work backward and verify input byte
  - This is a $O(2^8)$ attack.
  - Repeat for all 5 bytes – this gives you the 5 bytes of known output for prior weakness.

CSS decryption algorithm
Attack on CSS Decryption Scheme

1. Knowing encrypted and decrypted title key, try 256 possibilities to recover 40 output bits of the LFSRs – this takes $O(2^{25})$
2. Guess 16 bits of the key contained in LFSR-17 – this takes $O(2^{16})$
3. Clock out 24 bits out of LFSR-17, use them to determine the corresponding output bits of LFSR-25 (this reveals all of LFSR-25 except the highest bit)
4. Clock back 24 bits, try both possibilities – this takes $O(2)$
5. Verify the key

This attack takes $O(2^{25})$ [due to Frank Stevenson]

Reading assignment

- **Aircrack-ng**