5 Stream Ciphers

5.1 Symmetric Cryptography

Symmetric Cryptography uses the same secret key $k$ for encryption and decryption.

**Symmetric Ciphers**

- Typically the same (reversible) algorithm is used for encryption and decryption.
- The algorithm(s) used may or may not be kept secret, but in either case, the strength of the algorithm should rest on the size of the keys and the design of the algorithm.
  - In theory, we can recover a symmetric key by performing an exhaustive search.
  - For a secure symmetric cipher, there should not be a more efficient algorithm for finding a key.
- There are numerous symmetric ciphers in use.
  - DES, triple-DES, IDEA, Skipjack, CAST, Blowfish, AES, ...
  - They have different key sizes (e.g., DES - 56 bits, IDEA - 128 bits, AES - 128, 192 or 256 bits).
  - Some are subject to patents in some countries.

**Symmetric Ciphers**

There are two types of symmetric cipher:

- **Block ciphers** that encrypt one block of data at a time.
  - Typically, blocks consist of 64 bits (8 bytes) or 128 bits (16 bytes) of data.
  - The same plaintext block always encrypts to the same ciphertext block.
In certain circumstances, this is a weakness and we need to use some sort of feedback to disguise patterns in the plaintext. This leads to different modes of operation.

- Stream ciphers that encrypt an arbitrary stream of data.
  - Depending on the cipher, the data may consist of a stream of bits or a stream of bytes.
  - Each plaintext bit (or byte) encrypts to a different cipher text bit (or byte) depending on what data occurred earlier in the stream.
  - It is possible to use a block cipher to implement a stream cipher.

Stream Ciphers
Basic idea: replace the random key in the one-time pad by a pseudo-random sequence, generated by a cryptographic pseudo-random generator (PRG) that is ‘seeded’ with the key.

5.2 Random Numbers
PRGs
PRG requirements:

- Randomness
  - Uniformity, scalability, consistency

- Unpredictability
  - Cannot determine what next bit will be despite knowledge of the algorithm and all previous bits.

- Unreproducible
  - Cannot be reliably reproduced.
Characteristics of the seed:
- Must be kept secret
- If known, adversary can determine output
- Must be random or pseudorandom number

**Linear Congruential Generator**
Common iterative technique using:

\[ X_{n+1} = (aX_n + c) \mod m \]

Given suitable values of parameters can produce a long random-like sequence

Suitable criteria to have are:
- Function generates a full period
- Generated sequence should appear random
- Efficient implementation with 32-bit arithmetic

Note that an attacker can reconstruct sequence given a small number of values - this can be made harder in practice.

**Blum Blum Shub Generator**
Find two large primes \( p, q \) congruent to 3 (mod 4) where \( m = p \times q \)

Seed: \( X_0 = k^2 \mod m \) (\( k \) relatively prime to \( m \))

Use least significant bit from iterative equation:

\[ X_{n+1} = X_n^2 \mod m \]

- **Unpredictable** given any run of bits
  - Passes next-bit test
- **Security** rests on difficulty of factoring \( m \)
- **Slow** since very large numbers must be used
  - Too slow for cipher use, but good for key generation

**Real Random Numbers**
In some cryptographic implementations, real random numbers are used.

Such numbers can be generated from random physical events.
- Measuring radioactive decay.
- Measuring the time to read blocks of data from a disk - this is affected by air turbulence and has a random component.
- Measuring the time between key strokes on a key board.
- Using the least significant bits of a computer’s clock.

Using real random numbers is not usually a practical option.

Geoff Hamilton
5.3 Stream Ciphers

Stream Ciphers

Finite state machine with output filter.
Keys determine state update ($k_1$), initial state ($k_2$) and output filter ($k_3$)

Thus we have $c_i = m_i \oplus k_i$, where:

- $m_i$ are the plaintext bits/bytes.
- $k_i$ are the keystream bits/bytes.
- $c_i$ are the ciphertext bits/bytes.

This means $m_i = c_i \oplus k_i$ and thus decryption is the same as encryption. Encryption/decryption can be very fast.
No error propagation: one error in ciphertext gives one error in plaintext.
Loss of synchronisation means decryption fails for remaining ciphertext.
No protection against message manipulation.
Same key used twice gives same keystream.

Stream Ciphers
For the stream cipher to be secure, the keystream must:

- Look random, i.e. pass pseudo-random tests.
- Be unpredictable, i.e. have a long period.
- Have large linear complexity (see textbooks).
- Have low correlation between key bits and keystream bits.

Furthermore, the keystream should be efficient to generate.
Most stream ciphers are based on a non-linear combination of LFSRs (Linear Feedback Shift Registers).

Geoff Hamilton
5.4 LFSRs

LFSRs
A feedback shift register:

It is desirable to use some form of non-linear function as the feedback function. However, this is often hard to do in practice. The feedback function can be expressed as a polynomial. For example, the following can be expressed as $X^3 + X + 1$ (mod 2):

LFSRs
An $n$-bit LFSR could cycle through $2^n - 1$ states before repeating. To obtain a maximal cycle length the feedback polynomial must be primitive. However, if the feedback polynomial is known, then given $n$ bits of the output, the entire state of the LFSR is known, and its output predictable. Even if the feedback polynomial is unknown, entire state of the LFSR can be determined from only $2n$ bits of output using the Berlekamp-Massey algorithm. The basic problem is that LFSRs are linear.

Combining LFSRs
In practice, a number of LFSRs are commonly used, and their outputs combined using a non-linear combining function:

A5/1
Probably the most famous of the recent LFSR based stream ciphers. Used to encrypt the on-air traffic in the GSM mobile phone networks in Europe and the US.

Geoff Hamilton
Makes use of three LFSRs of length 19, 22 and 23. The output of the cipher is the XOR of the three output bits of the three LFSRs. It is very quick in hardware. It is weak cryptography, as a simple exhaustive search breaks it.

- We can guess the state of the first two registers, and determine the third from the key-stream.

### 5.5 RC4

**RC4**

Invented by Ron Rivest

- “RC” is “Rons Code” or “Rivest Cipher”

Generate keystream one byte at each step

- Efficient in software
- Simple and elegant
- Diffie: RC4 is “too good to be true”

Used in lots of places: SSL, TLS, WEP, etc. Most popular stream cipher in existence.

#### RC4 Initialisation

Array $S$ indexed from 0 to 255 is initialised with the integers $0, \ldots, 255$, permuted in a key-dependent way.

Array $key$ contains $N$ bytes of key.

```plaintext
for i = 0 to 255
    S[i] = i
    K[i] = key[i (mod N)]
next i

j = 0
for i = 0 to 255
    j = (j + S[i] + K[i]) (mod 256)
    swap(S[i], S[j])
next i
```

#### RC4 Keystream

The output of the RC4 algorithm is a keystream of bytes which is XORed with the plaintext byte by byte. For each keystream byte, swap elements of array $S$ and select a byte from the array:

```plaintext
i = (i + 1) (mod 256)
j = (j + S[i]) (mod 256)
swap(S[i], S[j])
t = (S[i] + S[j]) (mod 256)
keystreamByte = S[t]
```

Use keystream bytes like a one-time pad: XOR to encrypt or decrypt

Geoff Hamilton
The WEP Protocol

WEP = Wired Equivalent Privacy
Very badly designed security protocol.
WEP uses RC4 for confidentiality.
- Considered a strong cipher.
- But WEP introduces a subtle flaw.
WEP uses CRC for integrity.
- Should have used a cryptographic hash instead.
- CRC is for error detection, not cryptographic integrity.

The WEP Protocol

WEP “integrity” does not provide integrity.
- CRC is linear, so is stream cipher XOR
  - Can change ciphertext and CRC so that checksum remains correct.
    - For example, we could change the destination IP address.
- Such introduced errors go undetected.
- This requires no knowledge of the plaintext.
- Even worse if plaintext is known.
- CRC does not provide a cryptographic integrity check.
  - CRC designed to detect random errors.
  - Not designed to detect intelligent changes.

The WEP Protocol

WEP uses a long-term secret 128-bit key $K$.
RC4 is a stream cipher, so each packet must be encrypted using a different key.
- 24-bit Initialization Vector ($IV$) sent with packet
  - Sent in the clear ($IV$ is not secret)
Actual RC4 key for packet is $(IV || K)$
The WEP Protocol
Each packet gets a new IV.
Long term key $K$ seldom changes
If long-term key $K$ and $IV$ are the same, then same keystream is used.

- This is bad.
- It is at least as bad as reuse of one-time pad.
- If $C_1 = RC4(IV||K) \oplus P_1$ and $C_2 = RC4(IV,K) \oplus P_2$ then:
  \[ C_1 \oplus C_2 = P_1 \oplus P_2 \]

The WEP Protocol
This attack would be prevented if a different $IV$ were used for every message encrypted using the same key.
However this was merely recommended in the standard and not required.
So many manufacturers simply reset it to 0 every time the card was powered up.
Some generated them randomly, but with only $2^{24}$ possible variations, a collision can be expected after just $2^{12}$ packets.
In practice it is common for the same $IV$ to be re-used with the same key.

The WEP Protocol
A simple linear checksum has the property:
\[ CRC(x \oplus y) = CRC(x) \oplus CRC(y) \]
Say the ciphertext $CT = RC4(IV||K) \oplus (m, CRC(m))$ The attacker creates $(m',CRC(m'))$ and XORs this with the ciphertext to get:
\[ CT' = CT \oplus (m',CRC(m')) \]
\[ = RC4(IV||K) \oplus (m,CRC(m)) \oplus (m',CRC(m')) \]
The attacker has changed the ciphertext in such a way that it will now decrypt to $m \oplus m'$, and the CRC will still be okay.
If the attacker wants to change the message to $m''$, then select $m'$ s.t. $m \oplus m' = m''$ i.e. $m' = m \oplus m''$.

The WEP Protocol

Geoff Hamilton
Consider the following message to eBay:

\[
\begin{array}{c}
m = \text{“Bid for 100$”, CRC(m)} \\
\text{⊕} \quad \text{RC4(IV || K)} \\
\text{IV, CT} \\
\text{⊕} \quad m’ \quad \text{CRC(m’)} \\
\text{IV, CT’}
\end{array}
\]

Select \( m’ \) such that \( m \oplus m’ = \text{“Bid for 900$”} \)

Geoff Hamilton