Stream Ciphers

Çetin Kaya Koç

http://cs.ucsb.edu/~koc
koc@cs.ucsb.edu

Gilbert Vernam

Gilbert Sandford Vernam was an AT&T Bell Labs engineer who, in 1917, invented an additive polyalphabetic stream cipher and later co-invented an automated one-time pad cipher. Wikipedia

Born: April 3, 1890

Died: February 7, 1960

Block Ciphers

- Plaintext: M_i with $|M_i| = n$, where n is the block length (in bits)
- Ciphertext: C_i with $|C_i| = m$, where $m \geq n$, however, generally output size is equal to intput size: $m = n$
- If $m < n$, there will be more than one ciphertext for a given plaintext — ambiguity in decryption
- If $m > n$, some ciphertexts will never appear
- Encryption and decryption functions:

$$E_k(M_i) = C_i \quad ; \quad D_k(C_i) = M_i$$

- Key size: $|K|$, the length of the key in bits

Stream Ciphers

- Plaintext: m_i with $|m_i| = k$, where k is the plaintext length (in bits), which is generally a small number: 1, 2, 4, 8, etc
- Ciphertext: c_i with $|c_i| = k$, in other words, $|m_i| = |c_i|$
- Running key: r_i with $|c_i| = k$, a sequence of symbols length k
- Plaintext, ciphertext, and running keys are from the same alphabet; for example, for $k = 4$ this would be $\{0000, 0001, \ldots, 1111\}$
- Encryption and decryption functions:

$$E(m_i) = c_i = m_i \oplus r_i \quad ; \quad D(c_i) = m_i = c_i \oplus^{-1} r_i$$

where $\oplus$ is the (appropriate) addition function

A Stream Cipher — à la Vigenère

- Plaintext, Ciphertext, Running Key Alphabet: $\{a, b, c, \ldots, z\}$ encoded as elements of $\mathcal{Z}_{26}$
- Given a plaintext message: $m_i \in \mathcal{Z}_{26}$ for $i = 1, 2, 3, \ldots$
- Given a sequence of running keys: $r_i \in \mathcal{Z}_{26}$ for $i = 1, 2, 3, \ldots$
- The ciphertext sequence is computed using the encryption function

$$c_i = m_i + r_i \pmod{26}$$

- Similarly, the plaintext is computed using the decryption function

$$m_i = c_i - r_i \pmod{26}$$

A Stream Cipher — à la Vigenère

- The encryption and decryption function are

$$\begin{aligned} c_i &= m_i \oplus r_i &\equiv& \quad m_i + r_i \pmod{26} \\ m_i &= c_i \oplus^{-1} r_i &\equiv& \quad c_i - r_i \pmod{26} \end{aligned}$$

- The sequence of running keys r_i needs to have certain properties in order for a stream cipher to be cryptographically strong
- For the classic Vigènere:
  - The running key sequence is repeating: `herbalistherbalistherbalistherbalistherbali`$\cdots$
  - The period is equal to the length of the key word, which is generally a small integer

Cryptanalyzing Stream Ciphers

- In order to understand what properties the running key sequence needs to have we need to see if the stream cipher can be cryptanalyzed under the usual attack scenarios: CO, KP, CP, CT
- Under the CO scenario, given the ciphertext sequence c_i, the purpose of the adversary is to guess or to compute:
  - A portion or all of the running key sequence r_i
  - A portion or all of the plaintext sequence m_i
- These actions produce equivalent results in the sense that:
  - If a portion of r_i is obtained, we compute m_i using $m_i = c_i \oplus^{-1} r_i$
  - If a portion of m_i is obtained, we compute r_i using $r_i = c_i \oplus^{-1} m_i$

Cryptanalyzing Stream Ciphers

- On the other hand, under the known or chosen text attack scenarios, the adversary obtains (or chooses) a portion of the plaintext sequence m_i
- This immediately implies that the adversary can compute a portion of the running key sequence r_i (which is of the same length as m_i) using

$$r_i = c_i \oplus^{-1} m_i$$

- In order to obtain longer portions of the plaintext, we cannot assume that the adversary will receive further known (or chosen) text
- At this stage, the adversary can try guess what the other (past or future) portions of the running key would be, given a portion of the running key

Properties of Running Key Sequences

- As we have said: the sequence of running keys r_i needs to have certain properties in order for a stream cipher to be cryptographically strong
- Considering the CO attack scenario: The running key sequence needs to have **uniformly distributed** or **statistically random** finite segments so that all segments appear with equal probability, and any segment of the sequence cannot be guessed with better probability than the probability of that segment appearing in the sequence — **Requirement R1**
- Considering the CT attack scenario: *Given any finite segment(s) of the running key sequence*, any past or future segments need to be **unpredictable** which means they cannot be computed or guessed with better probability than the probability of that segment appearing in the sequence — **Requirement R2**

Binary Stream Cipher

- For the rest of our discussions, we will consider the binary stream cipher in which the plaintext m_i, ciphertext c_i, and the running key r_i words are binary bits, $m_i, c_i, r_i \in \{0, 1\}$ — The plaintext, ciphertext, and running key sequences are binary bit streams
- The encryption and decryption functions are the same:

$$\begin{aligned} c_i &= m_i \oplus r_i = m_i + r_i \pmod 2 \\ m_i &= c_i \oplus r_i = c_i + r_i \pmod 2 \end{aligned}$$

- The operation $\oplus$ is the mod 2 addition, which is its own inverse

```
mi    0101 0010 1101 1001 0011
ri    0110 0101 0110 0110 0101
ci    0011 0111 1011 1111 0110
```

Running Key Sequence Generators

- A running key sequence generator needs to work in both sides of the channel, at the side of the sender and the receiver, and produce exactly the same sequence r_i in order for the stream cipher to function properly

  Sender: r_i is produced; $c_i = m_i \oplus r_i$ is computed; c_i is sent
  Receiver: c_i is received; the same r_i is produced; $m_i = c_i \oplus r_i$ is computed
- Therefore, we need to have a **deterministic state machine** producing the running key sequence
- Furthermore, in order for it to be computable, the state machine needs to be finite, i.e., it needs to have a finite number of states (memory)
- Therefore: A stream cipher running key generator is a deterministic finite state machine whose sequences r_i satisfy Requirements R1 and R2

Random Number Generators (RNGs)

- A random number generator (RNG) produces a sequence of random (or random-looking) numbers in a predetermined range, such as $r_i \in \{0, 1\}$ or $r_i \in [0, 1]$
- Random (or random-looking) numbers have many applications: statistical physics, simulation, industrial testing and labeling, games, gambling, Monte Carlo methods, and cryptography
- True random numbers cannot be computed on deterministic computers.
- True random numbers are best produced using physical random number generators which operate by measuring a well controlled and specially prepared random physical process
- An information-theoretic provable RNG seems to be possible only by exploiting randomness inherent to certain quantum systems

Random Number Generators (RNGs)

- There are two basic categories of RNGs: True RNGs (TRNGs) and Deterministic RNGs (DRNGs)
- TRNGs are produced using physical or quantum processes; physical processes include free running oscillators, electrical noise from a resistor or semiconductor, and decay times from a radio-active material
- We cannot use TRNGs as stream ciphers, except for the special case of the Vernam cipher, called the **one-time pad**
- In order to understand the properties of the one-time pad, we need to define perfect secrecy, a concept introduced by Claude Elwood Shannon, an American mathematician, electronic engineer, and cryptographer known as "The father of Information Theory" — however, we will study perfect secrecy after we study block ciphers

Stream Ciphers and DRNGs

- Our purpose is to build and understand the properties of stream ciphers
- DRNGs are finite state machines that have a fixed but large number of starting conditions and states, and thus, very long periods
- Having long periods is an essential quality for stream ciphers; repeated sequences of running keys will yield information about the plaintext
- In addition to long period, we also would like the have DRNGs that satisfy Requirement R1 (uniform distribution or statistical randomness) and Requirement R2 (unpredictability)
- In this course, we will limit our attention to DRNGs, and study linear congruential generators, linear feedback shift registers, and cellular automata