Cryptography III: Stream Ciphers

Noah Singer

Montgomery Blair High School Cybersecurity Team

November 11, 2016

Overview

One-time pad

This will provide a good segue into modern cryptography from classical cryptography.
Let the key k be a string the same length as the message m.
We encrypt by XORing k with m, and decrypt a ciphertext c by XORing with the key again.

$E(m, k) := m \oplus k$
$D(c, k) := c \oplus k$

We can only use the key k once before discarding it.

Properties of the one-time pad

- It possesses **perfect security**.
  - Knowing the ciphertext reveals exactly *zero* knowledge about the key or plaintext, in the information-theoretic sense.
  - True even for an adversary with infinite computational power and time.
  - We'll make this much more rigorous later.
- *The key can only be used once and must be random.*
- Equally hard to exchange secure keys as it is to exchange secure messages, since they're the same length.
  - Can still agree on the key beforehand so is still useful in some cases.
  - Turns out that schemes with perfect secrecy must have this problem (keyspace is bigger than message space) due to Shannon's "Bad-News" Theorem.

Two-time pad

Proposition

There exists a ciphertext-only attack to get a total break on the one-time pad, given that we have at least *two* ciphertexts and they correspond to English text. We call this **crib dragging**.

The fundamental issue with the **two-time pad** is thus: given two messages m_1 and m_2 encrypted with the same key k, we have that $c_1 = E(m_1, k)$ and $c_2 = E(m_2, k)$, so $c_1 \oplus c_2 = E(m_1, k) \oplus E(m_2, k) = (m_1 \oplus k) \oplus (m_2 \oplus k) = (m_1 \oplus m_2) \oplus (k \oplus k) = (m_1 \oplus m_2) \oplus 0 = m_1 \oplus m_2$. The value $m_1 \oplus m_2$ is an example of something encrypted with a **running key cipher**. Since m_1 and m_2 are not totally random data but rather English text, we can recover both of them.

Crib dragging

1. XOR the two ciphertexts together, which cancels out the key, leaving us with an XOR of plaintexts, $m_1 \oplus m_2$.
2. Guess a string which is probably in one of the plaintexts. Some good examples are `flag{` and `the`.
3. XOR this string at every position in the XORed plaintext string.
4. If the result, in that position, looks like reasonable English text, then the string might be at that position, since if the string is actually in one of the plaintexts, XORing with the string cancels that plaintext and leaves the other.
5. Repeat steps 2-4, and make a better guess about the new string based on what we've already found out.

Stream ciphers

Major issue with one-time pad: we don't want to carry around a key that's as long as our message all the time.

Stream ciphers

Major issue with one-time pad: we don't want to carry around a key that's as long as our message all the time.

Solution: Start out with a (comparatively) short key, and then derive as much random-*looking* data from it, using it as a **seed**, and XOR our message with it to get ciphertext as in the one-time pad. We need to use a **cryptographically secure pseudorandom number generator (CSPRNG)**. This random data is called the **keystream**, and the resulting cipher a **stream cipher**.

Bit-flipping attack

Proposition

There exists a known-plaintext attack allowing the attack to generate arbitrary ciphertexts corresponding to any plaintext, given the known-plaintext.

Bit-flipping attack

Proposition

There exists a known-plaintext attack allowing the attack to generate arbitrary ciphertexts corresponding to any plaintext, given the known-plaintext.

Given a ciphertext c_1 we know corresponds to the plaintext m_1, `money transfer to account 314156:  $0001.00`, we can easily XOR the ciphertext with a mask to get a new plaintext c_2 that corresponds to our new message m_2, `money transfer to account 271828:  $9999.99`, by simply taking $c_1 \oplus m_2 \oplus m_1$. This requires no cryptanalysis of the algorithm itself. Can be mitigated with a good **message authentication code (MAC)**.

Stream ciphers in practice

- Not as popular these days, because many algorithms from the past few decades have now been easily cracked.
- eSTREAM specification is a good source for modern, secure stream ciphers, e.g. Salsa20/20, which as of now (November 2016) has no published attacks.
- RC4 is a popular stream cipher used in WEP and WPA for Wi-Fi and in TLS to protect HTTPS data.
  - According to Snowden, the NSA can likely crack it, but we don't know how.
- Often vulnerable to issues such as key reuse (same as two-time pad) and bit-flipping.

Case study: RC4

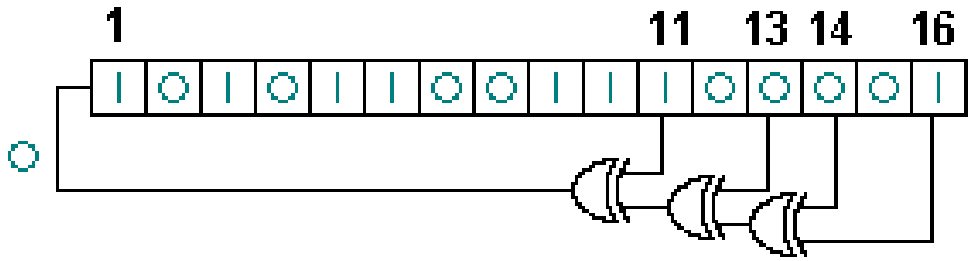
- Used to be really popular, now it's discouraged but still in use.
- Vulnerable to all sorts of amazing attacks:
  - **Distinguishing attack**: second byte of output has a $\frac{1}{128}$ chance of being zero, should be $\frac{1}{256}$ (Martin and Shamir)
  - **Fluhrer, Martin and Shamir attack**: first few bytes of output are strongly correlated to key, solution is to drop first several bytes of output
  - **Bar mitzvah attack**
  - **NOMORE attack**: broke encrypted HTTPS cookies in 75 hours, which is a huge deal
- As you can see, not good. Stream ciphers in general tend to have iffier security.

Random numbers in cryptography

- Pseudorandom number generators are the heart of stream ciphers and serve important roles in many other cryptosystems as well.
- Be especially wary that your CSPRNG is actually a cryptographically secure PRNG. Many popular PRNGs (e.g. in programming languages' "random" functions) produce random-looking data, but data that appears "random" for many purposes often isn't suited to cryptography.
- If you want the highest level of security to, for example, choose your key, use "true randomness", usually derived from microscale fluctuations in environmental factors such as atmospheric pressure or thermal noise.

Linear-feedback shift registers

Here's an example of a CSPRNG.



A 16-bit LFSR with characteristic polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1 \pmod 2$ (backwards).

To generate the next random number, an LFSR will take its current value and compute its **characteristic polynomial**, shifting its value one to the left, discarding the leftmost bit, and replacing the rightmost bit with the result of the computation.

Linear-feedback shift registers

- Linear operation, so relatively easy to cryptanalyze and pretty insecure.
- Finite number of states, so will eventually repeat.
- Still useful since it is very pseudorandom.
- Much more secure when non-linearity is introduced; can be combined into systems like *shrinking generators* for which no cryptanalytic attack is known.
- Used in ciphers like A5/1 and A5/2 in GSM cellphone networks and E0 in Bluetooth (all broken or seriously flawed).
- Very related to linear congruential generator, $r[n+1] = (ar[n] + b) \pmod{m}$ (super insecure).
  - Used in Java's `math.util.Random` and C++11's `minstd_rand`.