CISC 468: CRYPTOGRAPHY

LESSON 4: RANDOM NUMBER GENERATION

Furkan Alaca

TODAY, WE WILL LEARN ABOUT...

1. The difference between:
    - True random number generators (TRNGs),
    - Pseudorandom number generators (PRNGs), and
    - Cryptographically-secure pseudorandom number generators (CSPRNG).
2. How random number generation is used in cryptography.

READINGS

- Section 2.2.1 (Random Number Generators), Paar & Pelzl
- Section 2.2.3 (Towards Practical Stream Ciphers), Paar & Pelzl

RANDOM NUMBER GENERATORS

- All stream ciphers use a keystream to encrypt and decrypt data with the extremely simple XOR function
- The real complexity of a stream cipher lies in the generation of the keystream
- The keystream must be *indistinguishable from random*
  - Otherwise, an attacker may exploit the predictability of the keystream to recover the secret plaintext

TRUE RANDOM NUMBER GENERATORS (TRNGS)

True random number generators (TRNGs) are based on non-deterministic physical processes, e.g.,

- Flipping coins
- Rolling dice
- Thermal noise
- Radioactive decay

Thus, generating large amounts of true random numbers is expensive and time-consuming.

BASIC CHARACTERISTICS OF TRUE RANDOM NUMBERS

- Uniform distribution: Frequency of occurrence of each number should be approximately equal
- Independence: No value in the sequence can be inferred from the other

PSEUDORANDOM NUMBER GENERATORS (PRNGS)

- Pseudorandom number generators (PRNGs) compute sequences of numbers from an initial seed value
- The *Linear Congruential Generator* (LCG) is a widely-used class of PRNGs:

$$s_0 = \text{seed}$$
$$s_{i+1} \equiv a \times s_i + b \bmod m, \;\; i = 0, 1, \ldots$$

where a, b, and m are integer constants.

LCG EXAMPLE

The `rand()` function used in ANSI C is an example of an LCG:

$$s_0 = 12345$$
$$s_{i+1} \equiv 1103515245 s_i + 12345 \bmod 2^{31}, \quad i = 0, 1, \ldots$$

- Note that all PRNGs are deterministic (not truly random), so given the same seed as input, a PRNG will always generate the same output sequence
  - The sequence will also repeat periodically (the length of the period depends on the parameters chosen)

AN LCG-BASED STREAM CIPHER: KEYSTREAM GENERATION

- Alice and Bob want to exchange data, but instead of agreeing on a keystream in advance they will use an LCG to generate the keystream on-demand
- Alice and Bob agree on two secret integers a and b, and a publicly-known integer m, each 30 digits long
  - a and b can be considered a symmetric key
  - 60 decimal digits (a and b) is equivalent to 200 binary bits — more than enough to withstand a brute-force attack
  - Each output integer from the LCG can be zero-padded to a 100-bit value

AN LCG-BASED STREAM CIPHER: ENCRYPTION & DECRYPTION

- Alice encrypts a message by computing a sequence of pseudorandom numbers that is long enough to XOR the entirety of the plaintext message
- Bob decrypts a message by computing a sequence of pseudorandom numbers that is long enough to XOR the entirety of the ciphertext received from Alice

AN LCG-BASED STREAM CIPHER: AVOIDING KEYSTREAM REUSE

- To avoid reusing any portion of the keystream, upon exchanging a message Alice and Bob must either:
  - Agree on a new a and b for the next message (how?)

    or
  - For the next message, instead of restarting the LCG from the initial value, they should generate the keystream starting right after the value that was last generated

ATTACKING AN LCG-BASED STREAM CIPHER (1)

- Assume Oscar is eavesdropping on Alice and Bob, and records all ciphertext
- Assume Oscar knows the first 300 bits of plaintext
- This is not unrealistic — consider below the first few lines of an HTTP request sent to `www.queensu.ca` by Firefox 89 on Ubuntu 21.04

```
GET / HTTP/1.1
Host: www.queensu.ca
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101 Firefox/89.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-CA,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate, br
```

ATTACKING AN LCG-BASED STREAM CIPHER (2)

- Since Oscar knows both the first 300 bits of plaintext and 300 bits of ciphertext, Oscar can recover the first 300 bits of the keystream
  - This can be done in the case of any stream cipher, as follows

$$s_i \equiv y_i + x_i \bmod m, \;\; i = 0, 1, \ldots, 300$$

ATTACKING AN LCG-BASED STREAM CIPHER (3)

- The first 300 bits of the keystream gives Oscar the first three outputs of the LCG, s_1, s_2, and s_3
- Oscar now has a system of two equations and two unknowns:

$$s_2 \equiv as_1 + b \bmod m$$
$$s_3 \equiv as_2 + b \bmod m$$

- Solving the above system of equations gives Oscar a and b, which is the secret key

ATTACKING AN LCG-BASED STREAM CIPHER: LESSONS LEARNED

1. PRNGs are not suitable for use in stream ciphers.
2. For a PRNG to be cryptographically secure, it must ensure that given n bits of the keystream, e.g.,

$$s_1, s_2, \ldots, s_n,$$

it is computationally infeasible to:

- recover the key;

  or

- compute any subsequent portion of the keystream.

LINEAR FEEDBACK SHIFT REGISTERS (LFSRS)

- LFSRs are another mechanism for generating random numbers, however, being a linear system makes them susceptible to cryptanalysis much like LCGs
  - See Sections 2.2.1 to 2.2.2
  - LFSRs are easy and efficient to implement in hardware
- Combining multiple LFSRs with non-linear functions was a common technique used to build many historical stream ciphers such as A5/1 and A5/2 (used in GSM phones)
  - Trivium is a more recent example: See Section 2.2.3

CRYPTOGRAPHICALLY SECURE PRNGS (CPRNGS)

- CPRNGs are not truly random, but are **computationally infeasible** to distinguish from a truly random sequence of numbers
  - Slower than PRNGs, so they are essentially never used outside of cryptographic applications

MODERN STREAM CIPHERS

- True random numbers are hard to generate in large quantities, but we **can** generate a small true random seed value (e.g., 128 bits) and use a CPRNG to "stretch" it into a keystream
  - This is essentially what modern stream ciphers do
- ChaCha20 is a modern stream cipher widely-used with HTTPS
- RC4 was a very popular stream cipher used in WiFi encryption (WEP and WPA) and over a decade with HTTPS
  - Considered insecure as of 2015

RC4

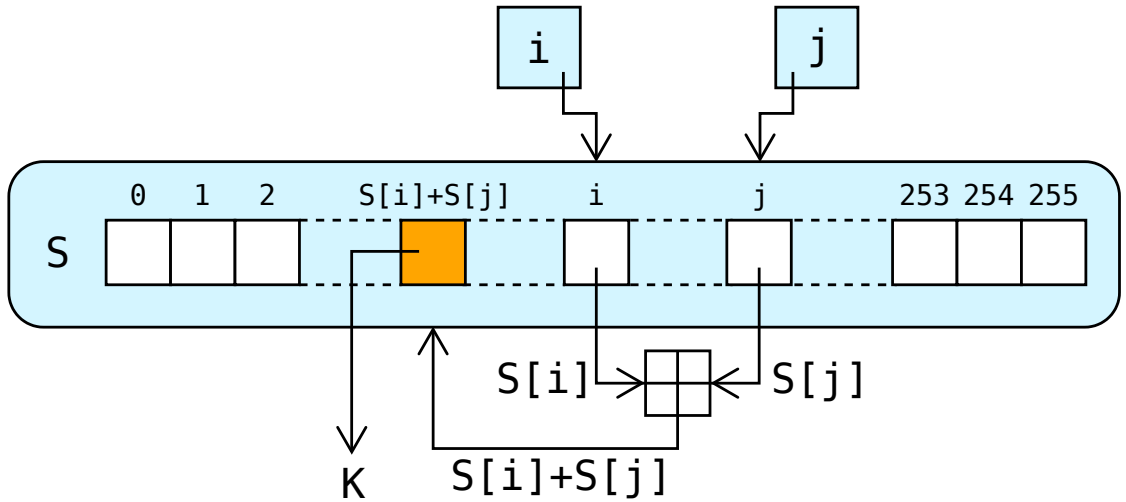
- RC4 consists of two algorithms: The *key-scheduling algorithm* (KSA) and the *pseudo-random generation algorithm* (PRGA)
- The KSA uses a variable-length key (40 to 2048 bits) to initialize a secret internal state
- The PRGA iteratively modifies the internal state to generate a stream of cryptographically-secure pseudorandom bytes

RC4 KSA

```
(* The 255-byte array S is the internal state,
   initialized to the identity permutation. *)
for i from 0 to 255
    S[i] := i
endfor

(* S is then "scrambled" using the secret key. *)
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

RC4 PRGA




```
i := 0
j := 0
(* Each loop iteration yields one byte of the keystream *)
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

AVOIDING KEYSTREAM REUSE WITH RC4 (1)

- As with any stream cipher, keystream cannot be reused in whole or in part
- Solution 1: Agree on a new key each time a message is to be sent
  - Seems cumbersome, but public-key cryptography (will be covered later) makes this easier

AVOIDING KEYSTREAM REUSE WITH RC4 (2)

- Solution 2: Use a long-term secret key and a per-message *nonce* ("number used once") to generate a short-term per-message key
  - Must be properly "mixed" (concatenating with key is insufficient); e.g., using a cryptographic hash function (will be covered later)
- A nonce is not secret, and can be sent in the clear, but it must be long enough to ensure that same value is never reused
  - WEP used a 24-bit nonce, which (alongside other problems) facilitated an attack that can reconstruct the key by capturing/analyzing a few minutes worth of wireless traffic

AVOIDING KEYSTREAM REUSE WITH CHACHA20

- The WEP attack shows that even when using a cipher that is known (or was known, at the time) to be secure, a protocol built on it may not be secure
- Modern stream ciphers like ChaCha20 incorporate the nonce as an input to the algorithm itself, avoids
- The inputs to ChaCha20 are:
  - 256-bit key
  - 32-bit initial counter (usually set to 0 or 1)
  - 96-bit nonce, also called an *initialization vector*
  - An arbitrary-length plaintext

RECAP

- True RNGs are used to generate short (e.g., 128 or 256-bit) secret keys
  - Generating long sequences of true random numbers is costly
- Stream ciphers use cryptographically-secure PRNGs to "stretch" a secret key into a keystream
  - The keystream must be indistinguishable from random to be considered secure
- A stream cipher requires the use of a per-message nonce (or a new key) to avoid reusing the keystream in whole or in part

BONUS: RC4 TRIVIA

- Due to cryptanalysis attacks, a widespread recommendation was to discard an initial portion of the keystream, e.g., RFC 4345 recommends discarding the first 1536 bytes to ensure that the internal state is "thoroughly mixed"
- RC4 is proprietary and the name is trademarked by RSA Security; its implementation was never officially released, but it was reverse engineered within days of release
- Third-party or open-source implementations are often called "ARC4", for "Alleged RC4"