MAT 302: LECTURE SUMMARY

GUEST LECTURE BY AARON CHOW

Notation

An integer X can be written as:

$$X = \sum_{j=0}^{n-1} x_j 2^j = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + 2x_1 + x_0$$

where each $x_j \in \{0, 1\}$. Here, X is said to be an n-bit integer. Hence, there are three ways to represent an n-bit integer:

$$X = x_{n-1}x_{n-2}\cdots x_1x_0 = (x_0, x_1, \ldots, x_{n-2}, x_{n-1}).$$

The first way is just, well, the usual way. The second term is the binary expansion of X. The last term is the sequence/vector representation.

1. Stream Ciphers and Block Ciphers

1.1. **The One-Time Pad (OTP).** The one-time pad is an encryption scheme that uses a sequence of random numbers. Suppose Alice wants to transmit a message to Bob securely.

(1) Let $s_0, s_1, s_2, \ldots$ be a sequence of random 0's and 1's (called the *keystream*).
(2) The keystream is known only to Alice and Bob.
(3) Every keystream bit s_i is used only once.

Then Alice can encrypt the plaintext $x_0, x_1, x_2, \ldots$ by computing

$$y_i = e_{s_i} = x_i + s_i \pmod 2 \quad i = 0, 1, 2, \ldots.$$

The ciphertext $y_0, y_1, y_2, \ldots$ is sent to Bob, who decrypts it by computing

$$x_i = d_{s_i} = y_i + s_i \pmod 2 \quad i = 0, 1, 2, \ldots.$$

The problem with this scheme is that the keystream bits s_i can never by reused, so the keystream length must equal the length of the plaintext. That is, to encrypt a 1000-bit message, a 1000-bit secret keystream must first be pre-shared!

Date: January 20th, 2011.

1.2. **Stream Cipher.** The one-time pad serves as a template for the general stream cipher. A stream cipher takes plaintext $x_0, x_1, x_2, \ldots$ and a keystream $s_0, s_1, s_2, \ldots$, and produces the ciphertext $y_0, y_1, y_2, \ldots$ via the encryption

$$y_i = e_{s_i} = x_i + s_i \pmod 2 \quad i = 0, 1, 2, \ldots.$$

Decryption is given by

$$x_i = d_{s_i} = y_i + s_i \pmod 2 \quad i = 0, 1, 2, \ldots.$$

Notice encryption and decryption are the same functions.

1.3. **Block Ciphers.** A block cipher processes the plaintext in blocks. Specifically, the plaintext is broken up into blocks, so $X = X_0, X_1, \ldots$. Each block is then encrypted with the secret key K. Therefore, each block of the ciphertext of a block cipher has the form

$$Y_i = f_K(X_i) \quad i = 0, 1, \ldots$$

with the ciphertext $Y = Y_0, Y_1, \ldots$.

In practice, block ciphers are used in most applications.

Examples of block ciphers:

(1) DES (Data Encryption Standard) has a block size of 64 bits and key size 56 bits.
(2) AES (Advanced Encryption Standard) has a block size of 128 bits and supports a key size of 128, 192, or 256 bits.

2. Security of Ciphers

2.1. Unconditional Security.

Definition 1 (Unconditional Security)**.** A cryptosystem is unconditionally secure if it cannot be broken even with infinite computational resources.

Example 1. OTP is unconditionally secure. To see this, observe that the ciphertext bits satisfy the linearly independent equations

$$y_i = x_i + s_i \pmod 2 \quad i = 0, 1, 2, \ldots.$$

An attacker knowing y_i cannot determine x_i with probability greater than 50% since $x_i = 0$ or 1 are equally likely if s_i is truly random.

2.2. **Computational Security.**

Definition 2 (Computational Security)**.** A cryptosystem is computationally secure if the best known attack takes an infeasible number of operations.

Example 2. Imagine a cryptosystem with a 100,000-bit key length and the only attack is the brute force attack (exhaustive search). Then 2^{100000} computations will break it, so it not unconditionally secure! However, 2^{100000} is far beyond current capabilities, so it is computationally secure.

3. Random Number Generators (RNG)

A *random number generator* is a function that outputs an (infinite) sequence of "random" numbers $s_0, s_1, s_2, \ldots$. RNGs are important because the security of stream ciphers depends entirely on this sequence.

3.1. **True Random Number Generators (TRNG).**

A *true random number generators* is a RNG whose output cannot be reproduced. For example, one can flip a coin and record a 0 or 1 depending on which side lands; this is a TRNG. TRNGs usually arises from physical processes such as semi-conductor noise, clock jitter in circuits, etc. In cryptography, TRNGs are needed for sessional keys distributed to the relevant communicating parties.

3.2. **Pseudorandom Number Generators (PRNG).**

A *pseudorandom number generator* generate sequences computed from an initial value, called the *seed*. It has the form

$$\begin{aligned} S_0 &= \text{seed} \\ S_{i+1} &= f(S_i) \quad i = 0, 1, 2, \ldots. \end{aligned}$$

More generally, it has the form

$$\begin{aligned} S_0 &= \text{seed} \\ S_{i+1} &= f(S_i, S_{i-1}, \ldots, S_{i-t}) \quad i = 0, 1, 2, \ldots \end{aligned}$$

for some fixed integer t.

Example 3 (Linear Congruential Generator)**.** Let a, b, and m be fixed integers.

$$\begin{aligned} S_0 &= \text{seed} \\ S_{i+1} &= AS_i + B \pmod{M} \quad i = 0, 1, 2, \ldots \end{aligned}$$

If the parameters A and B are chosen carefully,then this generator will have good statistical properties. For example, the ASCI C function `rand()` uses $A = 1103515245$, $B = 12345$, $M = 2^{31}$, and $S_0 = 12345$.

Note that PRNGs are not truly random (i.e. can be reconstructed) since the sequence is computed and is therefore completely deterministic. Instead, they try to "imitate" TRNGs in that the sequence produced possesses "good" statistical properties (i.e. passes certain statistical tests).

PRNGs are useful, especially in simulations, where there are no physical processes for a TRNG. However, they are usually not suitable for cryptographic applications such as stream ciphers.

3.3. **Cryptographically Secure Pseudorandom Number Generators (CSPRNG).** A *cryptographically secure pseudorandom number generator* is a PRNG with two additional properties:

(1) It is "unpredictable"; that is, given n bits of the keystream $s_i, s_{i+1}, \ldots, s_{i+n-1}$ it is computationally infeasible to compute the next bit s_{i+n}.
(2) Given $s_i, s_{i+1}, \ldots, s_{i+n-1}$ it is computationally infeasible to compute the previous bit s_{i-1}.

Example 4 (Blum-Blum-Shub)**.** Let $M = PQ$ where P and Q are two large primes. Let

$$X_{i+1} = X_i^2 \pmod{M}$$

The output keystream bit s_i is either the parity-bit of X_i (i.e. the parity of the number of 1's in the binary expansion), or the rightmost bit of X_i (sometimes referred to as the least significant bit).

The problem of distinguishing the output of the Blum-Blum-Shub generator from a string of random bits has been shown to be as computationally difficult as that of factoring M. This is good news, because factoring is currently considered to be a hard problem (when P and Q are large enough primes). Although the BBS generator itself is quite simple to implement, it is too slow to be used in practice.

4. Linear Feedback Shift Registers (LFSR)

A *linear feedback shift register* consists of clocked storage (called flip-flops) and a feedback path connecting some/all of them.

Example 5 (Simple LFSR)**.** Mathematical description: Let (s_0, s_1, s_2) be the initial state.

$$\begin{aligned} s_3 &= s_1 + s_0 \pmod 2 \\ s_4 &= s_2 + s_1 \pmod 2 \\ &\vdots \\ s_{i+3} &= s_{i+1} + s_i \quad i = 0, 1, 2, \ldots . \end{aligned}$$

4.1. **General Linear Feedback Shift Register.** In general, each flip-flop has a switch into the feedback path. Let

$$p_i = \begin{cases} 1 & \text{if the } i\text{-th switch is active} \\ 0 & \text{if the } i\text{-th switch is inactive.} \end{cases}$$

Suppose there are m flip-flops. Then a general linear feedback shift register of degree m can be described by

$$s_{i+m} = \sum_{j=0}^{m-1} p_j s_{i+j} \pmod 2.$$

The output is related to previous outputs by a linear equation, hence the equations define *linear recurrences*.

There are a finite number of flip-flops, therefore the output of a LFSR is periodic.

Theorem 3. *The maximum sequence length generated by a LFSR of degree m is $2^m - 1$.*

Proof. Note that once the internal flip-flops assumes a state that has previously occurred, the output will start to repeat. Also, the zero state must be avoided, otherwise the LFSR will be "stuck". Since there are m flip-flops, the number of different nonzero m-bit string is $2^m - 1$.