Understanding the definition:

Can we learn the least significant bit of a message given only the ciphertext (assuming a semantically-secure cipher)

No! Suppose we could. Then, adversary can choose two messages m_0, m_1 that differ in their least significant bit and distinguish with probability 1.

This generalizes to <u>any</u> efficiently-computable property of the two messages.

How does semantic security relate to perfect secrecy?

<u>Theorem.</u> If a cipher satisfies perfect secrecy, then it is semantically secure.

<u>Proof.</u> Perfect secrecy means that $\forall\, m_0, m_1 \in \mathcal{M},\ c \in \mathcal{C}$:

$$\Pr[k \xleftarrow{R} K : \mathrm{Encrypt}(k, m_0) = c] = \Pr[k \xleftarrow{R} K : \mathrm{Encrypt}(k, m_1) = c]$$

Equivalently, the distributions

$$\underbrace{\{k \xleftarrow{R} K : \mathrm{Encrypt}(k, m_0)\}}_{D_0} \quad \text{and} \quad \underbrace{\{k \xleftarrow{R} K : \mathrm{Encrypt}(k, m_1)\}}_{D_1}$$

are <u>identical</u> ($D_0 \equiv D_1$). This means that the adversary's output b' is identically distributed in the two experiments, and so

$$\mathrm{SSAdv}[A, \Pi_{SE}] = |W_0 - W_1| = 0.$$

<u>Corollary.</u> The one-time pad is semantically secure.

encryption key (PRG seed)

$c \leftarrow G(s) \oplus m$

$m \leftarrow G(s) \oplus c$

Seems straightforward, but takes some care to prove

<u>Theorem.</u> Let G be a secure PRG. Then, the resulting stream cipher constructed from G is semantically secure.

<u>Proof.</u> Consider the semantic security experiments:

Experiment 0: Adversary chooses m_0, m_1 and receives $c_0 = G(s) \oplus m_0$

Experiment 1: Adversary chooses m_0, m_1 and receives $c_1 = G(s) \oplus m_1$

Want to show that adversary's output in these two experiments are indistinguishable

Let $W_0 = \Pr[A \text{ outputs } 1 \text{ in Experiment } 0]$

$W_1 = \Pr[A \text{ outputs } 1 \text{ in Experiment } 1]$

<u>Idea:</u> If $G(s)$ is uniform random string (i.e., one-time pad), then $W_0 = W_1$. But $G(s)$ is like a one-time pad!

Define Experiment $0'$: Adversary chooses m_0, m_1 and receives $c_0 = t \oplus m_0$ where $t \xleftarrow{R} \{0,1\}^n$

Experiment $1'$: Adversary chooses m_0, m_1 and receives $c_1 = t \oplus m_1$ where $t \xleftarrow{R} \{0,1\}^n$

Define W_0', W_1' accordingly.

First, observe that $W_0' = W_1'$ (one-time pad is perfectly secure).

Now we show that $|W_0 - W_0'| = \text{negl}$ and $|W_1 - W_1'| < \text{negl}$.

$$\Rightarrow |W_0 - W_1| = |W_0 - W_0' + W_0' - W_1' + W_1' - W_1|$$

$$\leq |W_0 - W_0'| + |W_0' - W_1'| + |W_1' - W_1| \quad \text{by triangle inequality}$$

$$= \text{negl.} + \text{negl.} = \text{negl.}$$

Show. If G is a secure PRG, then for all efficient A, $|W_0 - W_0'| = \text{negl}$.

Common proof technique: prove the contrapositive.

Contrapositive: If A can distinguish Experiments 0 and 0', then G is not a secure PRG.

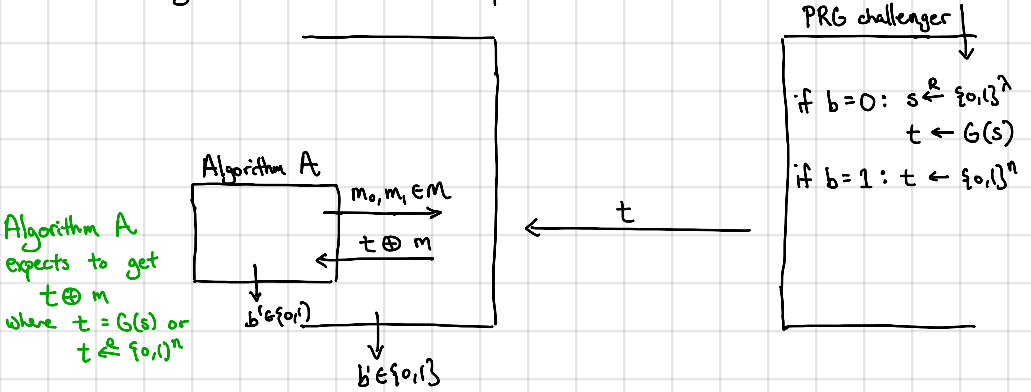
Suppose there exists efficient A that distinguishes Experiment 0 from 0'

$\Rightarrow$ We use A to construct efficient adversary B that breaks security of G.

↳ this step is a reduction

[we show how adversary (i.e., algorithm) for distinguishing Exp. 0 and 0' $\Rightarrow$ adversary for PRG]

Algorithm B (PRG adversary):



Running time of B = running time of A = efficient

Compute PRGAdv$[B, G]$.

$\Pr[B \text{ outputs } 1 \text{ if } b = 0] = W_0$ ← if $b=0$, then A gets $G(s) \oplus m$ which is precisely the behavior in Exp. 0

$\Pr[B \text{ outputs } 1 \text{ if } b = 1] = W_0'$ ← if $b=1$, then A gets $t \oplus m$ which is precisely the behavior in Exp. 0'

$\Rightarrow$ PRGAdv$[B,G] = |W_0 - W_0'|$, which is non-negligible by assumption. This proves the contrapositive.

Important note: Security of above schemes shown assuming message space is $\{0,1\}^n$ (i.e., all messages are n-bits long)

In practice: We have variable-length messages. In this case, security guarantees indistinguishability from other messages of the same length, but length itself is leaked [inevitable if we want short ciphertexts]

↳ can be problematic — see traffic analysis attacks!

So far, we have shown that if we have a PRG, then we can encrypt messages efficiently (stream cipher)

Question : Do PRGs exist?

Unfortunately, we do not know!

$n > \lambda$

Claim: If PRGs with non-trivial stretch exist, then $P \neq NP$.

Proof. Suppose $G : \{0,1\}^\lambda \to \{0,1\}^n$ is a secure PRG. Consider the following decision problem:

on input $t \in \{0,1\}^n$, does there exist $s \in \{0,1\}^\lambda$ such that $t = G(s)$

This problem is in NP (in particular, s is the witness). If G is secure, then no polynomial-time algorithm can solve this problem (if there was a polynomial-time algorithm for this problem, then it breaks PRF security with advantage $1 - \frac{1}{2^{n-\lambda}} > \frac{1}{2}$ since $n > \lambda$). Thus, $P \neq NP$.

In fact, there cannot even be a probabilistic polynomial-time algorithm that solves this problem with probability better than $\frac{1}{2} + \varepsilon$ for non-negligible $\varepsilon > 0$. This means that there is no BPP algorithm that breaks PRG security:

if PRGs exist, then $NP \not\subseteq BPP$

bounded error probabilistic polynomial time

"randomized algorithms that solves problem with bounded (constant) error"

Thus, proving existence of PRG requires resolving long-standing open questions in complexity theory!

$\Rightarrow$ Cryptography : We will assume that certain problems are hard and base constructions of (hopefully small) number of conjectures.

- Hardness assumptions can be that certain mathematical problems are intractable (e.g., factoring)
  - ↳ typically for public-key cryptography (2nd half of this course)
- Hardness assumptions can be that certain constructions are secure (e.g., "AES is a secure block cipher")
  - ↳ typically for symmetric cryptography
  - ↳ constructions are more ad hoc, rely on heuristics, but very fast in practice

Examples of stream ciphers (PRGs): designed to be very fast (oftentimes with hardware support)

- Linear congruential generator (e.g., rand() function in C)

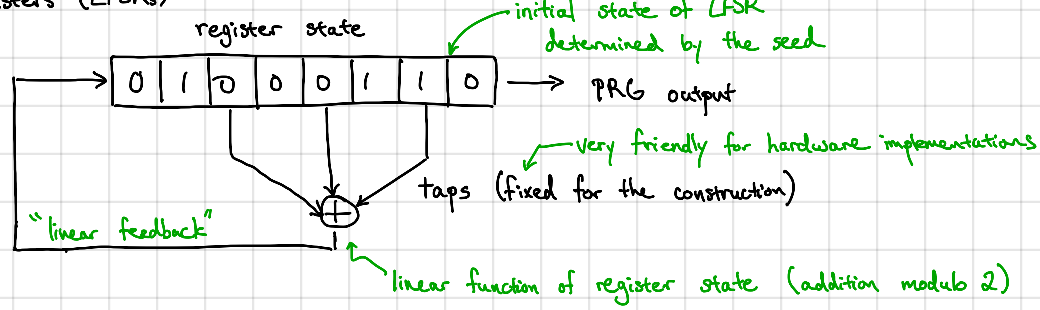
  $r_{i+1} = a r_i + b \pmod{m}$

  a, b, m are public constants
  r_0 is the initial seed
  } very simple, easy to implement (especially when m is a power of 2)
  → typical implementation: output is a few bits of $r_0, r_1, r_2, \ldots$ (full value of $r_0, r_1, r_2, \ldots$ never revealed) or $\lfloor r_i / w \rfloor$

  ↳ need to choose so outputs have long period

  <u>Not</u> a cryptographic PRG: NEVER USE rand() TO GENERATE CRYPTOGRAPHIC KEYS!
  - Given full outputs, outputs fully predictable (if enough bits of state revealed, can brute force unknown bits)
  - Even given partial outputs (e.g., least significant few bits of output) and having secret a, b, m, can still be broken (linear functions are not secure! see Boneh-Shoup Ch. 3.7.1 and related papers)
  - Often good enough for non-cryptographic applications (e.g., statistical simulation)

- Linear feedback shift registers (LFSRs)



<u>Each iteration</u>: rightmost bit is output by LFSR
bits at tap positions are xored and shifted in from the left
1 clock cycle = 1 output bit — very simple and fast!

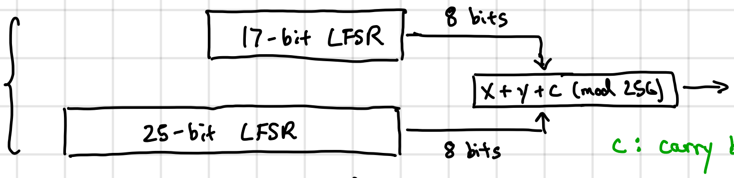
By itself, LFSR is totally broken: after observing n-bits of output, the entire state of the LFSR is known and subsequent bits are completely predictable!

<u>Proposal</u>: Use multiple LFSRs and combine in some non-linear way:

Example: CSS (content scrambling system) for DVD encryption [1996]

↳ actual CSS encryption has a few differences, but the core attack is unaffected

1 bit of LFSR is 1 ⇓ 40-bit key



c: carry bit from previous operation (initially 0)

(needed to comply with export control restrictions)

- Brute-force attack: guess the seed ($\sim 2^{40}$ time)
- Can do much better with more clever strategy
  - ↳ General idea: - if we know a few bytes of output of the stream cipher and the output of the 17-bit LFSR, can subtract to obtain output of 25-bit LFSR
    - brute force the seed of the 17-bit LFSR, each guess induces a state for the 25-bit LFSR
    - check if output matches or not
  - ↳ Attack now runs in $\sim 2^{16}$ time
    - By 1999, full key-recovery attack on can recover key from DVD in just ~18 seconds on 450 MHz processor [totally broken!]

Pre-2000s

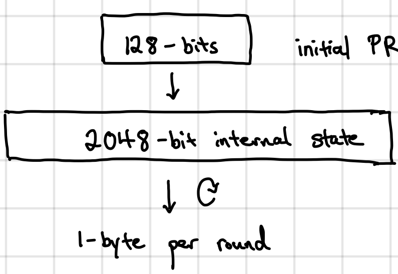
Other examples: GSM encryption (A5/1,2 stream ciphers for encrypting GSM cell phone traffic)
- ↳ xor outputs of 3 LFSRs
- ↳ tried to keep cipher design private, but eventually reverse engineered and attacks found

Snowden documents: NSA can process encrypted A5/1

Never rely on security by obscurity!

Bluetooth E0 stream cipher uses a design based on 4 LFSRs in conjuction with a 2-bit finite state machine — also not secure!

- RC4 (1987) stream cipher (widely used - SSL/TLS protocol, 802.11b)



Numerous problems:
- Bias in initial output: $\Pr[\text{second byte} = 0] = \frac{2}{256} > \frac{1}{256}$
  - ↳ When using RC4, recommendation is to ignore first 256 bytes due to potential bias
  - ↳ Correlations in output: probability of seeing (0,0) in output is $\frac{1}{256^2} + \frac{1}{256^3} > \frac{1}{256^2}$
  - ↳ Given outputs of RC4 with related keys (e.g., keys sharing common suffix), possible to recover keys after seeing few blocks of output
    - ↳ Can be very problematic on weak devices (who may not have good sources of entropy)

- Modern stream ciphers (eSTREAM project: 2004-2008)
  - Salsa20 (2005) ⇝ ChaCha (2008)
    - ↳ core design maps 256-bit key, 64-bit nonce, 64-bit counter onto a 512-bit output

enables using same key (and different nonces) to encrypt multiple messages (will discuss later)

allows random access into the stream

Design is more complex:
- relies on a sequence of rounds
- each round consists of 32-bit additions, xors, and bit-shifts

↳ very fast even in software (4-14 CPU cycles/output byte) — used to encrypt TLS traffic between Android and Google services