Quick recap from last time: PRG is function $G : \{0,1\}^\lambda \to \{0,1\}^n$ that takes short seed and stretches it to long random-looking string

PRG $\Rightarrow$ stream cipher (use PRG to obtain random string that looks like a OTP):

Encrypt(k,m): $G(k) \oplus m$

Decrypt(k,c): $G(k) \oplus c$

replaces the OTP

Why is this secure? To any efficient adversary, $G(k)$ looks like uniform random string

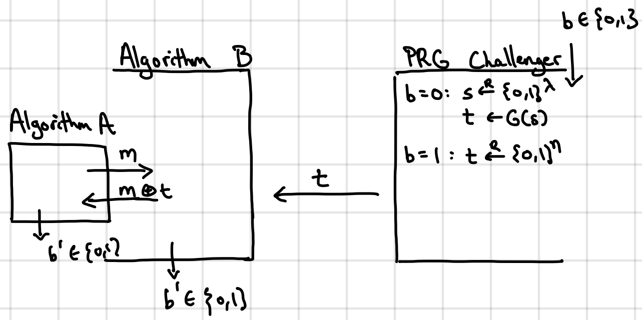
- Idea: real scheme will encrypt as $G(k) \oplus m$

  Secretly... we now replace $G(k)$ with $t \xleftarrow{R} \{0,1\}^n$ and set ciphertext to $t \oplus m$

  PRG security $\Rightarrow$ efficient adversary cannot tell we did this, since otherwise, we break PRG security!

Formally: if there exists A that can distinguish $G(k) \oplus m$ from $t \oplus m$, then there is B that can distinguish $G(k)$ from t (breaking PRG security)

Reduction: Construct B from A:



Let $W_0' = \Pr[A \text{ outputs } 1 \text{ given } G(k) \oplus m]$ ← assumption: $|W_0' - W_1'|$ non-negligible

$W_1' = \Pr[A \text{ outputs } 1 \text{ given } t \oplus m]$

Then:

$\Pr[B \text{ outputs } 1 \mid b = 0]$
$= \Pr[A \text{ outputs } 1 \text{ given } G(s) \oplus m]$
$= W_0'$

$\Pr[B \text{ outputs } 1 \mid b = 1]$
$= \Pr[A \text{ outputs } 1 \text{ given } t \oplus m]$
$= W_1'$

$\text{PRGAdv}[B, G] = |W_0' - W_1'|$ is non-negligible

Conclusion: efficient adversary cannot tell if it sees $G(k) \oplus m$ or $t \oplus m$

↳ but if ciphertext is $t \oplus m$, this is a OTP encryption, so m perfectly hidden

thus, no efficient adversary can distinguish $m_0 \oplus G(k)$ from $m_1 \oplus G(k)$ and semantic security holds

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)

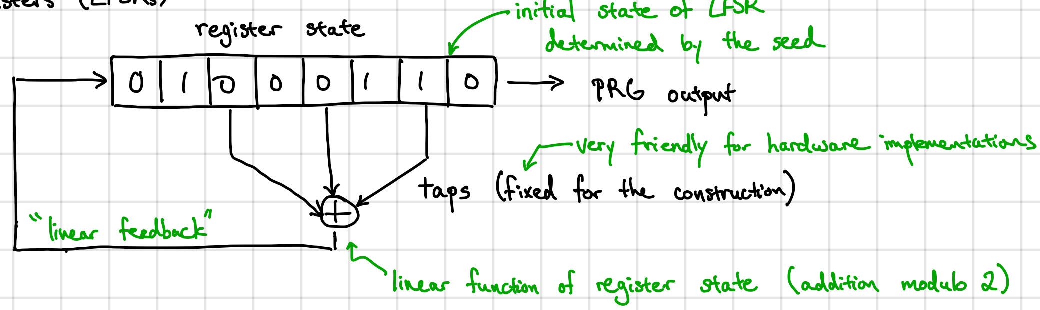
  → need to choose so outputs have long period

  → 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 $\lfloor r_i / w \rfloor$

  Not 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)



Each iteration: 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!

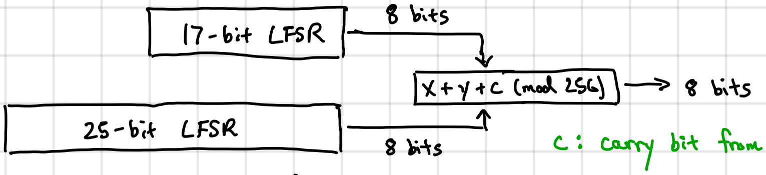
Proposal: Use multiple LFSRs and combine in some non-linear way:

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

(needed to comply with export control restrictions)



c: carry bit from previous operation (initially 0)

- 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

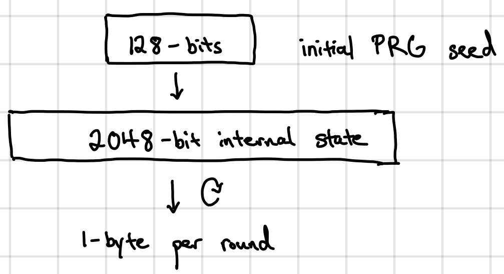
↳ tried to keep cipher design private, but eventually reverse engineered and attacks found

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

Recall: the one-time pad is not reusable (i.e., the two-time pad is totally broken) - see HW1A!

NEVER REUSE THE KEY TO A STREAM CIPHER!

But wait... we "proved" that a stream cipher was secure, and yet, there is an attack?

Recall security game:

$b \in \{0,1\}$

adversary $\qquad\qquad$ challenger

$\xrightarrow{m_0,\, m_1}$ $\qquad k \xleftarrow{R} \mathcal{K}$

$\xleftarrow{c_b}$ $\qquad c_b \leftarrow \text{Encrypt}(k, m_b)$

$b' \in \{0,1\}$

Observe: adversary only sees one ciphertext
key is only used once

$\Rightarrow$ Security in this model says nothing about multiple messages / ciphertexts

Problem: If we want security with multiple ciphertexts, we need a different or stronger definition (CPA security)