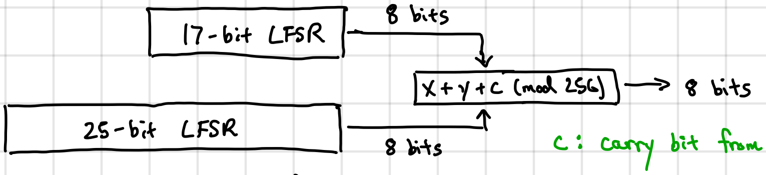
Example: CSS (content scrambling system) for DVD encryption [1996]

↳ 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

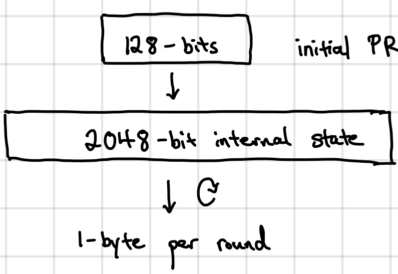
↳ tried to keep cipher design private, but eventually reverse engineered and attacks found

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




Numerous problems:
- Bias in initial output: Pr[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)

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:

| adversary | | challenger ($b \in \{0,1\}$) |
|---|---|---|
| | $m_0, m_1 \longrightarrow$ | $k \xleftarrow{R} \mathcal{K}$ |
| | $\longleftarrow c_b$ | $c_b \leftarrow \text{Encrypt}(k, m_b)$ |
| $\downarrow$ $b' \in \{0,1\}$ | | |

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

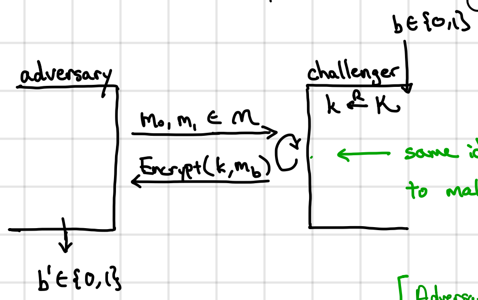
$\Rightarrow$ Security in this model says nothing about multiple messages / ciphertexts

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

**Definition:** An encryption scheme $\Pi_{SE} = (\text{Encrypt}, \text{Decrypt})$ is secure against chosen-plaintext attacks (CPA-secure) if for all efficient adversaries $\mathcal{A}$:

$$\text{CPAAdv}[\mathcal{A}, \Pi_{SE}] = \left| \Pr[W_0 = 1] - \Pr[W_1 = 1] \right| = \text{negl.}$$

where $W_b$ ($b \in \{0,1\}$) is the output of the following experiment:



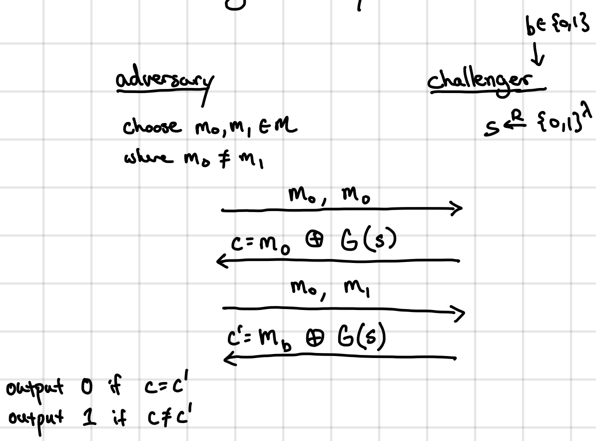

same idea as in original semantic security game, but allow adversary to make encryption queries (also called a "left-or-right" oracle)

output of experiment $W_b$

[Adversary's goal is to guess which of $m_0$ or $m_1$ was encrypted, given access to an encryption oracle (i.e., adversary gets to see encryptions of messages of its choice.]

**Claim.** A stream cipher is not CPA-secure.

**Proof.** Consider the following adversary:




$\Pr[b' = 1 \mid b = 0] = 0$ since $c' = m_0 \oplus G(s) = c$

$\Pr[b' = 1 \mid b = 1] = 1$ since $c' = m_1 \oplus G(s) \neq c$

$\Rightarrow \text{CPAAdv}[\mathcal{A}, \Pi_{SE}] = 1$

**Observe:** Above attack works for any deterministic encryption scheme.

$\Rightarrow$ CPA-secure encryption must be randomized!

$\Rightarrow$ To be reusable, cannot be deterministic. Encrypting the same message twice should not reveal that identical messages were encrypted.

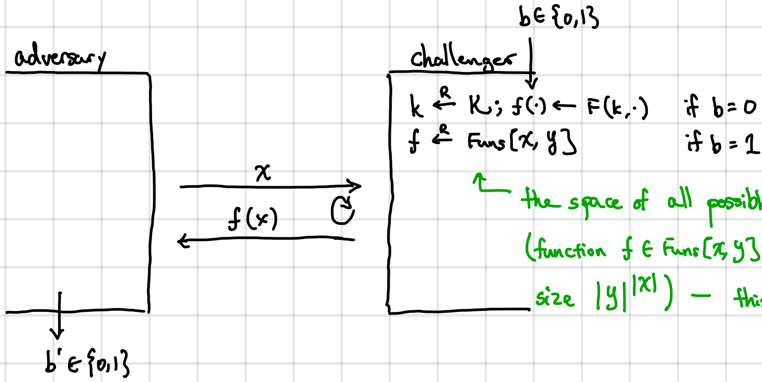
To build a CPA-secure encryption scheme, we will use a "block cipher"

- Block cipher is an invertible keyed function that takes a block of $n$ input bits and produces a block of $n$ output bits
- Examples include 3DES (key size 168 bits, block size 64 bits)
  AES (key size 128 bits, block size 128 bits)

Will define block ciphers abstractly first: pseudorandom functions (PRFs) and $\underbrace{\text{pseudorandom permutations (PRPs)}}_{\text{block ciphers}}$

↳ General idea: PRFs behave like random functions
PRPs behave like random permutations

**Definition.** A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with key-space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$ is a pseudorandom function (PRF) if for all efficient adversaries $\mathcal{A}$, $|W_0 - W_1| = \text{negl.}$, where $W_b$ is the probability the adversary outputs 1 in the following experiment:




the space of all possible functions from $\mathcal{X} \to \mathcal{Y}$ (function $f \in \text{Funs}[\mathcal{X}, \mathcal{Y}]$ can be represented by a truth table of size $|\mathcal{Y}|^{|\mathcal{X}|}$) — this is usually exponentially large!

$$\text{PRFAdv}[\mathcal{A}, F] = |W_0 - W_1| = \left|\Pr[\mathcal{A} \text{ outputs } 1 \mid b = 0] - \Pr[\mathcal{A} \text{ outputs } 1 \mid b = 1]\right|$$

**Intuitively:** input-output behavior of a PRF is indistinguishable from that of a random function (to any computationally-bounded adversary)

3DES: $\{0,1\}^{168} \times \{0,1\}^{64} \to \{0,1\}^{64}$ $\quad |\mathcal{K}| = 2^{168}$ $\quad |\text{Funs}[\mathcal{X}, \mathcal{Y}]| = (2^{64})^{(2^{64})}$

AES: $\{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ $\quad |\mathcal{K}| = 2^{128}$ $\quad |\text{Funs}[\mathcal{X}, \mathcal{Y}]| = (2^{128})^{(2^{128})}$

} space of random functions is exponentially-larger than key-space!

**Definition:** A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ is a pseudorandom permutation (PRP) if

- for all keys $k$, $F(k, \cdot)$ is a permutation and moreover, there exists an efficient algorithm to compute $F^{-1}(k, \cdot)$:

$$\forall k \in \mathcal{K} : \forall x \in \mathcal{X} : F^{-1}(k, F(k, x)) = x$$

- for $k \xleftarrow{R} \mathcal{K}$, the input-output behavior of $F(k, \cdot)$ is computationally indistinguishable from $f(\cdot)$ where $f \xleftarrow{R} \text{Perm}[\mathcal{X}]$ and $\text{Perm}[\mathcal{X}]$ is the set of all permutations on $\mathcal{X}$ (analogous to PRF security)

**Note:** a block cipher is another term for PRP (just like stream ciphers are PRGs)