Problem: OTP perfectly secret but $|k| > |m|$

Recall: Perfect Indistinguishability

$\Pr_{C,M} = \Pr_{\Sigma C, M} = \frac{1}{2}$ for all $M \in m$, $m \in \mathcal{M}$, prob. dist. over $\mathcal{M}$

Last time:

Designed algorithm to distinguish distributions using Chernoff tail bounds.

Inspirations:
- model computationally bounded adversary
- negligible difference from perfect
- game-style definition of security

"Computational indistinguishability in the presence of an eavesdropper" A adversary $\xrightarrow{\text{p.p.t.}}$ chooses $m_0, m_1$, $\xrightarrow{\text{p.p.t.}}$ challenge $c$

Verifier $k \leftarrow \text{Gen}(1^n)$, p.p.t. algorithm

$\xrightarrow{\text{p.p.t.}}$ $b \leftarrow \text{Enc}(m_b)$ challenge ciphertext

$\xrightarrow{\text{p.p.t.}}$ $\text{Enc}(1^n)$, p.p.t. algorithm

$\Pr(A \text{ succeeds}) \leq \frac{1}{2} + \varepsilon(n)$ $\varepsilon(n)$ negligible

- poly-time computable $f, h$

$\Pr[A\left(1^n, \text{Enc}_k(m), h(m)\right) = f(m)] - \Pr[A'\left(1^n, h(m)\right) = f(m)] \leq \varepsilon(n)$ $\varepsilon(n)$ negligible

- probabilities taken over $m, k$, randomness of $A, A', \text{Enc}$

Thus semantic security $\implies$ computational indistinguishability

Ex: Let Enc be such that $A$ can compute LSB of $m$ from $\text{Enc}(m)$

$\Pr(A \text{ succeeds}) = 1 > \frac{1}{2} + \varepsilon(m) \implies \text{Enc not semantically secure}$
Def "pseudorandom generator"

- deterministic, poly-time algorithm
- $G : \{0, 1\}^n \rightarrow \{0, 1\}^{3(n)}$
  - expansion: $|\ell(n)|=n$
  - pseudorandomness: $\forall$ p.p.t. distinguisher $D$
    \[ |\Pr[D(r)=1] - \Pr[D(G(s))=1]| < \epsilon(n) \]
    $r$ uniform in $\{0, 1\}^{\ell(n)}$
    $s \leftarrow \{0, 1\}^n$

Stream Cipher Encryption

Gen: $k \leftarrow \{0, 1\}^n$ uniformly random
Enc: $c = G(k) \oplus m$ (m $\in \{0, 1\}^{\ell(n)}$)
Dec: $m = G(k) \oplus c$

Thm: If $G$ is a PRG $\implies$ above has computationally indistinguishable encryptions.

Pf (by reduction.) Show if $A$ can distinguish encryptions, then $G$ is not PRG.

\[ \omega = \text{uniformly random or pseudorandom} \]
\[ (1 \text{ if } b' = 1, \text{otherwise } 0) \]
\[ w = \text{random} \quad \Pr[D(r)=1] = \frac{1}{2} \]
\[ A: \quad G(s) \implies \Pr[D(G(s))=1] = \Pr[A \text{ succeeds}] > \frac{1}{2} + \epsilon \]
(by assumption)
\[ |\Pr[D(r)=1] - \Pr[D(G(s))=1]| > \epsilon(n) \]

Problem: What about multiple encryptions?

Experiment: $A(m_0, m_1, m_2) \leftarrow (m_0, m_1, m_2)$

Then 3 encryption schemes indistinguishable in the presence of an adversary but not under multiple encryption:

Pf: $\text{Enc}_k(0^n), \text{Enc}_k(0^n) v. \text{Enc}_k(0^n), \text{Enc}_k(1^n)$

Thm: No deterministic encryption $A(s)$ has indistinguishable multiple encryptions

(Example: many-time pad protocol. Never reuse keys for a stream cipher!) (WhatsApp!)
Fixing Stream cipher for multiple encryptions:
- augment with initialization vector (IV)

\[ E_{k}(m) = (IV, G(k, IV)\oplus m) \]

requirement: \( G(k, IV) \) pseudorandom even when IV known

Stream cipher in practice:
- RC4
  - 1987: Rivest Cipher 4
  - Trade secret at RSA Corporation
  - September 1994: Code posted anonymously to cipher-related mailing list, sci.crypt newsgroup
    - used in SSL/TLS, WEP, WPA, Kerberos, etc.
  - 2001: Fluhrer, Mantin, Shamir
    - weaknesses in key scheduling algorithm
    - repeated IVs allow break
    - 2nd byte of keystream is biased
    - used to crack WEP (aircracking implements this and related attacks)

- biases in initial bytes

- 2013: Alfi  ğ dan, Bernstein, Paterson, Paillier, Schindler
  - Attack against RC4 in TLS
    - 1. Empirically measure bias probabilities for each byte
    - 2. Encrypt fixed (unknown) plaintext with many random keys and compare to empirical distribution of byte probabilities
    - 3. Return \( n \) maximizing probability

- LFSR
  - DVD encryption, GSM encryption, Bluetooth all broken
  - (CSU) (AS/1, AS/2) (60)

- Salsa 20, ChaCha
  - Dan Bernstein
Generating Random Numbers in Practice

Non-cryptographic RNGs:
- Linear Congruential Generator
  \[ X_{n+1} = (ax_n + b) \mod m \]
  implemented in C rand (also uses LFSR)
  easily broken (e.g., Monte Carlo simulation etc.)

Cryptographically suitable RNGs:
- `/dev/random`
- `/dev/urandom`
  "entropy from the environment"
  \[ \Downarrow \]
  "Entropy pool" - test/measurement/estimate
  - entropy of inputs
  - refuse to output if inputs have no entropy

```
state

PRG \( g_1 \) refresh/update

\[ \Downarrow \]

next/output PRG \( g_2 \)

output
```

Security Model:
- attacker can see/influence some inputs
- attacker can see outputs
- attacker may be able to see state sometimes

Dodis, Pointcheval, Ruhault, Vergnaud, Vechos 2013
"/dev/random is not robust"

Security Goals:
- forward secrecy
  adversary can't predict past output even if state is compromised
- backward secrecy
  adversary cannot influence inputs
- resilience
  adversary cannot predict future outputs even if state is compromised