Announcements

Homework 2 due tomorrow

Grades and comments should be visible on Blackboard
Last Time

$|\text{key}| \geq |\text{total information encrypted}|$ is necessary for statistical security

Computational Security

PRGs
Encryption Security Experiment

\[ \lambda \quad m_0, m_1 \in M_\lambda \quad c \quad b \]

Challenger

\[ k \leftarrow K_\lambda \quad c \leftarrow \text{Enc}(k, m_b) \]

\[ \text{IND-Exp}_b(\lambda) \]
Encryption Security Definition

Definition: \((\text{Enc}, \text{Dec})\) has ciphertext indistinguishability if, for all probabilistic polynomial time (PPT) \(\mathcal{A}\), there exists a negligible function \(\varepsilon\) such that

\[
| \Pr[1 \leftarrow \text{IND-Exp}_0(\mathcal{A}, \lambda)] - \Pr[1 \leftarrow \text{IND-Exp}_1(\mathcal{A}, \lambda)] | \leq \varepsilon(\lambda)
\]
Construction with $|k| << |m|$

Idea: use OTP, but have key generated by some expanding function $G$
Pseudorandom Generators

Definition: \( G: \{0,1\}^* \rightarrow \{0,1\}^* \) is a secure pseudorandom generator (PRG) if:

- \( G \) is computable in polynomial time
- \( G \) applied to \( \lambda \) bit strings produces strings of length \( t(\lambda) > \lambda \)
- \( G \) is deterministic
- For all PPT, \( \exists \ negl \; \epsilon \) such that:

\[
\left| \Pr \left[ (G(s))=1 : s \leftarrow \{0,1\}^\lambda \right] - \Pr \left[ (x)=1 : x \leftarrow \{0,1\}^{t(\lambda)} \right] \right| \leq \epsilon(\lambda)
\]
This Time

Stream Ciphers

Design of PRGs
Pseudorandom Generators

PRGs usually allow for streaming arbitrarily long sequences of random bits
Stream Ciphers

Use “streaming” PRG to encrypt messages

Keystream

Message
Stream Ciphers

Use “streaming” PRG to encrypt messages

In this way, can encrypt arbitrarily long messages
• security proof similar to last time

But remember, stream ciphers are really just OTP’s, so still cannot encrypt twice with the same part of keystream

Instead, encrypt like we did with the one-time pad
Multiple Messages with Stream Ciphers
Multiple Messages with Stream Ciphers
Limitations of Stream Ciphers

Just like with OTP, need to be careful because communication may be asynchronous
• Keep a different key/state for each direction of communication

Here, even bigger problem cause by out of order messages
Multiple Messages with Stream Ciphers
Multiple Messages with Stream Ciphers

\[ \text{state}_1 \oplus \text{state}_2 \]

\[ \text{key} \]

\[ 2 \]

\[ 1 \]

\[ 2 \]

\[ 2 \]
Multiple Messages with Stream Ciphers

\[ \text{state}_2 \oplus \text{state}_3 \oplus \text{key} \]

1 2 3 \ldots
Multiple Messages with Stream Ciphers

state_{n-1} \oplus n \rightarrow state_n

key

1 2 3 \ldots n
Multiple Messages with Stream Ciphers

Bob needs to either:

• Store entire keystream until he receives message 1
• Or re-compute keystream from scratch every message
Multiple Messages with Stream Ciphers

Out of order messages cause implementation difficulties

Mitigation?
• Self-synchronizing stream cipher
Self-Synchronizing Stream Ciphers

“state” is just (last several ciphertext bits seen, key)

Thus, you can always decrypt if the last several ciphertext bits were correct
How do we build PRGs?
Linear Feedback Shift Registers

In each step,
• Last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
• last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
• last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
• last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits

0 1 0 1 1
0 1

1 0
Linear Feedback Shift Registers

In each step,
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
• last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
• last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

In each step,
• last bit of state is removed and outputted
• Rest of bits are shifted right
• First bit is XOR of subset of remaining bits
Linear Feedback Shift Registers

Are LFSR’s secure PRGs? No!

First $n$ bits of output = initial state

Write $x = x_1, \ldots, x_n$, $x'$
Initialize LFSB to have state $x_1, \ldots, x_n$
Run LFSB for $|x|$ steps, obtaining $y$
Check if $y = x$
PRGs should be Unpredictable

More generally, it should be hard, given some bits of output, to predict subsequent bits

**Definition:** $G$ is unpredictable if, for all probabilistic polynomial time (PPT) and any polynomial $p$, there exists a negligible function $\varepsilon$ such that

$$\left| \Pr[G(s) \leftarrow (G(s)_{[1,p(\lambda)]}) - \frac{1}{2} \right| \leq \varepsilon(\lambda)$$
PRGs should be Unpredictable

More generally, it should be hard, given some bits of output, to predict subsequent bits

**Theorem:** $G$ is unpredictable iff it is pseudorandom
Proof

Pseudorandomness $\rightarrow$ Unpredictability

Assume towards contradiction PPT, polynomial $p$, non-negligible function $\varepsilon$ s.t.

$$\left| \Pr[G(s)_{p(\lambda)+1} \leftarrow (G(s)_{[1,p(\lambda)]})] - \frac{1}{2} \right| = \varepsilon(\lambda)$$
Proof

Pseudorandomness $\rightarrow$ Unpredictability

Construct

$x \rightarrow x_{[1,p(\lambda)]} \rightarrow 1 \oplus b \oplus x_{p(\lambda)+1}$
Proof

Pseudorandomness $\rightarrow$ Unpredictability

Analysis:

• If $x$ is random, $\Pr[1 \oplus b \oplus x_{p(\lambda)+1} = 1] = \frac{1}{2}$

• If $x$ is pseudorandom,

$$
\Pr[1 \oplus b \oplus x_{p(\lambda)+1} = 1]
= \Pr[G(s)_{p(\lambda)+1} \leftarrow (G(s)_{[1,p(\lambda)]})]
= \frac{1}{2} \pm \varepsilon(\lambda)
$$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Assume towards contradiction PPT $\text{PPT}$, non-negligible function $\varepsilon$ s.t.

$$\left| \Pr[ (G(s))=1 : s \leftarrow \{0,1\}^{\lambda} ] - \Pr[ (x)=1 : x \leftarrow \{0,1\}^{\lambda} ] \right| = \varepsilon(\lambda)$$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Hybrids:
$H_i$: $x_{[1,i]} \leftarrow G(s)$, $x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$

$H_0$: truly random $x$
$H_+:$ pseudorandom $t$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Hybrids:

$H_i$: $x_{[1,i]} \leftarrow G(s)$, $x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$

$$\Pr[\text{ } H_s(x)=1:x\leftarrow H_s] - \Pr[\text{ } H_0(x)=1:x\leftarrow H_0] = \varepsilon(\lambda)$$

Let $q_i = \Pr[\text{ } H_i(x)=1:x\leftarrow H_i]$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Hybrids:
$$H_i: \ x_{[1,i]} \leftarrow G(s), \ x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$$

$$|q_t - q_0| = \varepsilon(\lambda)$$

Let $q_i = \Pr[H_i(x) = 1: x \leftarrow H_i]$
Proof

Unpredictability $\Rightarrow$ Pseudorandomness

Hybrids:
$H_i: x_{[1,i]} \leftarrow G(s), \ x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$

By triangle inequality, there must exist an $i$ s.t.

$$|q_i - q_{i-1}| \geq \frac{\varepsilon(\lambda)}{t}$$

Can assume wlog that

$$q_i - q_{i-1} \geq \frac{\varepsilon(\lambda)}{t}$$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Construct

\[ y = G(s)[1,i-1] \]

\[ b \leftarrow \{0,1\} \]
\[ y' \leftarrow \{0,1\}^{t-i} \]
\[ x = y || b || y' \]
\[ b' \]
\[ 1 \oplus b \oplus b' \]
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Analysis:
• If $b = G(s)_i$, then $H_i$ sees $H_i$
  $\Rightarrow$ outputs 1 with probability $q_i$
  $\Rightarrow$ outputs $b = G(s)_i$ with probability $q_i$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Analysis:

- If $b = 1 \oplus G(s)_i$, then
  - Define $q'_i$ as $\Pr[\text{outputs } 1]$
  - $\frac{1}{2}(q'_i + q_i) = q_{i-1} \Rightarrow q'_i = 2q_{i-1} - q_i$
  - $\Rightarrow$ outputs $G(s)_{[1,i]}$ with probability
  - $1-q'_i = 1 + q_i - 2q_{i-1}$
Proof

Unpredictability $\rightarrow$ Pseudorandomness

Analysis:
- $\Pr[\text{outputs } G(s)_i]$
  
  $= \frac{1}{2} (q_i) + \frac{1}{2} (1 + q_i - 2q_{i-1})$

  $= \frac{1}{2} + q_i - q_{i-1}$

  $\geq \frac{1}{2} + \varepsilon(\lambda)/t$
Linearity
Linearity

LFSR’s are linear:

\[
\text{state'} = \begin{pmatrix}
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{pmatrix} \cdot \text{state}
\]

\[
\text{output} = (0 \ 0 \ 0 \ 0 \ 1) \cdot \text{state}
\]
Linearity

LFSR’s are linear:
• Each output bit is a linear function of the initial state (that is, $G(s) = A \cdot s \pmod{2}$)

Any linear $G$ cannot be a PRG
• Can check if $x$ is in column-span of $A$ using linear algebra
Introducing Non-linearity

Non-linearity in the output:

Non-linear feedback:
LFSR period

Period = number of bits before state repeats

After one period, output sequence repeats

Therefore, should have extremely long period
• Ideally almost $2^n$
• Possible to design LFSR’s with period $2^n - 1$
Hardware vs Software

PRGs based on LFSR’s are very fast in hardware

Unfortunately, not easily amenable to software
RC4

Fast software based PRG

Resisted attack for several years

No longer considered secure, but still widely used
RC4

State = permutation on $[256]$ plus two integers
• Permutation stored as 256-byte array $S$

**Init(16-byte $k$):**
• For $i=0,\ldots,255$
  \[ S[i] = i \]
• $j = 0$
• For $i=0,\ldots,255$
  \[ j = j + S[i] + k[i \mod 16] \pmod{256} \]
  Swap $S[i]$ and $S[j]$
• Output $(S,0,0)$
RC4

GetBits(S,i,j):
• i++ (mod 256)
• j+= S[i] (mod 256)
• Swap S[i] and S[j]
• t = S[i] + S[j] (mod 256)
• Output (S,i,j), S[t]

New state    Next output byte
Insecurity of RC4

Second byte of output is slightly biased towards 0
• $\Pr[\text{second byte} = 0^8] \approx 2/256$
• Should be $1/256$

Means RC4 is not secure according to our definition
• Outputs 1 iff second byte is equal to $0^8$
• Advantage: $\approx 1/256$

Not a serious attack in practice, but demonstrates some structural weakness
Insecurity of RC4

Possible to extend attack to actually recover the input $k$ in some use cases

• The seed is set to $(IV, k)$ for some initial value $IV$
• Encrypt messages as $RC4(IV,k) \oplus m$
• Also give $IV$ to attacker
• Cannot show security assuming RC4 is a PRG

Can be used to completely break WEP encryption standard
Extending the Stretch of a PRG

Suppose you have a fixed-stretch PRG \( G \)
- Better yet, a PRG that expands by a single bit
  \[ G: \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+1} \]

Construct a PRG \( G' \) of arbitrary output length
Extending the Stretch of a PRG
Security Proof

Assume towards contradiction PPT, non-negligible $\varepsilon$...

Define hybrids...
Security Proof

$H_0: \{0,1\}^\lambda$

\[ H_0: \{0,1\}^\lambda \]

\[ \text{seed} \rightarrow \text{state}_1 \rightarrow \text{state}_2 \rightarrow \text{state}_3 \rightarrow \ldots \]
Security Proof

$H_1$: 

\[
\{0,1\}^\lambda \xrightarrow{\text{state}_1} G \xrightarrow{\text{state}_2} G \xrightarrow{\text{state}_3} G \xrightarrow{\ldots}
\]
Security Proof

$$H_2:$$

\[
\{0,1\} \xrightarrow{\text{state}_2} \{0,1\}^\lambda \xrightarrow{\text{state}_3} \{0,1\} \xrightarrow{\ldots} \]

\[
\text{G} \xrightarrow{\text{state}_3} \text{G} \xrightarrow{\ldots}
\]
Security Proof

$H_+:

\{0,1\} \quad \{0,1\} \quad \{0,1\} \quad \{0,1\} \quad \ldots
Security Proof

$H_0$ corresponds to pseudorandom $x$

$H_t$ corresponds to truly random $x$

Let $q_i = \Pr[ H_i(x) = 1 : x \leftarrow H_t ]$

By assumption, $|q_t - q_0| = \varepsilon(\lambda)$

$\Rightarrow \exists i \text{ s.t. } |q_i - q_{i-1}| = \varepsilon(\lambda)/t$
Security Proof

\( y \)

\( \text{state}_i \) \( \text{G} \) \( \text{state}_{i+1} \) \( \text{G} \) \( \text{state}_{i+2} \) \( \text{G} \) \( \ldots \)
Security Proof

Analysis

• If \( y = G(s) \), then \( \text{sees} \ H_{i-1} \)
  \[ \Rightarrow \Pr[\text{outputs 1}] = q_{i-1} \]
  \[ \Rightarrow \Pr[\text{outputs 1}] = q_{i-1} \]

• If \( y \) is random, then \( \text{sees} \ H_i \)
  \[ \Rightarrow \Pr[\text{outputs 1}] = q_i \]
  \[ \Rightarrow \Pr[\text{outputs 1}] = q_i \]
Summary

Stream ciphers = secure encryption for arbitrary length, number of messages
               (though we did not completely prove it)

However, implementation difficulties due to having to maintaining state
Next Time

Stateless encryption for arbitrary messages

Pseudorandom Functions (PRFs)