Course Business

- Homework 2 Due on Friday at 11:30 AM
- Course Schedule Updated
  - “Duplicate week” removed
- Midterm is on March 1
- Final Exam is Monday, May 1 (7 PM)
  - Location: Right here

Cryptography
CS 555

Topic 16: Block Ciphers

An Existential Crisis?

- We have used primitives like PRGs, PRFs to build secure MACs, CCA-Secure Encryption etc…
- Do such primitives exist? In practice?
- How do we build them?

Recap

Last Class: Stream Ciphers

- Linear Feedback Shift Registers (and attacks)
- RC4 (and attacks)
- Trivium

Goals for This Week:

- Practical Constructions of Symmetric Key Primitives

Today's Goals: Stream Ciphers

- Block Ciphers

Pseudorandom Permutation

A keyed function $\mathrm{F}: \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$, which is invertible and "looks random" without the secret key k.

- Similar to a PRF, but
- Computing $F_k(x)$ *and* $F_k^{-1}(x)$ is efficient (polynomial-time)

Definition 3.28: A keyed function $\mathrm{F}: \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a **strong pseudorandom permutation** if for all PPT distinguishers D there is a negligible function μ s.t.

$$\left|Pr\left[D^{F_k(.),F_k^{-1}(.)}(1^n)\right] - Pr\left[D^{f(.),f^{-1}(.)}(1^n)\right]\right| \leq \mu(n)$$

Pseudorandom Permutation

Definition 3.28: A keyed function $\mathrm{F}: \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a **strong pseudorandom permutation** if for all PPT distinguishers D there is a negligible function μ s.t.

$$\left| Pr\left[D^{F_k(.),F_k^{-1}(.)}(1^n)\right] - Pr\left[D^{f(.),f^{-1}(.)}(1^n)\right] \right| \leq \mu(n)$$

Notes:

- the first probability is taken over the uniform choice of $k \in \{0,1\}^n$ as well as the randomness of D.
- the second probability is taken over uniform choice of f $\in$ $\mathbf{Perm}_{\mathbf{n}}$ as well as the randomness of D.
- D is *never* given the secret k
- However, D is given oracle access to keyed permutation and inverse

How many permutations?

- **$|Perm_n|$=?**
- **Answer:** $2^n!$
- How many bits to store f $\in$**$Perm_n$**?
- **Answer**:

$$\log(2^n!) = \sum_{i=1}^{2^n} \log(i)$$

$$\geq \sum_{i=2^{n-1}}^{2^n} n-1 \geq (n-1) \times 2^{n-1}$$

How many bits to store permutations?

$$\log(2^n!) = \sum_{i=1}^{2^n} \log(i)$$

$$\geq \sum_{i=2^{n-1}}^{2^n} n-1 \geq (n-1) \times 2^{n-1}$$

Example: Storing f ∈$\mathbf{Perm_{50}}$ requires over 6.8 petabytes (10^{15})

Example 2: Storing f ∈$\mathbf{Perm_{100}}$ requires about 12 yottabytes (10^{24})

Example 3: Storing f ∈$\mathbf{Perm_{8}}$ requires about 211 bytes

Attempt 1: Pseudorandom Permutation

- Select 16 random permutations on 8-bits $f_1,\ldots,f_{16} \in \mathbf{Perm_8}$.

- **Secret key:** $k = f_1,\ldots,f_{16}$ (about 3 KB)
- **Input**: $x=x_1,\ldots,x_{16}$ (16 bytes)

$$F_k(x) = f_1(x_1) \parallel f_2(x_2) \parallel \cdots \parallel f_{16}(x_{16})$$

- Any concerns?

Attempt 1: Pseudorandom Permutation

- Select 16 random permutations on 8-bits $f_1,\ldots,f_{16} \in \mathbf{Perm_8}$.

$$\mathrm{F_k}(x) = \mathrm{f_1(x_1)} \parallel \mathrm{f_2(x_2)} \parallel \cdots \parallel \mathrm{f_{16}(x_{16})}$$

- Any concerns?

$$\mathrm{F_k}(\mathrm{x_1} \parallel \mathrm{x_2} \parallel \cdots \parallel \mathrm{x_{16}}) = \mathrm{f_1(x_1)} \parallel \mathrm{f_2(x_2)} \parallel \cdots \parallel \mathrm{f_{16}(x_{16})}$$

$$\mathrm{F_k}(\mathbf{0} \parallel \mathrm{x_2} \parallel \cdots \parallel \mathrm{x_{16}}) = \mathbf{f_1(0)} \parallel \mathrm{f_2(x_2)} \parallel \cdots \parallel \mathrm{f_{16}(x_{16})}$$

- Changing a bit of input produces insubstantial changes in the output.
- A truly random permutation $\mathrm{F} \in \mathbf{Perm_{128}}$ would not behave this way!

Pseudorandom Permutation Requirements

- Consider a truly random permutation $\mathrm{F} \in \mathbf{Perm}_{128}$
- Let inputs x and x’ differ on a single bit
- We expect outputs F(x) and F(x’) to differ on approximately half of their bits
  - F(x) and F(x’) should be (essentially) independent.
- A pseudorandom permutation must exhibit the same behavior!

Confusion-Diffusion Paradigm

- Our previous construction was not pseudorandom, but apply the permutations do accomplish something
  - They introduce confusion into F
  - Attacker cannot invert (after seeing a few outputs)
- Approach:
  - **Confuse**: Apply random permutations f_1,…, to each block of input to obtain y1,…,
  - **Diffuse**: Mix the bytes y1,…, to obtain byes z1,…,
  - **Confuse**: Apply random permutations f_1,…, with inputs z1,…,
  - Repeat as necessary

Confusion-Diffusion Paradigm

Example:

- Select 8 random permutations on 8-bits $f_1,\ldots,f_{16} \in \mathbf{Perm_8}$
- Select 8 extra random permutations on 8-bits $g_1,\ldots,g_8 \in \mathbf{Perm_8}$

$F_k(x_1 \parallel x_2 \parallel \cdots \parallel x_8)$=

1. $y_1 \parallel \cdots \parallel y_8 := f_1(x_1) \parallel f_2(x_2) \parallel \cdots \parallel f_8(x_8)$
2. $z_1 \parallel \cdots \parallel z_8 := \mathbf{Mix}(y_1 \parallel \cdots \parallel y_8)$
3. **Output:** $f_1(z_1) \parallel f_2(z_2) \parallel \cdots \parallel f_8(z_8)$

Example Mixing Function

Mix$(y_1 \parallel \cdots \parallel y_8)$=

1. For i=1 to 8
2. $\quad z_i := y_1[i] \parallel \cdots \parallel y_8[i]$
3. End For
4. **Output:** $g_1(z_1) \parallel g_2(z_2) \parallel \cdots \parallel g_8(z_8)$

Substitution Permutation Networks

- S-box a public "substitution function" (e.g. $S \in \mathbf{Perm_8}$).

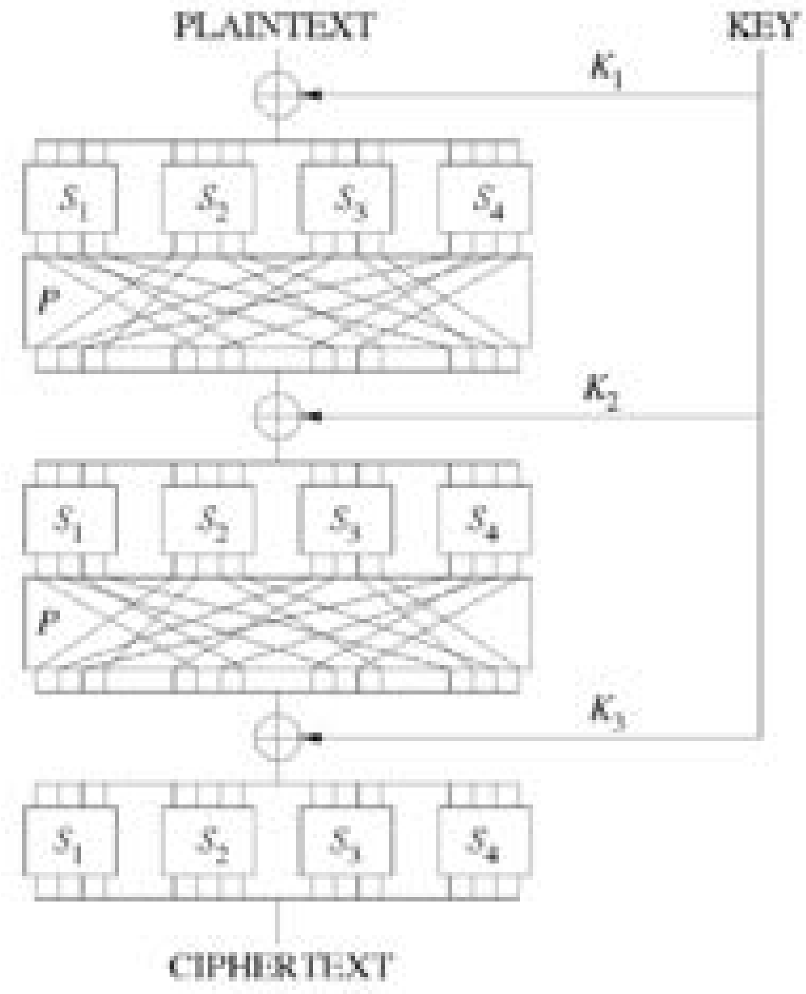
- S is not part of a secret key, but can be used with one

$$f(x) = S(x \oplus k)$$

- Input to round: x, k (k is subkey for current round)
- **Key Mixing**: Set $x := x \oplus k$
- **Substitution**: $x := S_1(x_1) \parallel S_2(x_2) \parallel \cdots \parallel S_8(x_8)$
- **Bit Mixing Permutation**: permute the bits of x to obtain the round output

Note: there are only n! possible bit mixing permutations of [n] as opposed to 2^n! Permutations of $\{0,1\}^n$

Substitution Permutation Networks



- **Proposition 6.3:** Let F be a keyed function defined by a Substitution Permutation Network. Then for any keys/number of rounds F_k is a permutation.

- Why? Composing permutations f,g results in another permutation h(x)=g(f(x)).

Remarks

- Want to achieve "avalanche effect" (one bit change should "affect" every output bit)

- Should a S-box be a random byte permutation?

- Better to ensure that S(x) differs from x on at least 2-bits (for all x)
  - Helps to maximize "avalanche effect"

- Mixing Permutation should ensure that output bits of any given S-box are used as input to multiple S-boxes in the next round

Remarks

- How many rounds?

- **Informal Argument:** If we ensure that S(x) differs from S(x') on at least 2-bits (for all x,x' differing on at least 1 bit) then every input bit effects
    - 2 bits of round 1 output
    - 4 bits of round 2 output
    - 8 bits of round 3 output
    - ….
    - 128 bits of round 4 output

- Need at least 7 rounds (minimum) to ensure that every input bit effects every output bit

Attacking Lower Round SPNs

- Trivial Case: One full round with no final key mixing step
- **Key Mixing**: Set $\mathrm{x} \coloneqq \mathrm{x} \oplus k$
- **Substitution**: $\mathrm{y} \coloneqq \mathrm{S}_1(\mathrm{x}_1) \parallel \mathrm{S}_2(\mathrm{x}_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8)$
- **Bit Mixing Permutation**: P permute the bits of y to obtain the round output

- Given input/output $(x, F_k(x))$
  - Permutations P and S_i are public and can be run in reverse
  - $P^{-1}(F_k(x)) = \mathrm{S}_1(\mathrm{x}_1 \oplus k_1) \parallel \mathrm{S}_2(\mathrm{x}_2 \oplus k_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8 \oplus k_8)$
  - $x_i \otimes k_i = S_i^{-1}\big(\mathrm{S}_1(\mathrm{x}_1 \oplus k_1)\big)$
  - Attacker knows x_i and can thus obtain k_i

Attacking Lower Round SPNs

- Easy Case: One full round with final key mixing step
- **Key Mixing**: Set $\mathrm{x} := \mathrm{x} \otimes k_1$
- **Substitution**: $\mathrm{y} := \mathrm{S}_1(\mathrm{x}_1) \parallel \mathrm{S}_2(\mathrm{x}_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8)$
- **Bit Mixing Permutation**: $\mathrm{z}_1 \parallel \cdots \parallel \mathrm{z}_8 = \mathrm{P}(\mathrm{y})$
- **Final Key Mixing**: Output $\mathrm{z} \oplus k_2$

- Given input/output $(x, F_k(x))$
  - Permutations P and S_i are public and can be run in reverse once k_2 is known
  - Immediately yields attack in 2^{64} time (k_1,k_2 are each 64 bit keys) which narrows down key-space to 2^{64} but we can do much better!

Attacking Lower Round SPNs

- Easy Case: One full round with final key mixing step
- **Key Mixing**: Set $\mathrm{x} \coloneqq \mathrm{x} \oplus k_1$
- **Substitution**: $\mathrm{y} \coloneqq \mathrm{S}_1(\mathrm{x}_1) \parallel \mathrm{S}_2(\mathrm{x}_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8)$
- **Bit Mixing Permutation**: $\mathrm{z}_1 \parallel \cdots \parallel \mathrm{z}_8 = \mathrm{P}(\mathrm{y})$
- **Final Key Mixing**: Output $\mathrm{z} \oplus k_2$

- Given input/output $(x, F_k(x))$
  - Permutations P and S_i are public and can be run in reverse once k_2 is known
  - Guessing 8 specific bits of k_2 (which bits depends on P) we can obtain one value $y_i = S_i(x_i \otimes k_i)$
  - Attacker knows x_i and can thus obtain k_i by inverting S_i and using XOR
  - Narrows down key-space to 2^{64} , but in time 8×2^8

Attacking Lower Round SPNs

- Easy Case: One full round with final key mixing step
- **Key Mixing**: Set $\mathrm{x} \coloneqq \mathrm{x} \oplus k_1$
- **Substitution**: $\mathrm{y} \coloneqq \mathrm{S}_1(\mathrm{x}_1) \parallel \mathrm{S}_2(\mathrm{x}_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8)$
- **Bit Mixing Permutation**: $\mathrm{z}_1 \parallel \cdots \parallel \mathrm{z}_8 = \mathrm{P}(\mathrm{y})$
- **Final Key Mixing**: Output $\mathrm{z} \oplus k_2$

- Given several input/output pairs $(x_j, F_k(x_j))$
  - Can quickly recover k_1 and k_2

Attacking Lower Round SPNs

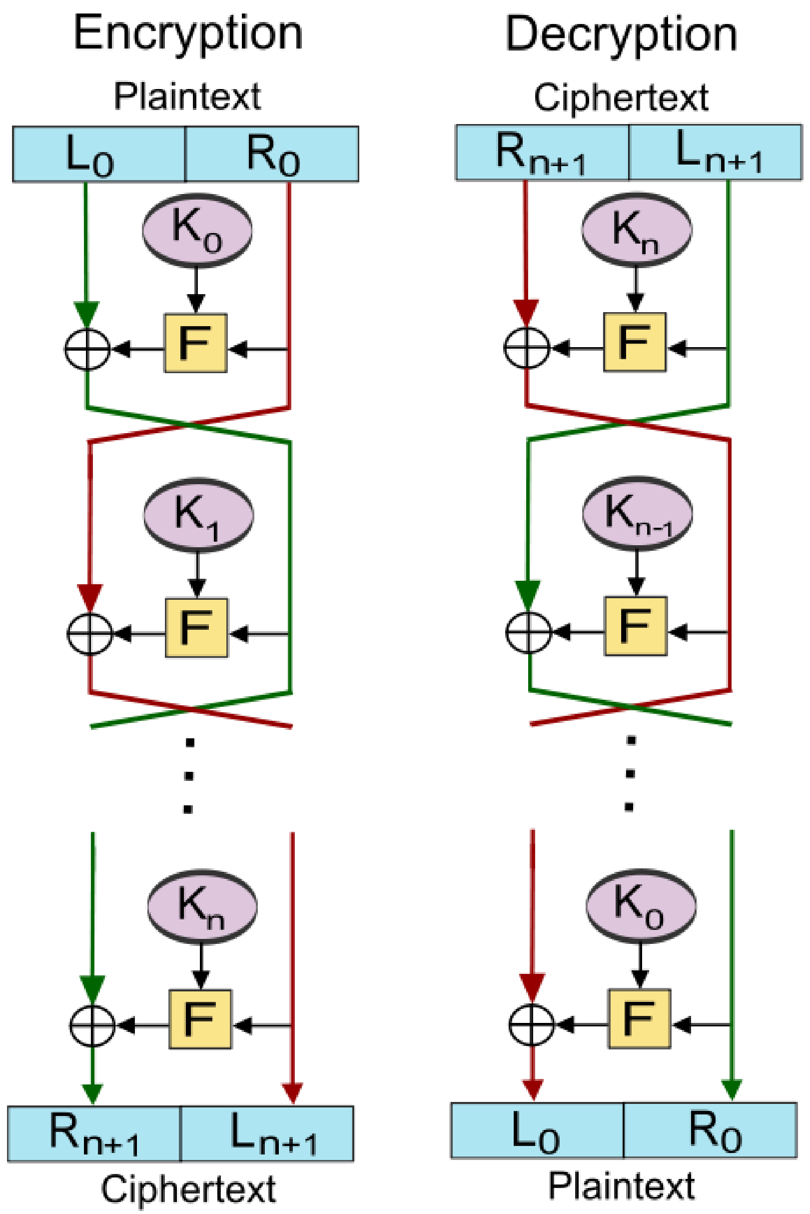
- Harder Case: Two round SPN

- Exercise ☺

Feistel Networks

- Alternative to Substitution Permutation Networks

- **Advantage**: underlying functions need not be invertible, but the result is still a permutation



- $R_{i-1} = L_i$
- $L_{i-1} := R_i \oplus F_{k_i}(R_{i-1})$

Proposition: the function is invertible.

Digital Encryption Standard (DES): 16-round Feistel Network.

Next class…

Next Class

- Read Katz and Lindell 6.2.3-6.2.4
- DES, 3DES & Attacks