Homework 2 Statistics

Minimum Value	40.00
Maximum Value	100.00
Range	60.00
Average	83.79
Median	85.00
Standard Deviation	15.93

Midterm Exam

- Thursday, October 5th at 9 AM (in class)
  - Multiple Choice, True/False, Fill-in-the-Blank
  - 75 minutes
- You may bring one (double sided) index card with notes
- No electronic devices/calculators
- May Incorporate Content from Today’s Lecture or Katz and Lindell Chapters 1--6

Final Exam

- **Time:** Tuesday, December 12th at 1PM (Tentative Subject to Change)
- **Location:** LWSN 1106

Recap

- Random Oracle Model
  - Pros (Easier Proofs/More Efficient Protocols/Solid Evidence for Security in Practice)
  - Cons (Strong Assumption)
- Hashing Applications
- Building Stream Ciphers
  - Linear Feedback Shift Registers (+ Attacks)
  - RC4 (+ Attacks)
  - Trivium
- Block Ciphers

Cryptography
CS 555

Week 7:

- Block Ciphers
- Feistel Networks
- DES, 3DES, AES

Readings: Katz and Lindell Chapter 6

CS 555: Week 7: Topic 1
Block Ciphers (Continued)

Review 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 $\in$**Perm$_n$**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?

- **$|\mathrm{Perm}_n|$=?**
- **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 $\in$**Perm$_{50}$** requires over 6.8 petabytes (10^{15})

Example 2: Storing f $\in$**Perm$_{100}$** requires about 12 yottabytes (10^{24})

Example 3: Storing f $\in$**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:** $\mathrm{k} = f_1,\ldots,f_{16}$ (about 3 KB)
- **Input**: $x=x_1,\ldots,x_{16}$ (16 bytes)

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

- Any concerns?

Attempt 1: Pseudorandom Permutation

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

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

- Any concerns?

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

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

- 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 y_1,…,
  - **Diffuse**: Mix the bytes y_1,…, to obtain byes z_1,…,
  - **Confuse**: Apply random permutations f_1,…, with inputs z_1,…,
  - 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:** $g_1(z_1) \parallel g_2(z_2) \parallel \cdots \parallel g_8(z_8)$

$$\begin{array}{cc} & \begin{array}{ccc} z_1 & \quad\quad & z_8 \end{array} \\ \begin{array}{c} y_1 = \\ \\ y_8 = \end{array} & \begin{bmatrix} y_1[1] & \cdots & y_1[8] \\ \vdots & \ddots & \vdots \\ y_8[1] & \cdots & y_8[8] \end{bmatrix} \end{array}$$

Substitution Permutation Networks

- S-box a public "substitution function" (e.g.$S \in$ **Perm$_8$**).

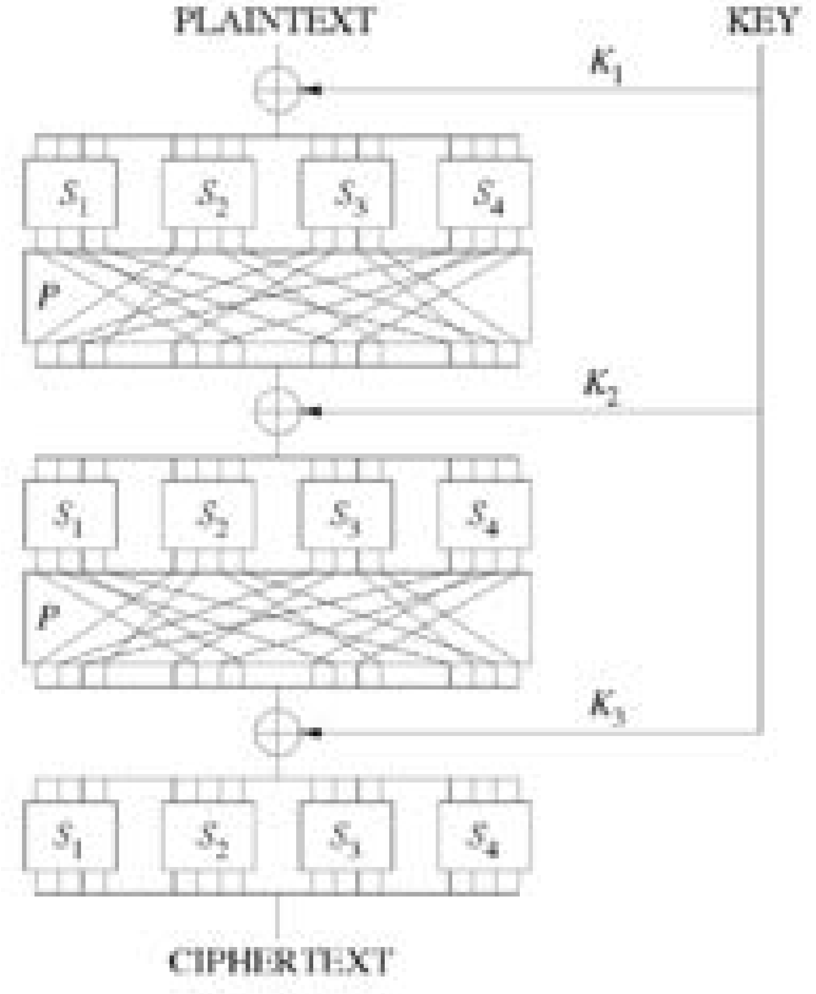
- S is not part of a secret key, but can be used with one

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

- 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 affects
  - 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 affects 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
  - $P^{-1}(F_k(x)) = \mathrm{S}_1(\mathrm{x}_1 \oplus k_1) \parallel \mathrm{S}_2(\mathrm{x}_2 \oplus k_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8 \oplus k_8)$
  - $x_i \otimes k_i = S_i^{-1}\left(\mathrm{S}_1(\mathrm{x}_1 \oplus k_1)\right)$
  - 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 $\mathrm{x} \coloneqq \mathrm{x} \otimes k_1$
- **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**: $\mathrm{z}_1 \parallel \cdots \parallel \mathrm{z}_8 = \mathrm{P(y)}$
- **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**: $\mathrm{z}_1 \parallel \cdots \parallel \mathrm{z}_8$ =P(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

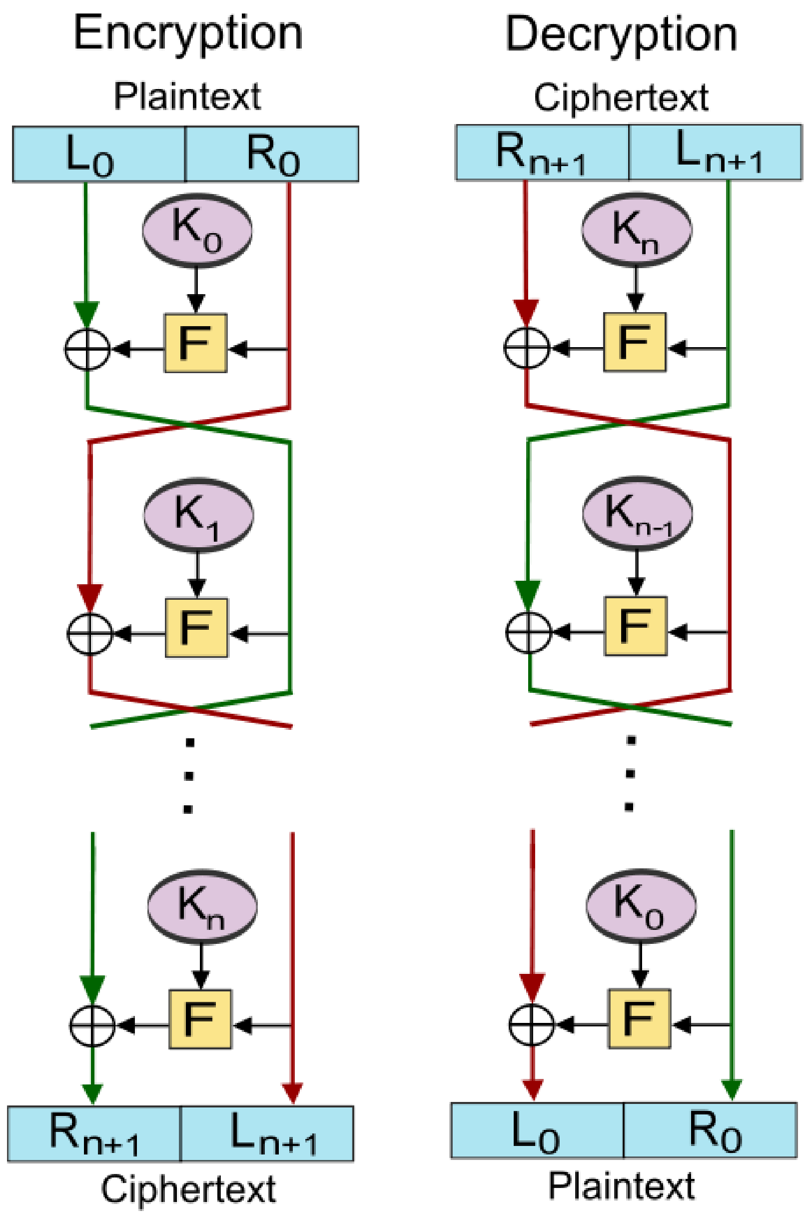
Attacking Lower Round SPNs

- 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.

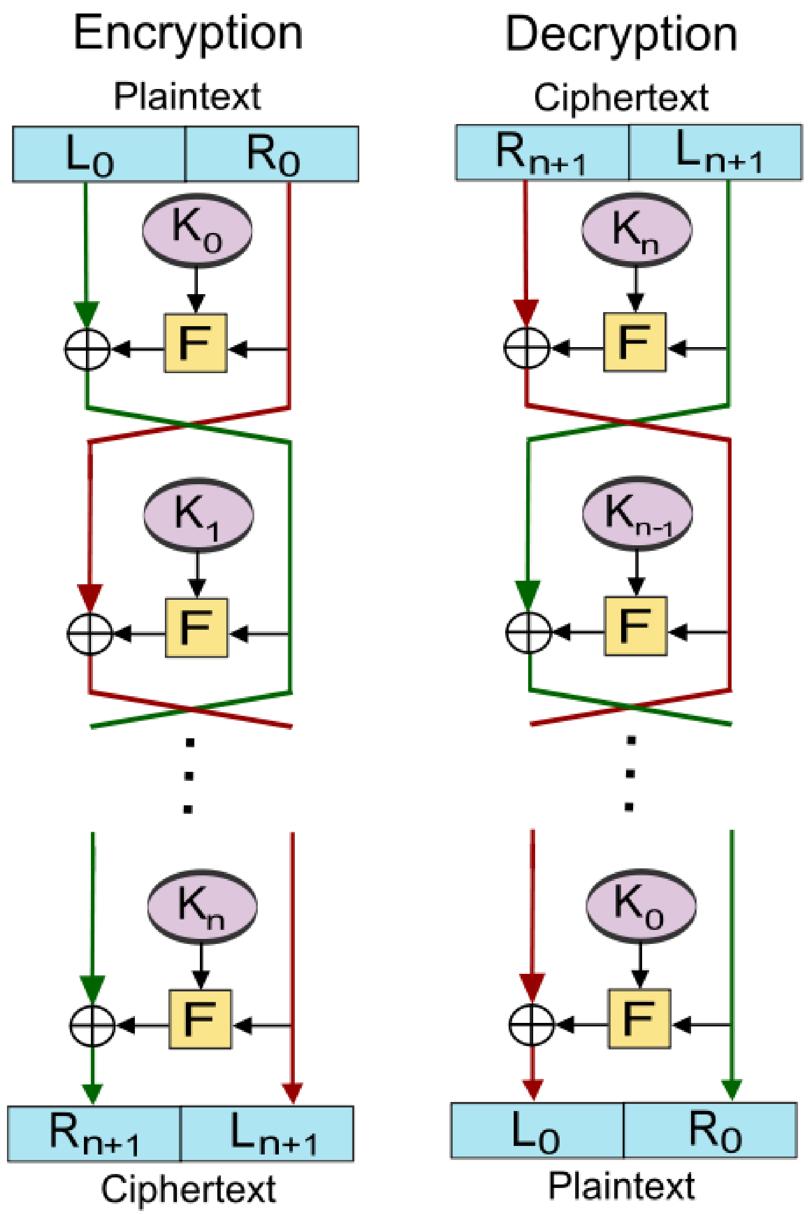
Digital Encryption Standard (DES): 16-round Feistel Network.

CS 555: Week 7: Topic 2
DES, 3DES, AES

Feistel Networks

- Alternative to Substitution Permutation Networks

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



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

Proposition: the function is invertible.

Data Encryption Standard

- Developed in 1970s by IBM (with help from NSA)
- Adopted in 1977 as Federal Information Processing Standard (US)
- Data Encryption Standard (DES): 16-round Feistel Network.
- Key Length: 56 bits
  - Vulnerable to brute-force attacks in modern times
  - 1.5 hours at 14 trillion keys/second (e.g., Antminer S9)

DES Round

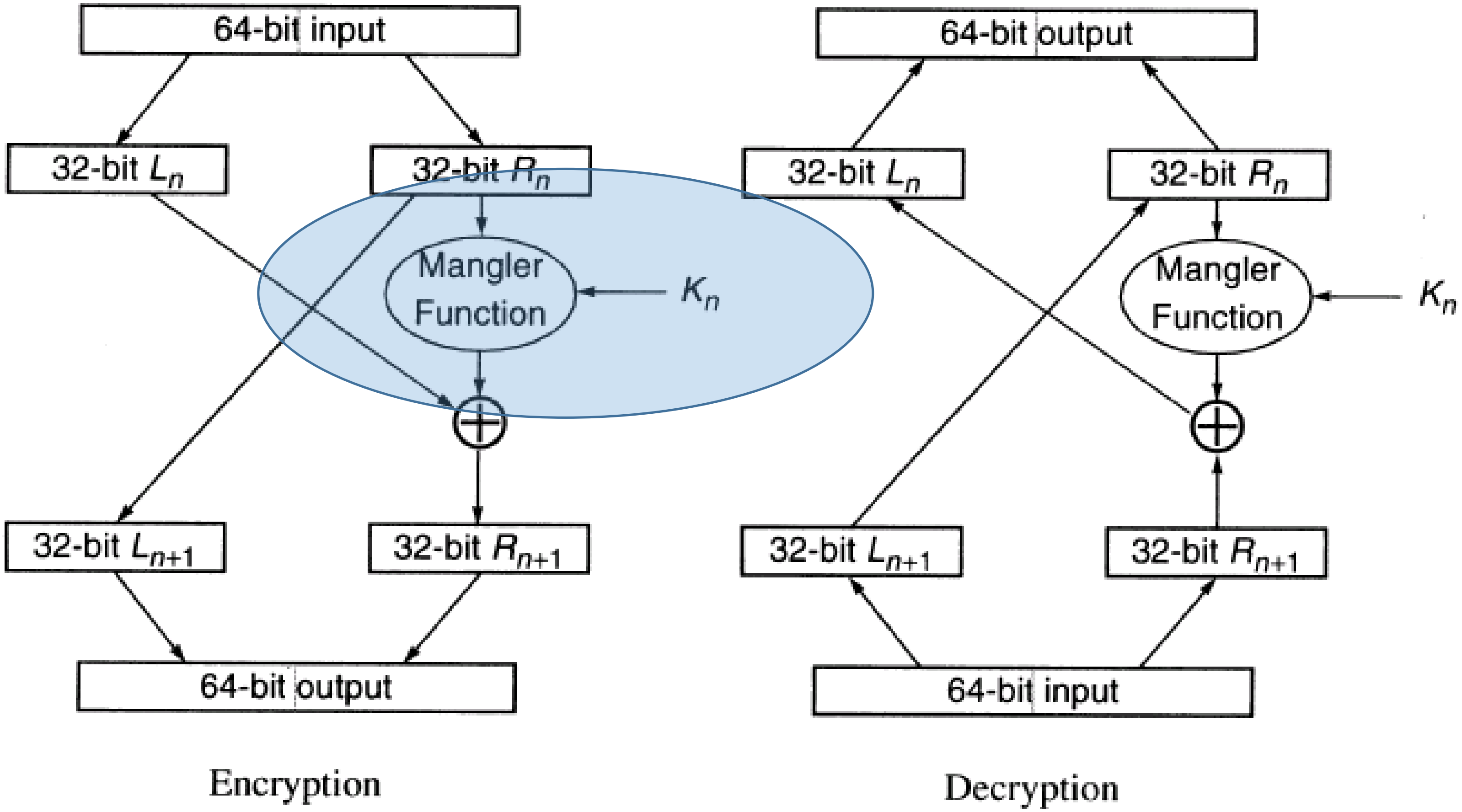
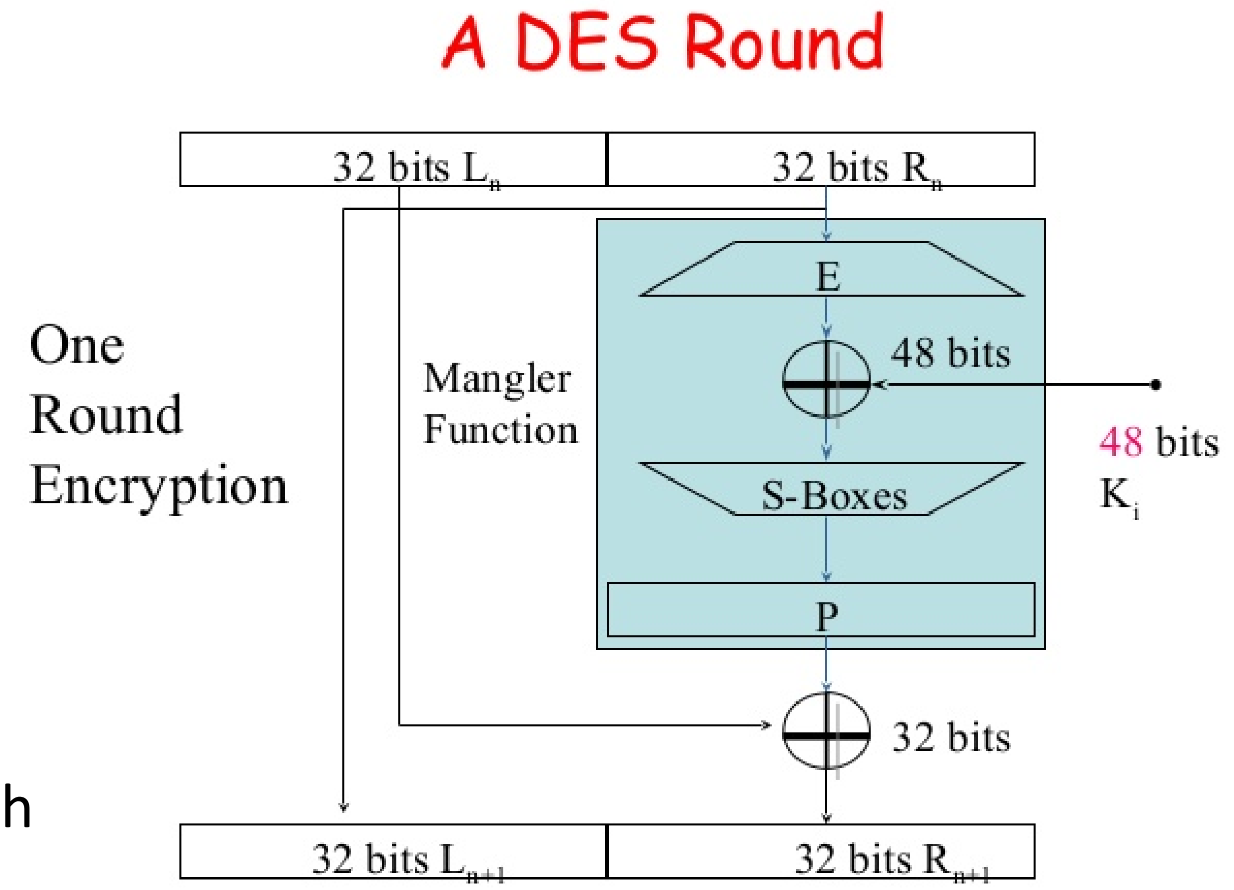


Figure 3-6. DES Round

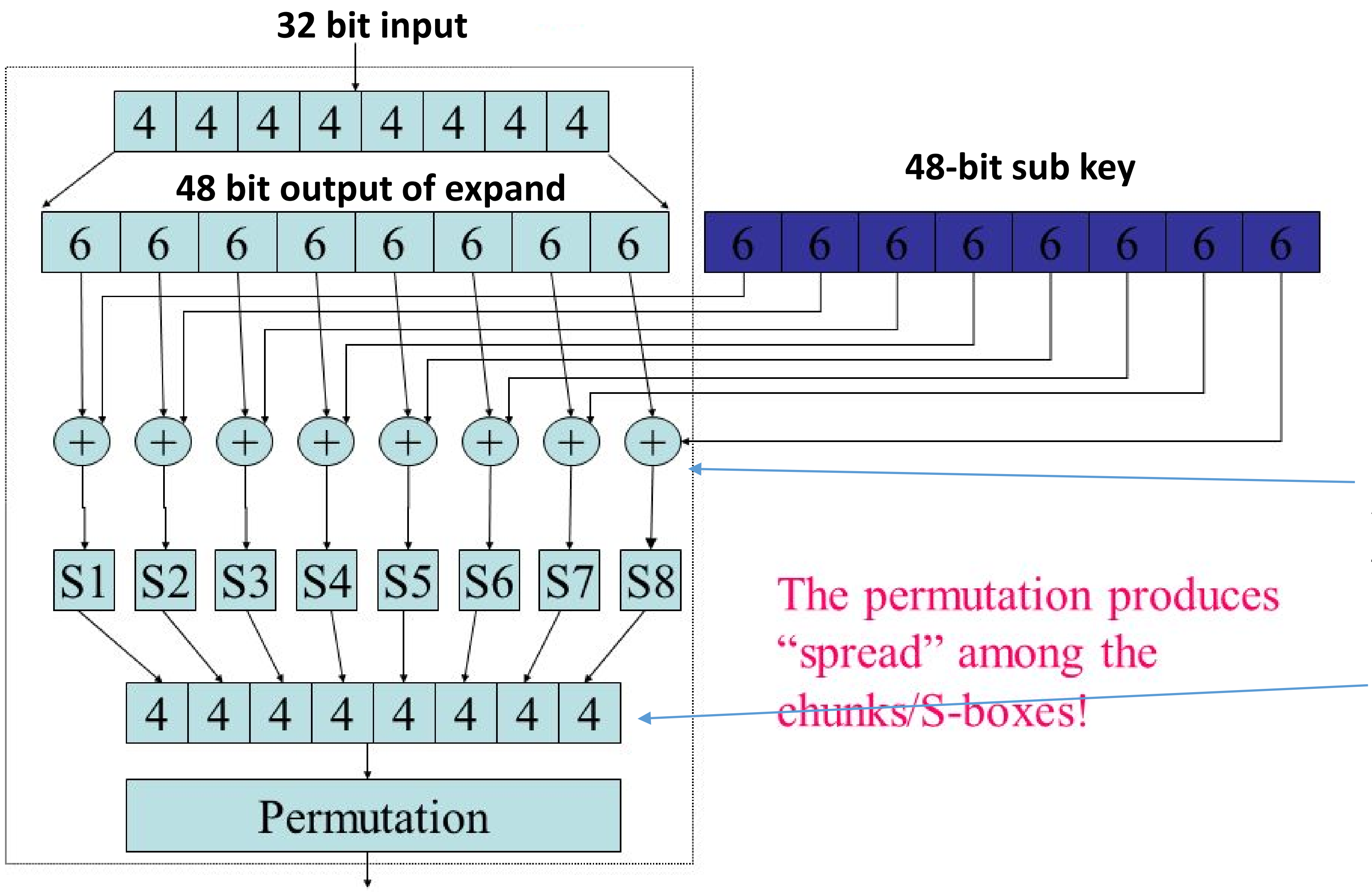
DES Mangle Function

- Expand E: 32-bit input → 48-bit output (duplicates 16 bits)

- S-boxes: $S_1,\ldots,S_8$
  - Input: 6-bits
  - Output: 4 bits
  - Not a permutation!
- 4-to-1 function
  - Exactly four inputs mapped to each possible output



Mangle Function



XOR block before Applying S-Boxes

Each S-box outputs 4 bits

S-Box Representation as Table

4 columns (2 bits)

16 columns (4 bits)

	00	01	10	11
0000				
0001				
0010				
0011				
0100				
0101				
0110				S(x)=1101
….	….	….	….	….
1111				

x =101101 S(x) = Table[0110,11]

S-Box Representation

Each column is permutation

4 columns (2 bits)

16 columns (4 bits)

	00	01	10	11
0000				
0001				
0010				
0011				
0100				
0101				
0110				S(x)=1101
….	….	….	….	….
1111				

$x = 101101$

$S(x) = T[0110, 11]$

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!
- **Requirement**: DES Avalanche Effect!

DES Avalanche Effect

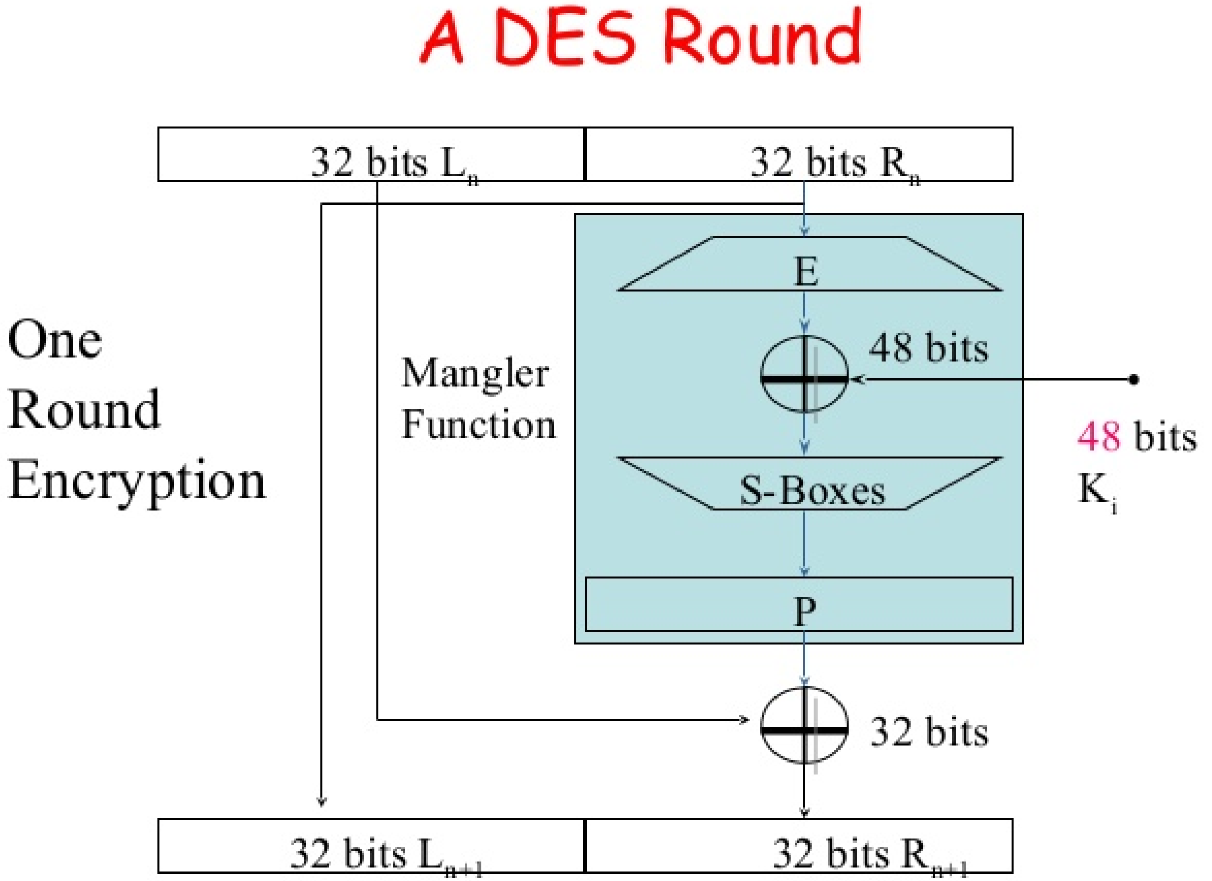
- Permutation the end of the mangle function helps to mix bits

- Special S-box property #1

Let x and x' differ on one bit then $S_i(x)$ differs from $S_i(x')$ on two bits.

Avalanche Effect Example

- Consider two 64 bit inputs
  - (L_n, R_n) and $(L_n', R'_n = R_n)$
  - L_n and L_n' differ on one bit
- This is worst case example
  - $L_{n+1} = L_{n+1}' = R_n$
  - But now R'_{n+1} and R_{n+1} differ on one bit
- Even if we are unlucky $E(R'_{n+1})$ and $E(R_{n+1})$ differ on 1 bit
- → R_{n+2} and R'_{n+2} differ on two bits
- → $L_{n+2} = R'_{n+1}$ and $L_{n+2}' = R'_{n+1}$ differ in one bit



Avalanche Effect Example

- R_{n+2} and R'_{n+2} differ on two bits
- $L_{n+2} = R_{n+1}$ and $L_{n+2}' = R'_{n+1}$ differ in one bit

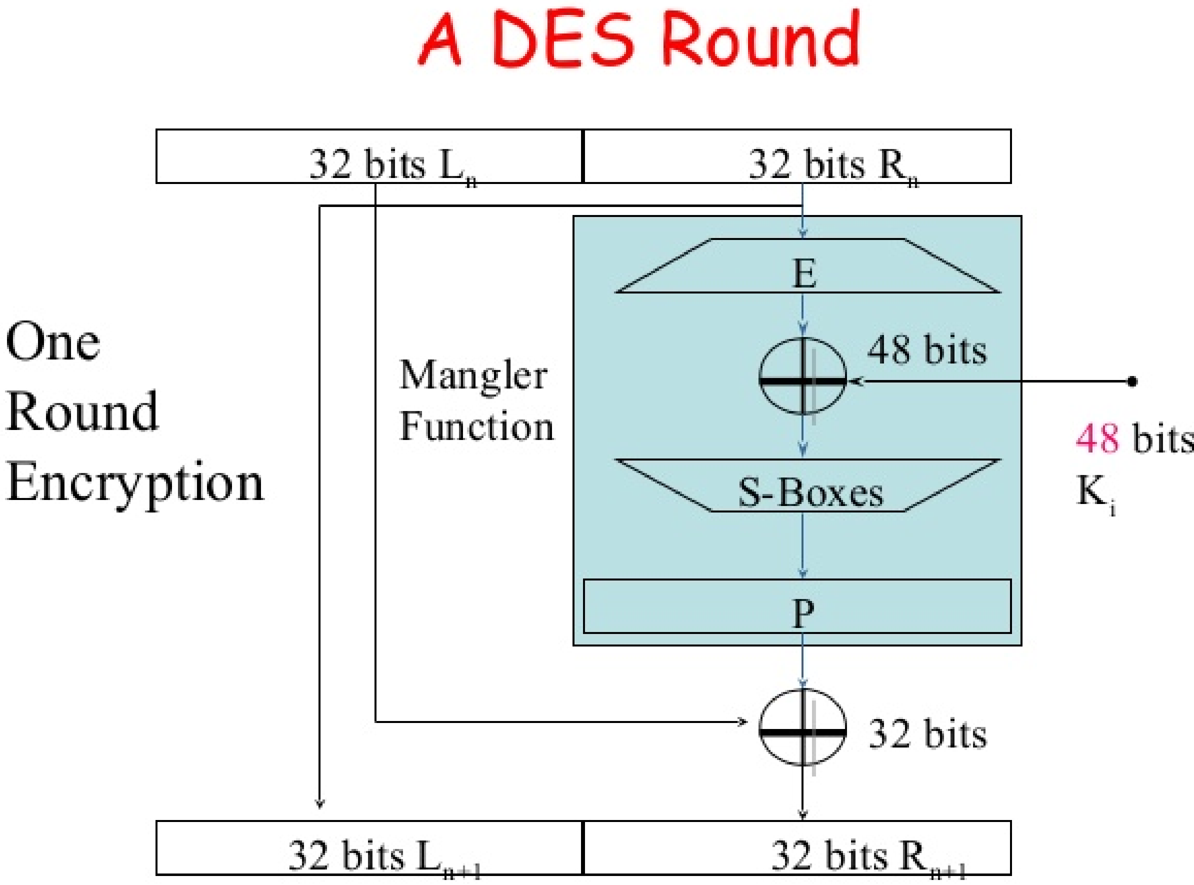
→R_{n+3} and R'_{n+3} differ on four bits since we have different inputs to two of the S-boxes

→$L_{n+3} = R'_{n+2}$ and $L_{n+2}' = R'_{n+2}$ now differ on two bits

- Seven rounds we expect all 32 bits in right half to be "affected" by input change

…

DES has sixteen rounds



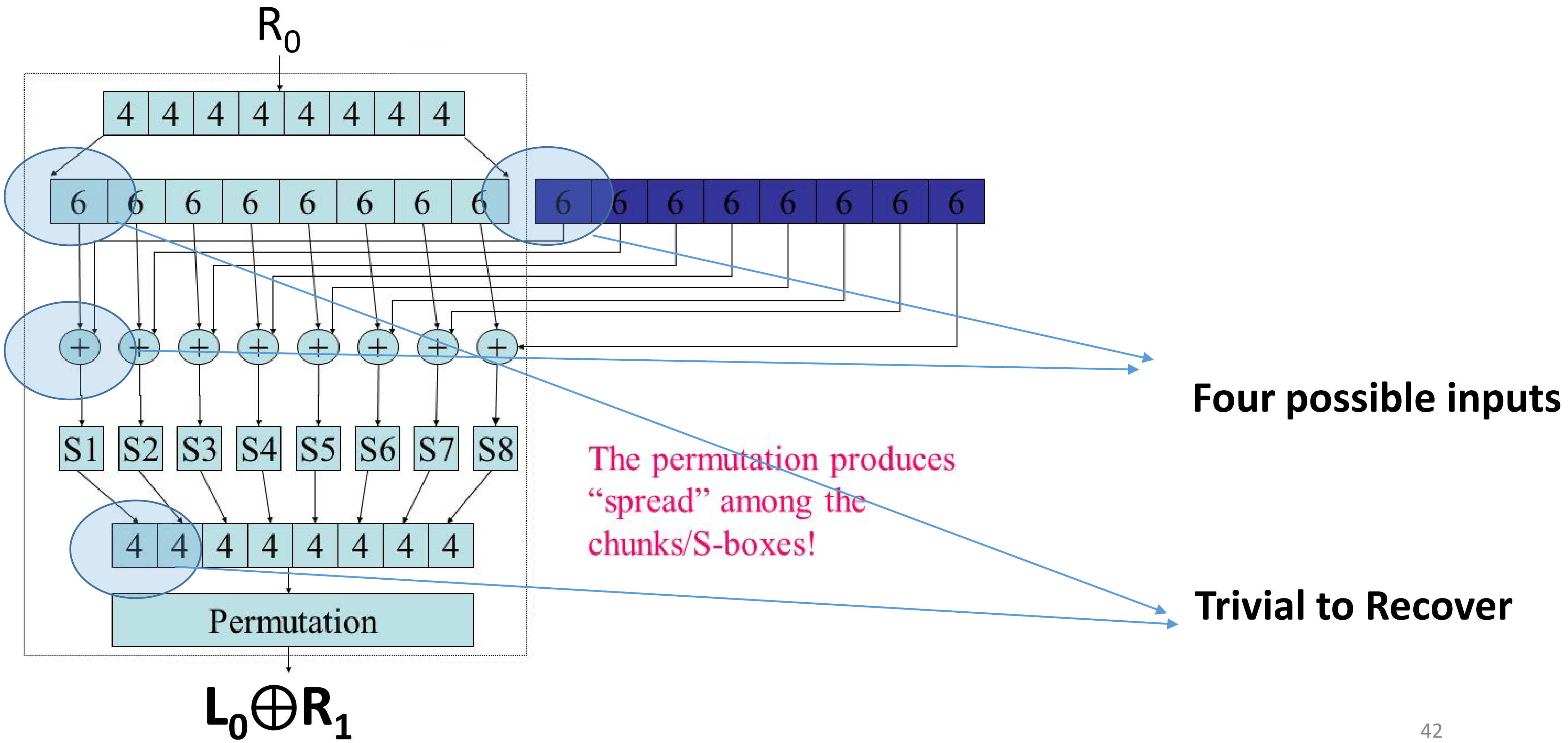
Attack on One-Round DES

- Given input/output pair (x,y)
  - **Output:** y=(L_1,R_1)
  - **Input:** X=(L_0,R_0)
- Note: $R_0=L_1$
- Note: $R_1=L_0 \oplus f_1(R_0)$ where f is the Mangling Function with key k_1

Conclusion:

$$f_1(R_0)=L_0 \oplus R_1$$

Attack on One-Round DES



Attack on Two-Round DES

- Output y =(L_2,R_2)
- Note: R_1=$L_0 \oplus f_1(R_0)$
  - Also,R_1= L_2
  - Thus, $f_1(R_0)$=$L_2 \oplus L_0$
- So we can still attack the first round key k1 as before as R_0 and $L_2 \oplus L_0$ are known
- Note:R_2=$L_1 \oplus f_2(R_1)$
  - Also,L_1=R_0 and R_1= L_2
  - Thus, $f_2(L_2)$=$R_2 \oplus R_0$
- So we can attack the second round key k2 as before as L_2 and $R_2 \oplus R_0$ are known

Attack on Three-Round DES

$$f_1(\mathbf{R_0}) \oplus f_3(\mathbf{R_2}) = (\mathrm{L}_0 \oplus \mathrm{L}_2) \oplus (\mathrm{L}_2 \oplus \mathrm{R}_3)$$
$$= \mathrm{L}_0 \oplus \mathrm{R}_3$$

We **know all of the values** L_0,R_0, R_3 and $\mathrm{L}_3 = \mathrm{R}_2$.

Leads to attack in time $\approx 2^{n/2}$

(See details in textbook)

Remember that DES is 16 rounds

DES Security

- Best Known attack is brute-force 2^{56}
  - Except under unrealistic conditions (e.g., 2^{43} known plaintexts)
- Brute force is not too difficult on modern hardware
- Attack can be accelerated further after precomputation
  - Output is a few terabytes
  - Subsequently keys are cracked in 2^{38} DES evaluations (minutes)
- Precomputation costs amortize over number of DES keys cracked
- Even in 1970 there were objections to the short key length for DES

Double DES

- Let $F_k(x)$ denote the DES block cipher

- A new block cipher F' with a key $k = (k_1, k_2)$ of length 2n can be defined by
$$F'_k(x) = F_{k_2}\left(F_{k_1}(x)\right)$$

- Can you think of an attack better than brute-force?

Meet in the Middle Attack

$$F'_k(x) = F_{k_2}\left(F_{k_1}(x)\right)$$

Goal: Given (x, $\mathrm{c} = F'_k(x)$) try to find secret key k in time and space $\mathrm{O}(n2^n)$.

- **Solution?**
  - **Key Observation**

$$F_{k_1}(x) = F_K^{-1}(\mathrm{c})$$

  - **Compute $F_K^{-1}(\mathrm{c})$ and $F_K(x)$ for each potential key K and store $\left(\boldsymbol{K}, F_K^{-1}(\mathrm{c})\right)$ and $\left(\boldsymbol{K}, F_K(\mathrm{x})\right)$**
  - **Sort each list of pairs (by $F_K^{-1}(\mathrm{c})$ or $F_K(\mathrm{x})$) to find K_1 and K_2.**

Triple DES Variant 1

- Let $F_k(x)$ denote the DES block cipher
- A new block cipher F' with a key $k = (k_1, k_2, k_3\ )$ of length 2n can be defined by

$$F'_k(x) = F_{k_3}\left(F_{k_2}^{-1}\left(F_{k_1}(x)\right)\right)$$

- Meet-in-the-Middle Attack Requires time $\Omega(2^{2n})$ and space $\Omega(2^{2n})$

Triple DES Variant 1

Allows backward compatibility with DES by setting k_1=k_2=k_3

- Let $F_k(x)$ denote the DES block cipher

- A new block cipher F' with a key $k = (k_1, k_2, k_3)$ of length 2n can be defined by

$$F'_k(x) = F_{k_3}\left(F^{-1}_{k_2}\left(F_{k_1}(x)\right)\right)$$

- Meet-in-the-Middle Attack Requires time $\Omega(2^{2n})$ and space $\Omega(2^{2n})$

Triple DES Variant 2

Just two keys!

- Let $F_k(x)$ denote the DES block cipher
- A new block cipher F' with a key $k = (k_1, k_2)$ of length 2n can be defined by

$$F'_k(x) = F_{k_1}\left(F_{k_2}^{-1}\left(F_{k_1}(x)\right)\right)$$

- Meet-in-the-Middle Attack still requires time $\Omega(2^{2n})$ and space $\Omega(2^{2n})$
- Key length is still just 112 bits (128 bits is recommended)

Triple DES Variant 1

$$F'_k(x) = F_{k_3}\left(F_{k_2}^{-1}\left(F_{k_1}(x)\right)\right)$$

- Standardized in 1999

- Still widely used, but it is relatively slow (three block cipher operations)
  - Now viewed as ``weak cipher” by OpenSSL

- Current gold standard: AES

Advanced Encryption Standard (AES)

- (1997) US National Institute of Standards and Technology (NIST) announces competition for new block cipher to replace DES

- Fifteen algorithms were submitted from all over the world
  - Analyzed by NIST
- Contestants given a chance to break competitors schemes

- October, 2000 NIST announces a winner Rijndael
  - Vincent Rijmen and Joan Daemen
  - No serious vulnerabilities found in four other finalists
  - Rijndael was selected for efficiency, hardware performance, flexibility etc…

Advanced Encryption Standard

- **Block Size:** 128 bits (viewed as 4x4 byte array)
- **Key Size:** 128, 192 or 256

- Essentially a Substitution Permutation Network
  - **AddRoundKey:** Generate 128-bit sub-key from master key XOR with current state
  - **SubBytes:** Each byte of state array (16 bytes) is replaced by another byte according a a single S-box (lookup table)
  - **ShiftRows –** shift ith row by i bytes
  - **MixColumns –** permute the bits in each column

Substitution Permutation Networks

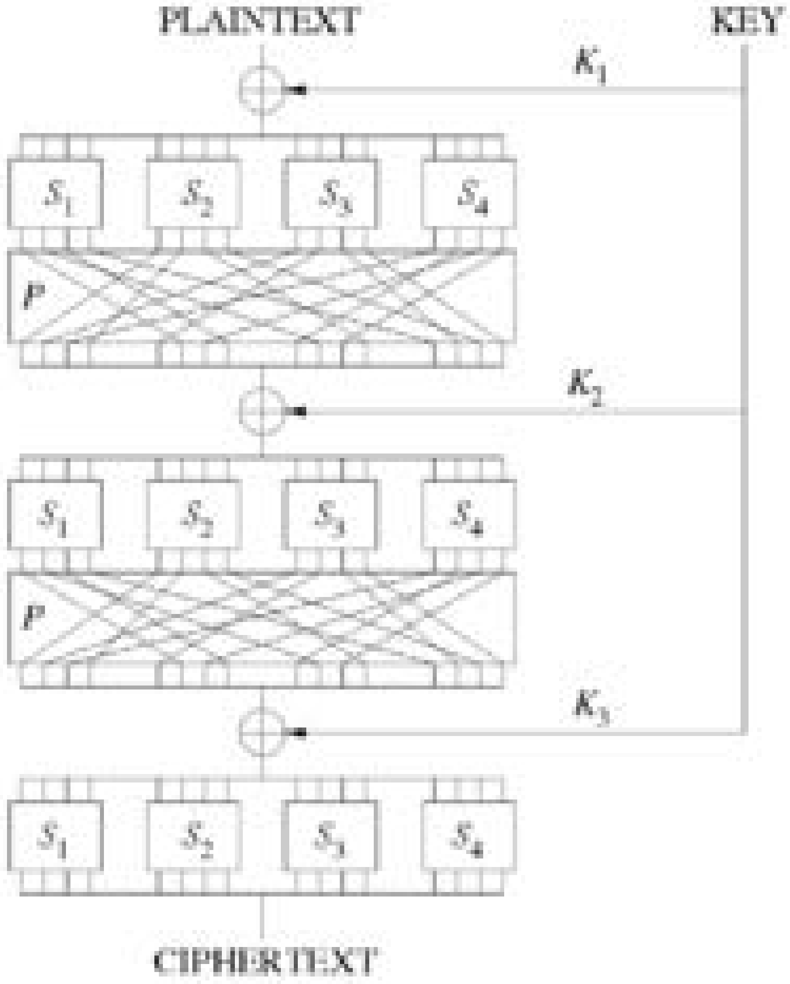
- S-box a public "substitution function" (e.g. $S \in \mathbf{Perm_8}$).
- S is not part of a secret key, but can be used with one
$$\mathrm{f(x)} = S(\mathrm{x} \oplus k)$$

Input to round: x, k (k is subkey for current round)

1. **Key Mixing**: Set $\mathrm{x} \coloneqq \mathrm{x} \oplus k$
2. **Substitution**: $\mathrm{x} \coloneqq \mathrm{S_1(x_1) \parallel S_2(x_2) \parallel \cdots \parallel S_8(x_8)}$
3. **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)).

Advanced Encryption Standard

- Block Size: 128 bits
- Key Size: 128, 192 or 256

- Essentially a Substitution Permutation Network
  - **AddRoundKey:** Generate 128-bit sub-key from master key, XOR with current state array
  - **SubBytes:** Each byte of state array (16 bytes) is replaced by another byte according a single S-box (lookup table)
  - **ShiftRows**
  - **MixColumns**

Key Mixing

Permutation

Substitution

AddRoundKey:

Round Key (16 Bytes)

00001111			
10100011	…		
11001100		…	
01111111			…

$\oplus$

State

11110000			
01100010	…		
00110000		…	
11111111			…

=

11111111			
11000001	…		
11111100		…	
10000000			…

AddRoundKey:

Round Key (16 Bytes)

10100011	…		
		…	
			…

State

11111111			
11000001	…		
11111100		…	
10000000			…

SubBytes (Apply S-box)

S(11111111)			
S(11000001)	S(…)		
S(11111100)		S(…)	
S(10000000)			S(…)

AddRoundKey:

Round Key (16 Bytes)

10100011	…		
		…	
			…

State

S(11111111)			
S(11000001)	S(…)		
S(11111100)		S(…)	
S(10000000)			S(…)

Shift Rows

S(11111111)			
	S(11000001)	S(…)	
S(…)		**S(11111100)**	
		S(…)	**S(10000000)**

AddRoundKey:

Round Key (16 Bytes)

10100011	…		
		…	
			…

State

S(11111111)			
	S(11000001)	S(…)	
S(…)		**S(11111100)**	
		S(…)	**S(10000000)**

Mix Columns

Invertible (linear) transformation.

Key property: if inputs differ in b>0 bytes then output differs in 5-b bytes (minimum)

AES

- We just described one round of the SPN

- AES uses
  - 10 rounds (with 128 bit key)
  - 12 rounds (with 192 bit key)
  - 14 rounds (with 256 bit key)

AES Attacks?

- Side channel attacks affect a few specific implementations
  - But, this is not a weakness of AES itself
  - Timing attack on OpenSSL's implementation AES encryption (2005, Bernstein)

- (2009) Attack on 11 round version of AES
  - recovers 256-bit key in time 2^{70}
  - But AES is 14 round (with 256 bit key) so the attack doesn't apply in practice
- (2009) Attack on 192-bit and 256 bit version of AES
  - recovers 256-bit key in time $2^{99.5}$.

- First public cipher approved by NSA for Top Secret information