Cryptography
CS 555

Week 6:

- Commitment Schemes
- Ideal Cipher Model + Hash Functions from Block Ciphers
- Block Ciphers
- Feistel Networks
- DES, 3DES

Readings: Katz and Lindell Chapter 6-6.2.4

Fall 2021

Recap

- Hash Functions
  - Definition
  - Merkle-Damgard
  - Merkle Trees
- HMAC construction
- Generic Attacks on Hash Function
  - Birthday Attack
  - Small Space Birthday Attacks (cycle detection)
- Pre-Computation Attacks: Time/Space Tradeoffs
- Random Oracle Model

Commitment Schemes

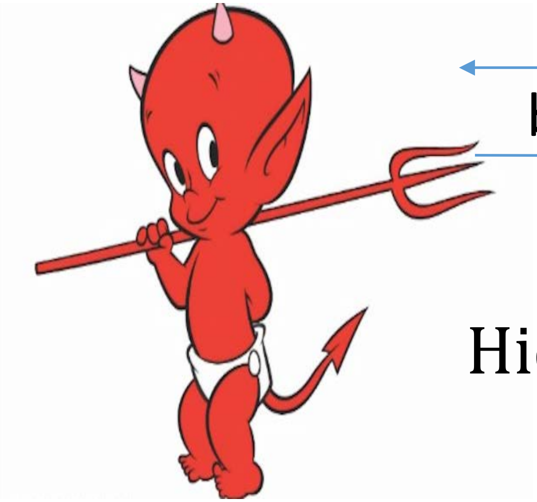
- Alice wants to commit a message m to Bob
  - And possibly reveal it later at a time of her choosing
- Properties
  - Hiding: commitment reveals nothing about m to Bob
  - Binding: it is infeasible for Alice to alter message

Syntax Commitment Scheme with Canonical Verification:

- $\mathbf{c} \coloneqq \mathbf{Commit}(\mathrm{m};\mathrm{r})$: takes as input a message m and random bits r and outputs a commitment **c** to the message m
- $\mathbf{CannonicalVerify}(\mathrm{c},\mathrm{m},\mathrm{r}) \coloneqq \begin{cases} 1 & if\, c == \mathbf{Commit}(\mathrm{m};\mathrm{r}) \\ 0 & otherwise \end{cases}$

- **Note:** Not all commitment schemes use canonical verification, but this definition suffices for our purposes. In this case there may be a third algorithm pp:=Setup(1^n) which generates public parameters for the commitment scheme.

Commitment Hiding $(\text{Hiding}_{A,Com}(n))$



m_0,m_1

commit(r,m_b)

b'

$$\text{Hiding}_{A,Com}(n) = \begin{cases} 1 & \text{if } b = b' \\ 0 & otherwise \end{cases}$$

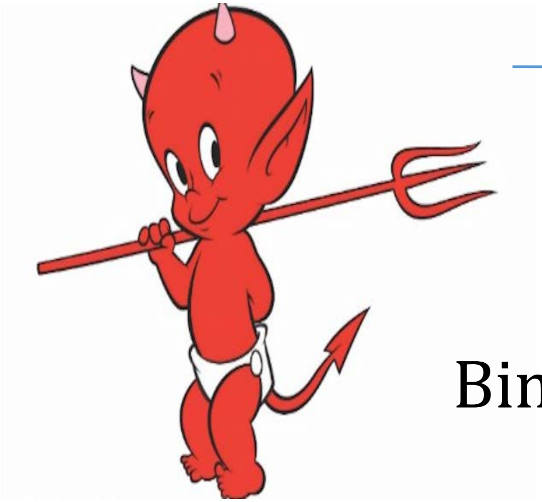
r = Gen(.)

Bit b

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}$$

$$\Pr[\text{Hiding}_{A,Com}(n) = 1] \leq \frac{1}{2} + \mu(n)$$

Commitment Binding ($\mathrm{Binding}_{A,Com}(n)$)



r_0, r_1, m_0, m_1

$$\mathrm{Binding}_{A,Com}(n) = \begin{cases} 1 & \text{if } \mathbf{commit(r_0,m_0) = commit(r_1,m_1)} \\ 0 & otherwise \end{cases}$$

$$\forall PPT\ A\ \exists \mu\ \text{(negligible) s.t}$$
$$\Pr\left[\mathrm{Binding}_{A,Com}(n) = 1\right] \leq \mu(n)$$

Secure Commitment Scheme

- **Definition:** A secure commitment scheme is **hiding** and **binding**
- **Hiding**

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$
$$\Pr\left[\text{Hiding}_{A,Com}(n) = 1\right] \leq \frac{1}{2} + \mu(n)$$

- **Binding**

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$
$$\Pr\left[\text{Binding}_{A,Com}(n) = 1\right] \leq \mu(n)$$

Commitment Scheme in Random Oracle Model

- $\mathbf{Commit}(\mathrm{r}, \mathrm{m}) \coloneqq \mathrm{H}(r \parallel m)$

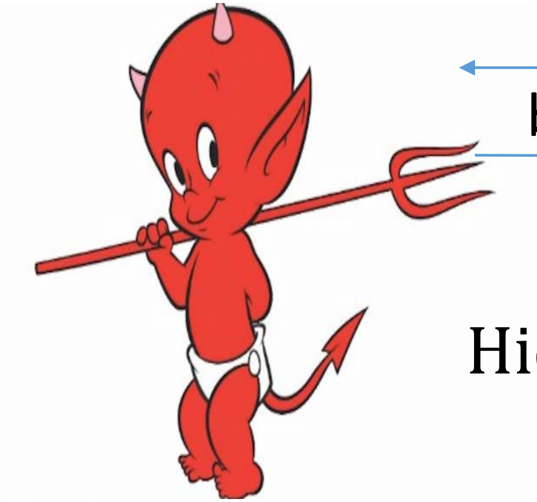
- $\mathbf{Reveal}(\mathrm{c}) \coloneqq (\mathrm{r}, \mathrm{m})$

Theorem: In the random oracle model this is a secure commitment scheme.

Binding:

$\mathbf{commit}(\mathbf{r_0},\mathbf{m_0}) = \mathbf{commit}(\mathbf{r_1},\mathbf{m_1}) \leftrightarrow \mathrm{H}(\mathbf{r_0} \parallel \mathbf{m_0}) = \mathrm{H}(\mathbf{r_1} \parallel \mathbf{m_1})$

Commitment Hiding $(\text{Hiding}_{A,Com}(n))$



m_0, m_1 →

← $\mathrm{H}(r \parallel m_b)$

b' →

$$\text{Hiding}_{A,Com}(n) = \begin{cases} 1 & if\, b = b' \\ 0 & otherwise \end{cases}$$

r = Gen(.)

Bit b

$$\forall \cancel{PPT}\ A\ making\ q(n)\ queries\ \text{s.t}$$

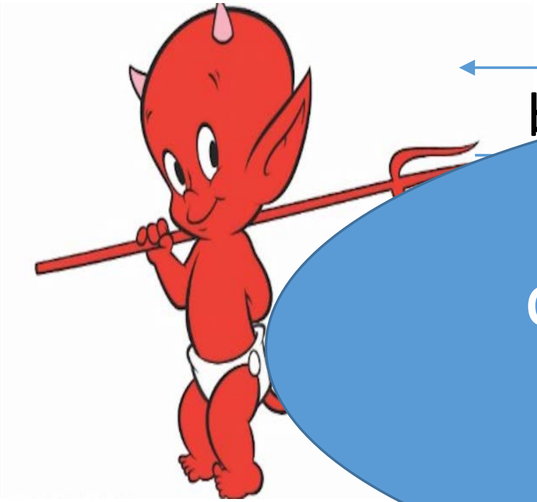
$$\Pr\left[\text{Hiding}_{A,Com}(n) = 1\right] \leq \frac{1}{2} + \frac{q(n)}{2^{|r|}}$$

Commitment Hiding $\left(\mathrm{Hiding}_{A,Com}(n)\right)$

m_0, m_1

$\mathrm{H}(r \parallel m_b)$

b’



If attacker never makes query of the form $\mathrm{H}(r \parallel x)$ then bit b is information theoretically hidden

r = Gen(.)

Bit b

$\forall \cancel{PPT}\ A\ making\ q(n)\ quer\ldots$ s.t

$$\Pr\left[\mathrm{Hiding}_{A,Com}(n) = 1\right] \le \frac{1}{2} + \frac{q(n)}{2^{|r|}}$$

Ideal Cipher Model

- For each n-bit string K we pick a truly random permutation F_K
- Public Oracles
  - $O(K, x) = F_K(x)$
  - $O^{-1}(K, y) = F_K^{-1}\ (x)$

- Real World: Instantiate Ideal Cipher with a modern block cipher like AES

- Similar Pros/Cons to Random Oracle Model
  - Pro: Powerful evidence of sound design
  - Con: No blockcipher is an ideal cipher (even AES)

Hash Functions from Ideal Block Ciphers

- Davies-Meyer Construction from block cipher F_K

$$H(K, x) = F_K(x) \oplus x$$

Theorem: If $F: \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ is modeled as an ideal block cipher then Davies-Meyer construction is a collision-resistant hash function

(**Concrete:** Need roughly $\mathrm{q} \approx 2^{\lambda/2}$ queries to find collision)

- **Ideal Cipher Model:** For each key K model F_K as a truly random permutation which may only be accessed in black box manner.
    - (Equivalent to Random Oracle Model)

Hash Functions from Block Ciphers

$$H(K,x) = F_K(x) \oplus x$$

Analysis: Suppose we have already made queries to the ideal cipher

- $(K_1, x_1), \ldots, (K_q, x_q)$ to F_K to get $F_{K_1}(x_1), \ldots, F_{K_q}(x_q)$ and queries
- $(K_{q+1}, y_1), \ldots, (K_{2q}, y_q)$ to $F_K^{-1}(.)$ to get $x_{q+1} \coloneqq F_{K_{q+1}}^{-1}(y_1), \ldots, x_{2q} \coloneqq F_{K_{2q}}^{-1}(y_q)$.

$H(K_i, x_i)$ is known for all $\mathrm{i} \leq 2q$ (but $H(K,x)$ is unknown at other points.

Now suppose we make a new query $(K,x) \notin \{(K_1, x_1), \ldots, (K_{2q}, x_{2q})\}$: $F_K(x)$ sampled uniformly from $2^\lambda - 2q$ possible choices.

➔ Collides with $\mathrm{H}(K_i, x_i)$ with probability at most $\frac{1}{2^\lambda - 2q}$

➔ Collides with $\mathrm{H}(K_{q+i}, x_{q+i})$ with probability at most $\frac{1}{2^\lambda - 2q}$

➔ $\mathrm{H}(K,x)$ Collides with prior query with probability at most $\frac{2\mathrm{q}}{2^\lambda - 2q}$

Hash Functions from Block Ciphers

$$H(K,x) = F_K(x) \oplus x$$

Analysis:

Fact 1: Query q+1 to ideal cipher yields collision (with prior query) with probability at most $\frac{q}{2^\lambda - q}$

Fact 2: The probability of finding a collision within q queries is at most $\sum_{i \leq q} \frac{i}{2^\lambda - i} \leq \frac{q(q-1)/2}{2^\lambda - q}$

A Broken Attempt

$$H(K_1, K_2, x_1, x_2) = F_{K_1}(x_1) \oplus F_{K_2}(x_2) \oplus K_1 \oplus K_2$$

Collision Attack: Pick arbitrary keys $K_0 \neq K_1$

Step 1: Query $\mathrm{x}_1 \coloneqq F_{K_0}^{-1}(0^n)$ and $\mathrm{x}_2 \coloneqq F_{K_0}^{-1}(1^n)$

Step 2: Query $\mathrm{w}_1 \coloneqq F_{K_1}^{-1}(0^n)$ and $\mathrm{w}_2 \coloneqq F_{K_1}^{-1}(1^n)$

$$H(K_0, K_0, x_1, x_2) = F_{K_0}(x_1) \oplus F_{K_0}(x_2) \oplus K_0 \oplus K_0 = 0^n \oplus 1^n$$

$$= F_{K_1}(w_1) \oplus F_{K_1}(w_2) = H(K_1, K_1, x_1, x_2)$$

Exploits the fact that we can query inverse oracle F_K^{-1}

CS 555: Week 6: Topic 1
Block Ciphers

An Existential Crisis?

- We have used primitives like PRGs, PRFs to build secure MACs, CCA-Secure Encryption, Authenticated Encryption etc…

- Do such primitives exist in practice?

- How do we build them?

Recap

- Hash Functions/PRGs/PRFs, CCA-Secure Encryption, MACs

Goals for This Week:

- Practical Constructions of Symmetric Key Primitives

Today's Goals: Block Ciphers

- Sbox
- Confusion Diffusion Paradigm
- Feistel Networks

Pseudorandom Permutation

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

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

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 ∈$\mathbf{Perm_n}$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?

- **$|Perm_n|$=?**
- **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 $f_1,\ldots,f_{16} \in \mathbf{Perm_8}$.

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

- 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 $\mathrm{F} \in \mathbf{Perm}_{\mathbf{128}}$
- 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 applying the permutations does 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

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!

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:** $g_1(z_1) \parallel g_2(z_2) \parallel \cdots \parallel g_8(z_8)$

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

Are We Done?

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

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:** $f_1(z_1) \parallel f_2(z_2) \parallel \cdots \parallel f_8(z_8)$

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

Suppose $f_1(x_1) = 00110101 = y_1$ and $f_1(x'_1) = 00110100 = y'_1$

$F_k(x'_1 \parallel x_2 \parallel \cdots \parallel x_8)=$

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:** $f_1(z_1) \parallel f_2(z_2) \parallel \cdots \parallel f_8(z'_8)$

Highly unlikely that a truly random permutation would behave this way!

Substitution Permutation Networks

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

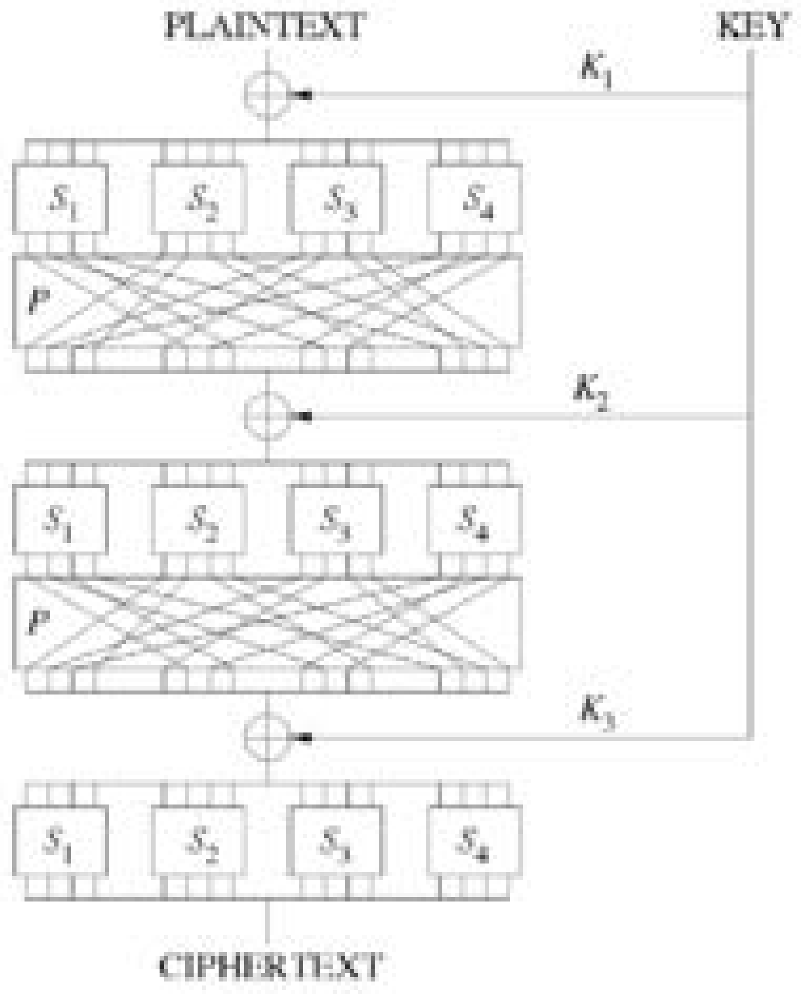
- S is not part of a secret key, but can be used with one

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

- Input to round: x, k (k is subkey for current round)
- **Key Mixing**: Set $x \coloneqq x \oplus k$
- **Substitution**: $x \coloneqq S_1(x_1) \parallel S_2(x_2) \parallel \cdots \parallel S_8(x_8)$
- **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 x on at least 2-bits (for all bytes x) 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
  - $P^{-1}(F_k(x)) = \mathrm{S}_1(\mathrm{x}_1 \oplus k_1) \parallel \mathrm{S}_2(\mathrm{x}_2 \oplus k_2) \parallel \cdots \parallel \mathrm{S}_8(\mathrm{x}_8 \oplus k_8)$
  - $x_i \otimes k_i = S_i^{-1}\left(S_i(x_i \oplus k_i)\right)$
  - 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**: $\mathrm{z}_1 \parallel \cdots \parallel \mathrm{z}_8 = \mathrm{P}(\mathrm{y})$
- **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 $\mathrm{x} \coloneqq \mathrm{x} \bigoplus k_1$
- **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 $\mathrm{z} \bigoplus k_2$

- 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 $\mathrm{y_i} = \mathrm{S_i}(\mathrm{x_i} \otimes k_\mathrm{i})$
  - 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 $8x2^8$

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

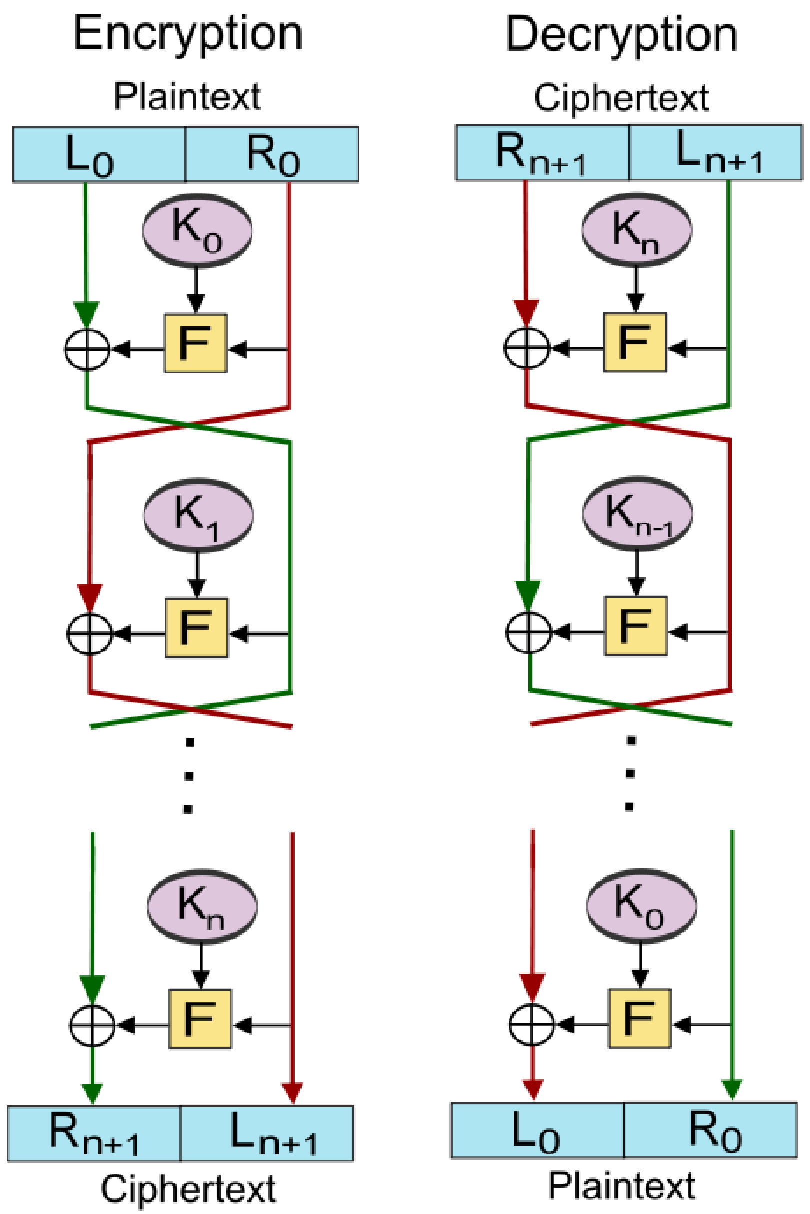
- Harder Case: Two round SPN

- Exercise ☺

- **Ideal Cipher Model:** For each key K model F_K as a truly random permutation which may only be accessed in black box manner.
  - Attacker may submit query (K,x,+) and oracle responds with $F_K(x)$ or
  - Stronger than assuming that F is a Pseudorandom Permutation
  - (Equivalent to Random Oracle Model)

Feistel Networks

- Alternative to Substitution Permutation Networks
- **Advantage**: underlying functions need not be invertible, but the result is still a permutation



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

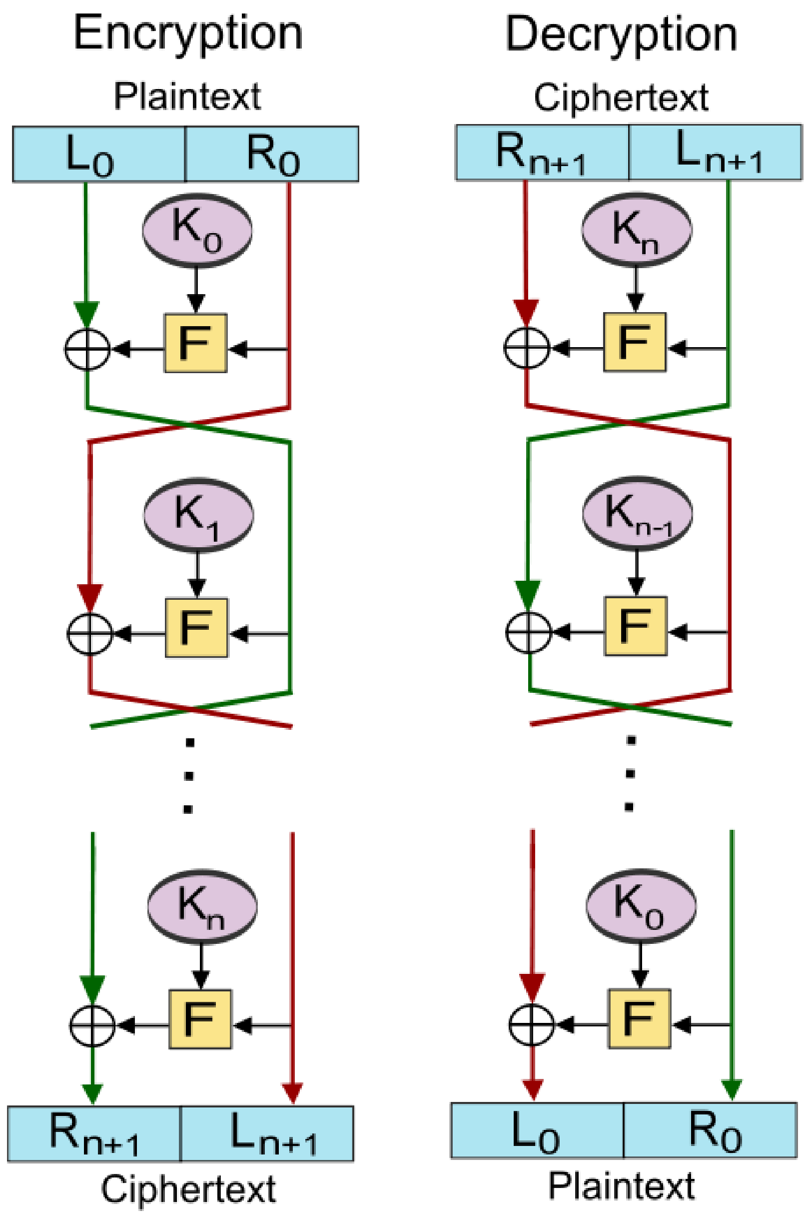
Proposition: the function is invertible.

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

CS 555: Week 6: Topic 2
DES, 3DES

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 DES evals/second e.g., Antminer S9 runs at 14 TH/s

DES Round

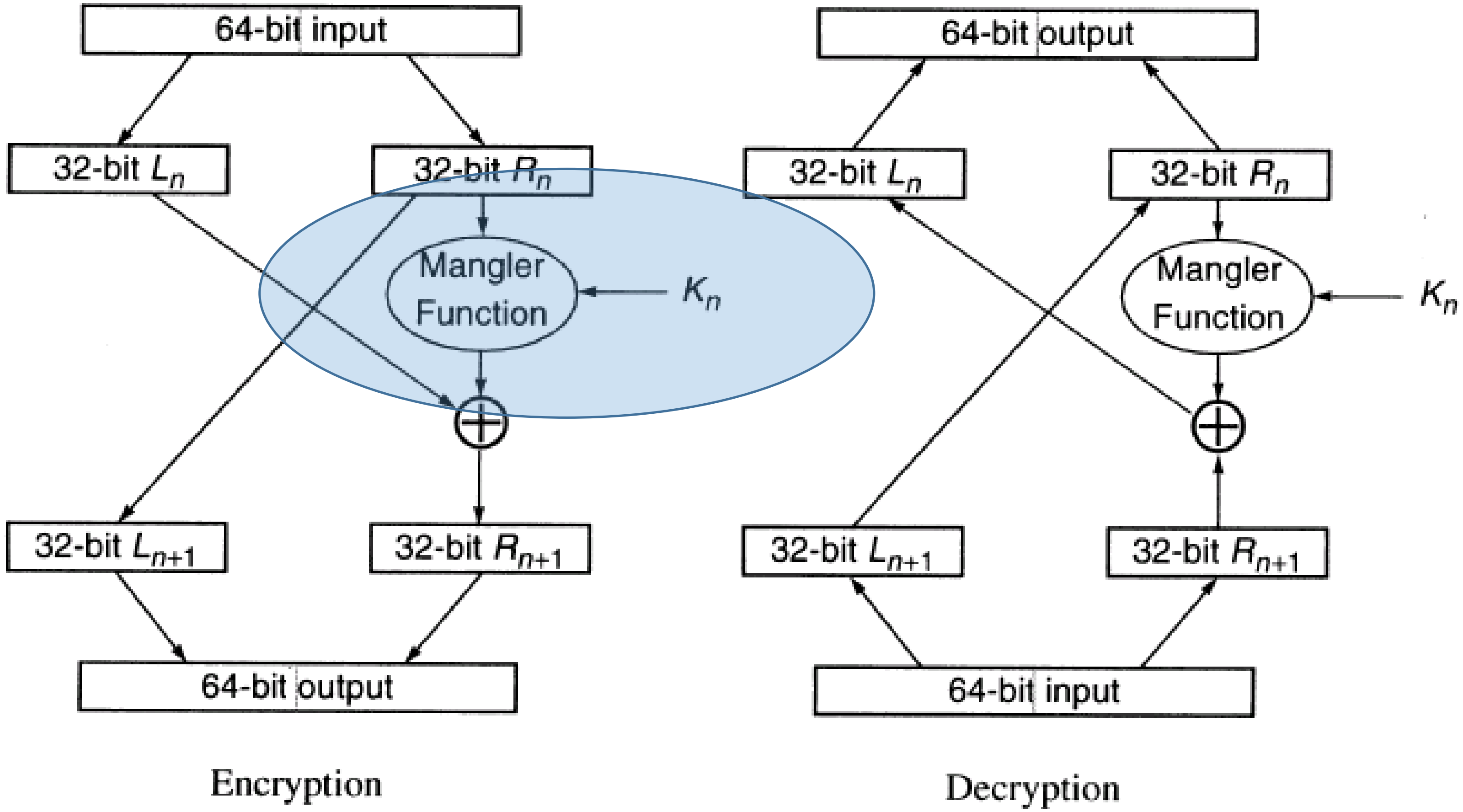


Figure 3-6. DES Round

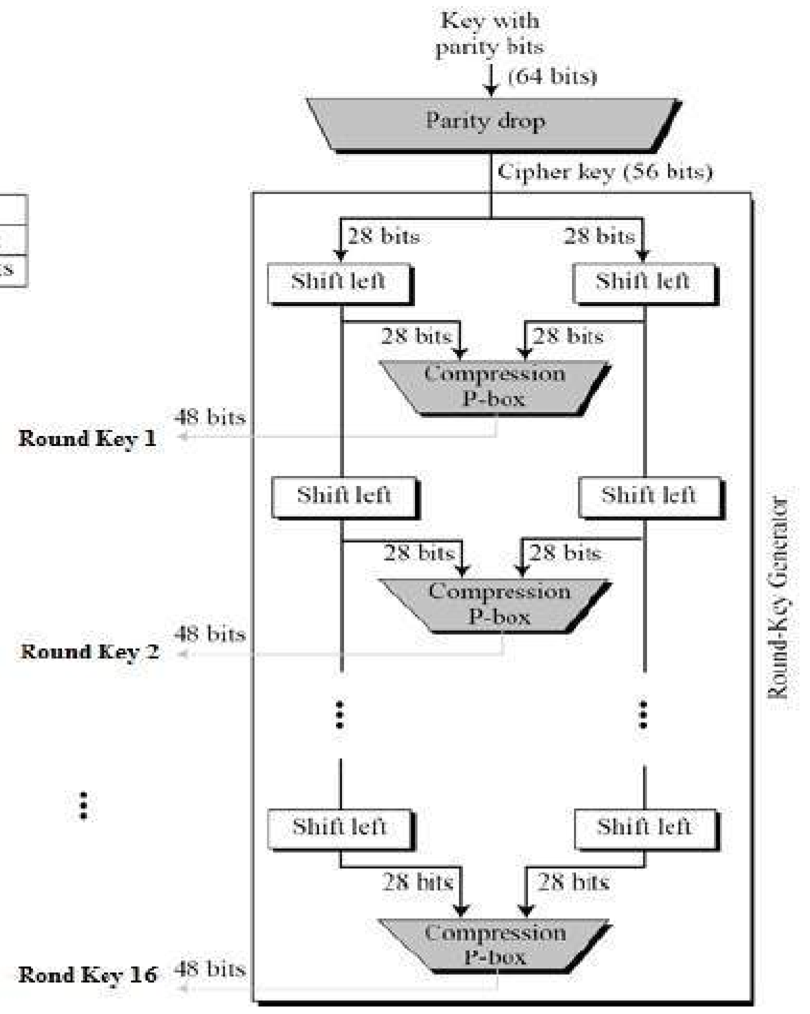
Generating the Round Keys

- **Initial Key:** 64 bits
- **Effective Key Length:** 56 bits
- **Round Key Length:** 48 bits (each)

- **16 round keys** derived from initial key

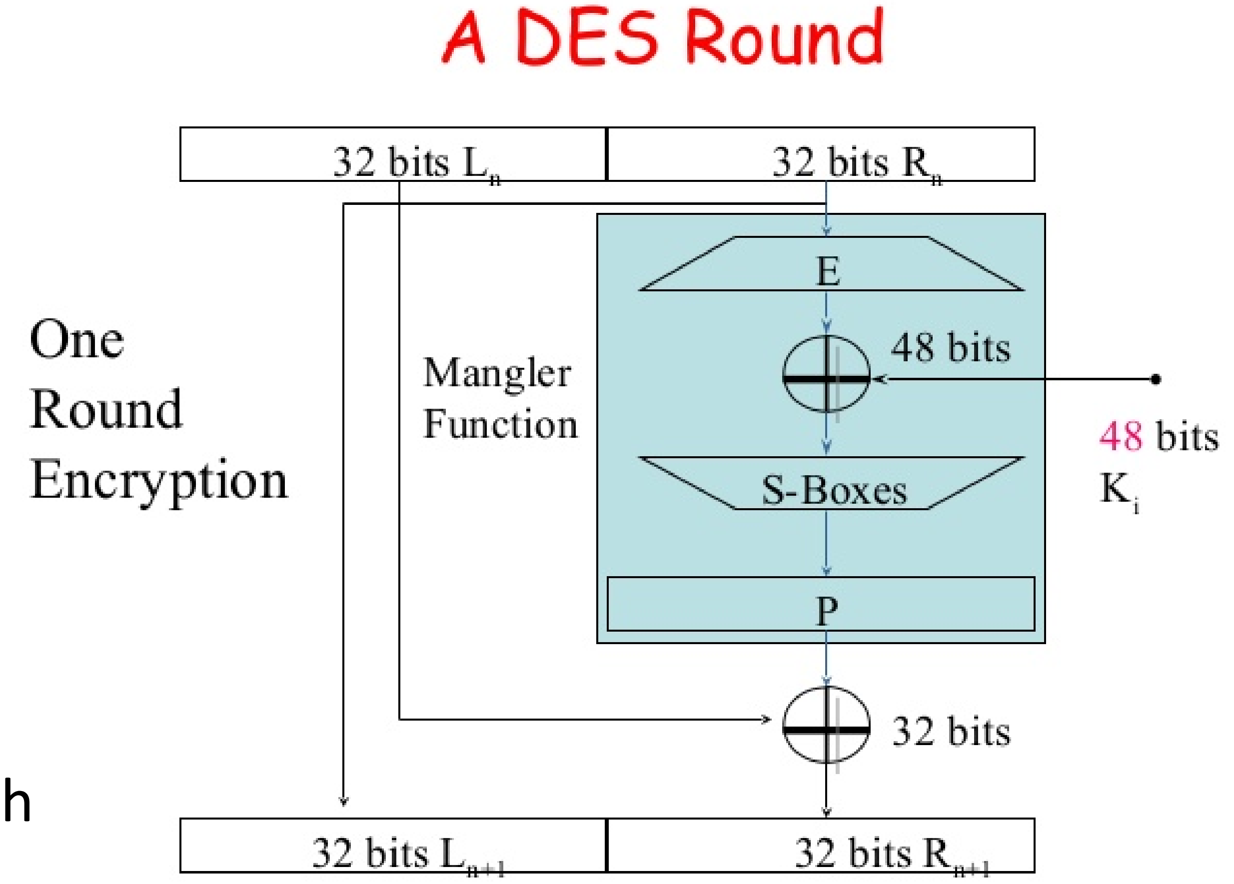
Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

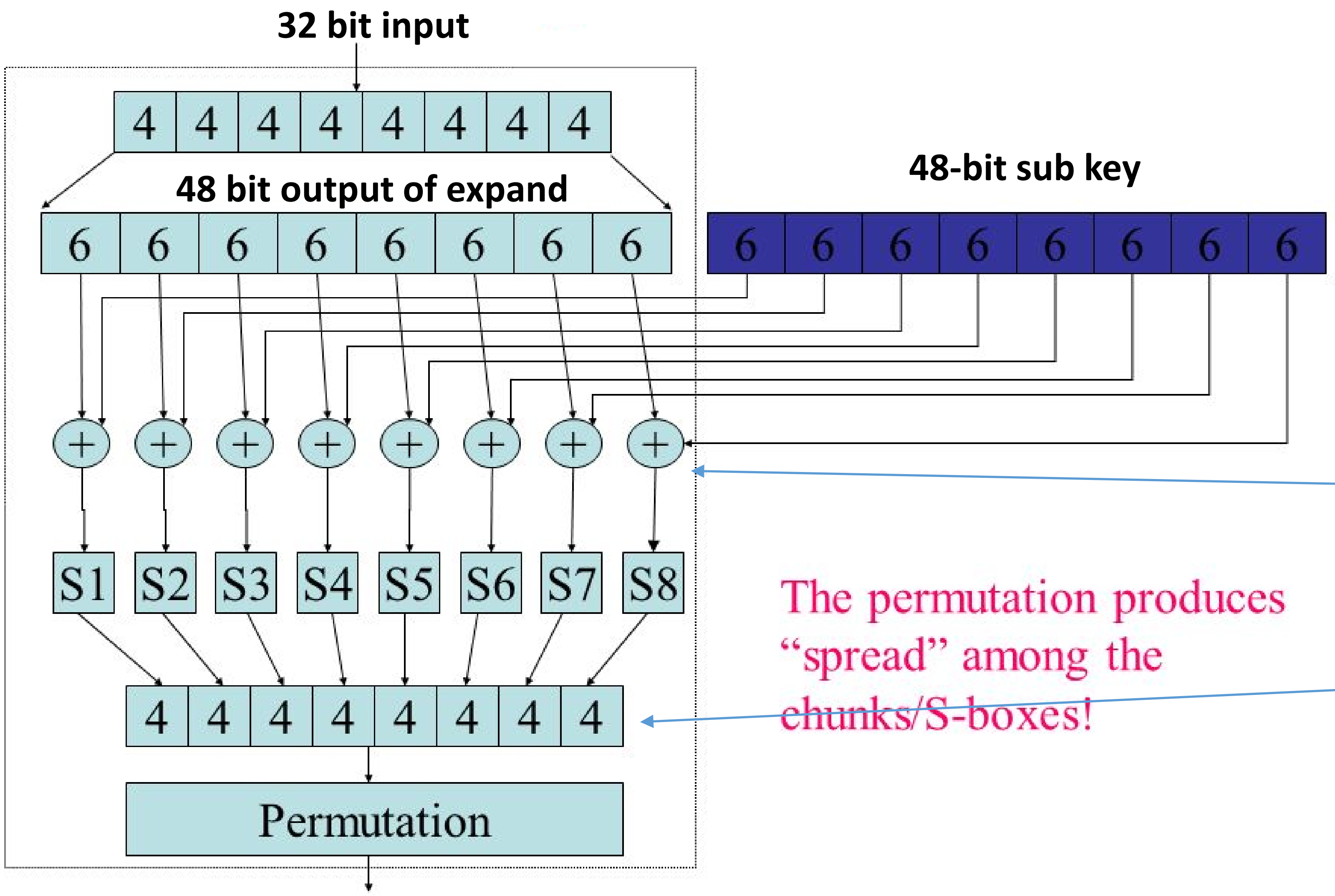


DES Mangle Function

- 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

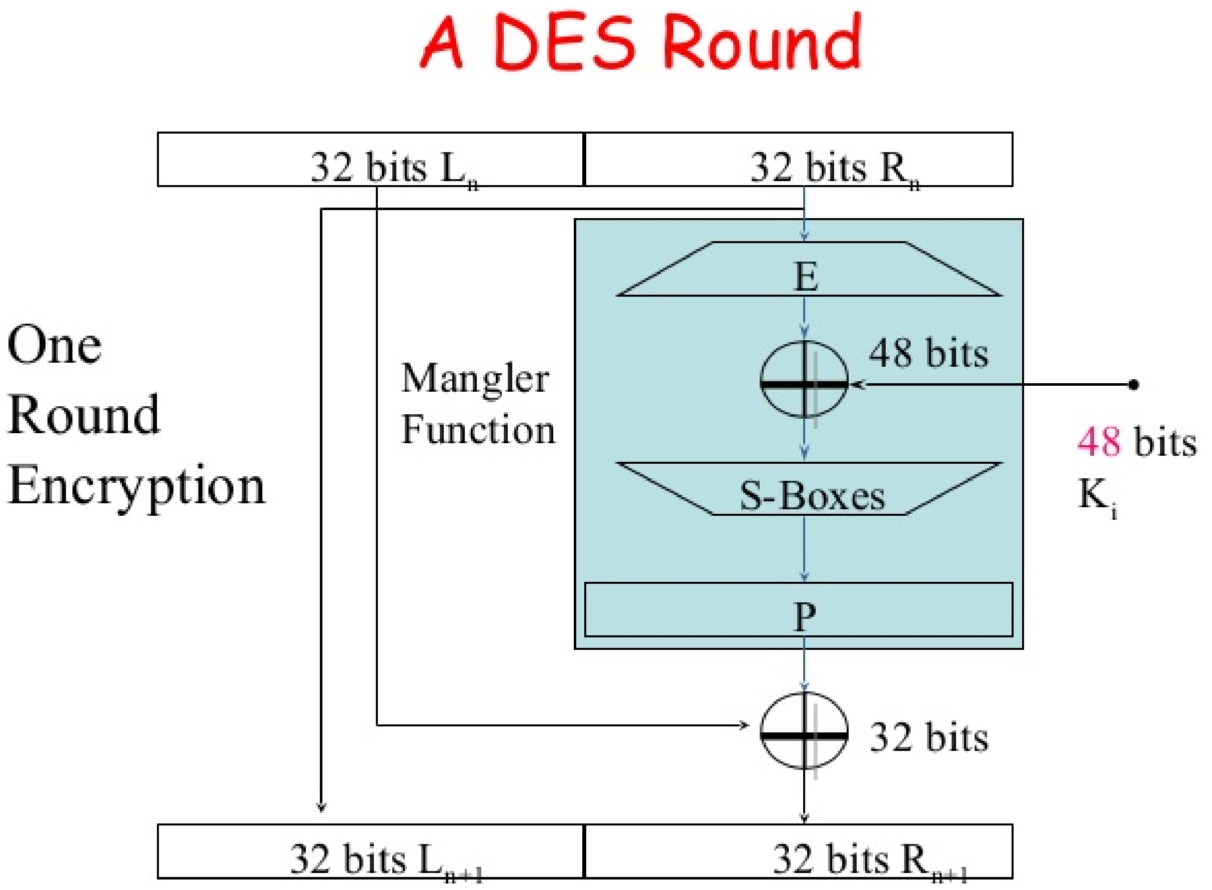
- Permutation the end of the mangle function helps to mix bits

- 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

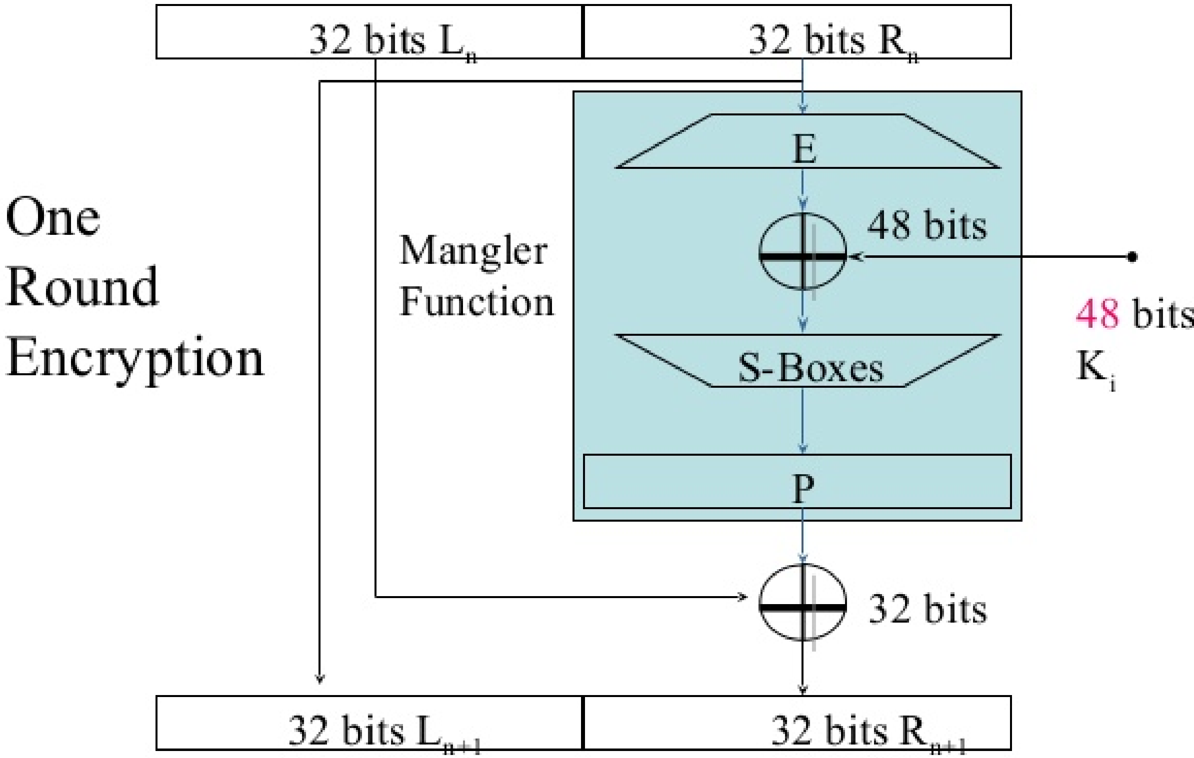
→$L_{n+3} = R'_{n+2}$ and $L_{n+2}' = R'_{n+2}$ now differ on two bits

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

…

DES has sixteen rounds

A DES Round



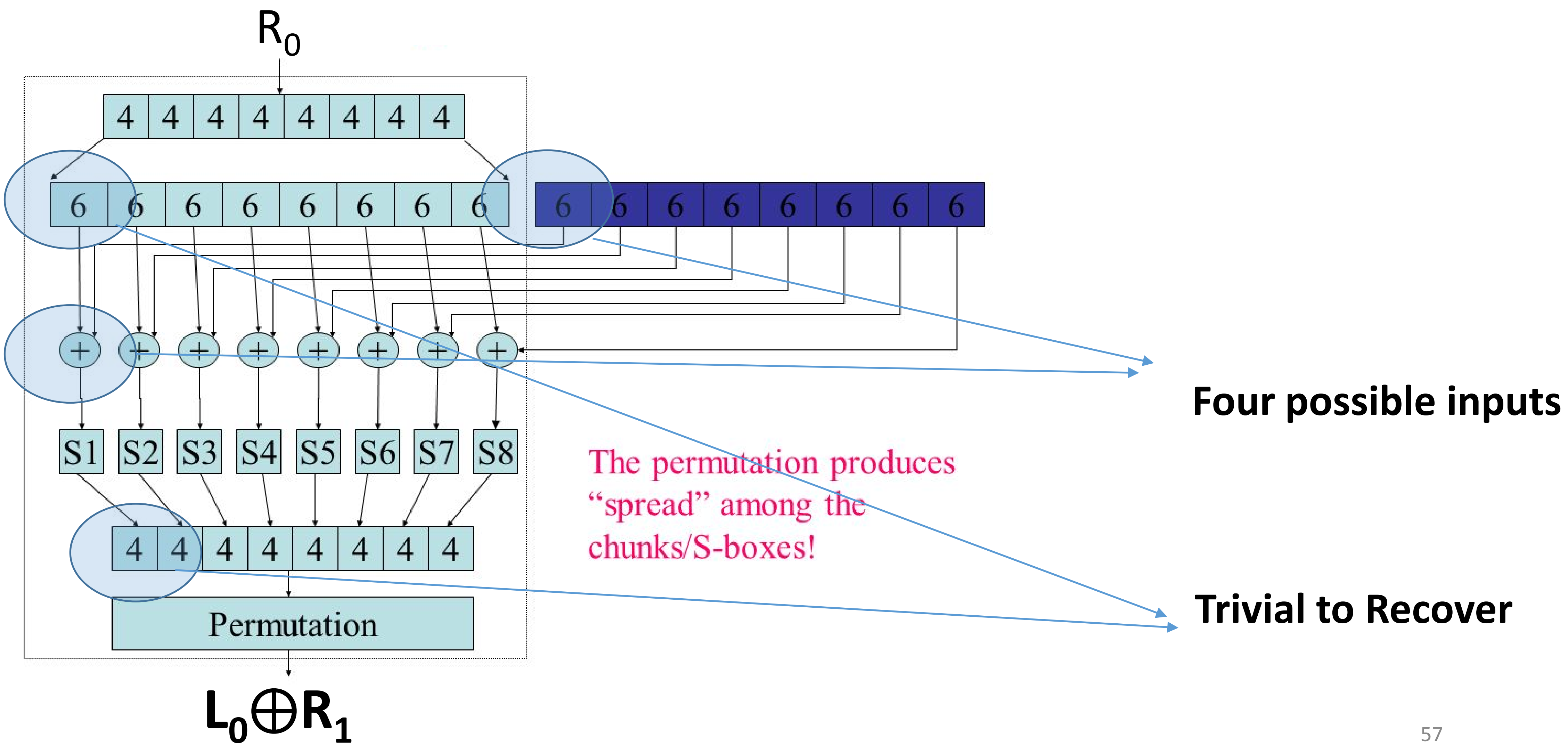
Attack on One-Round DES

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

Conclusion:

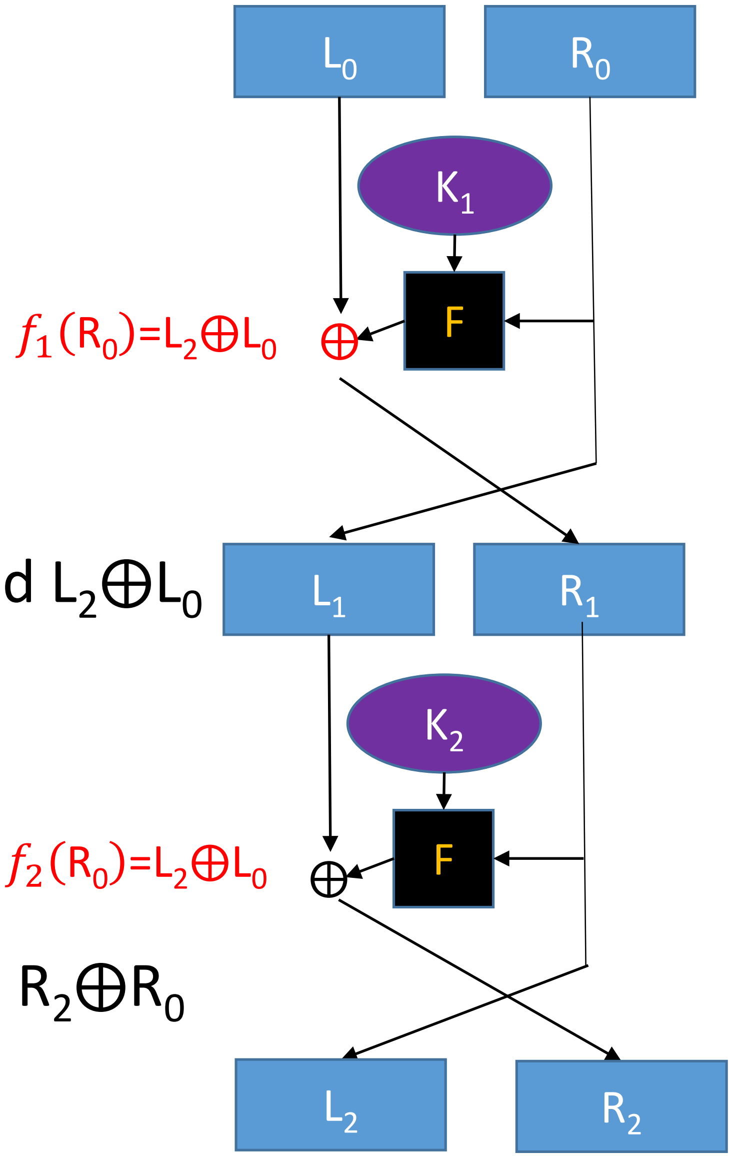
$$f_1(R_0) = L_0 \oplus R_1$$

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

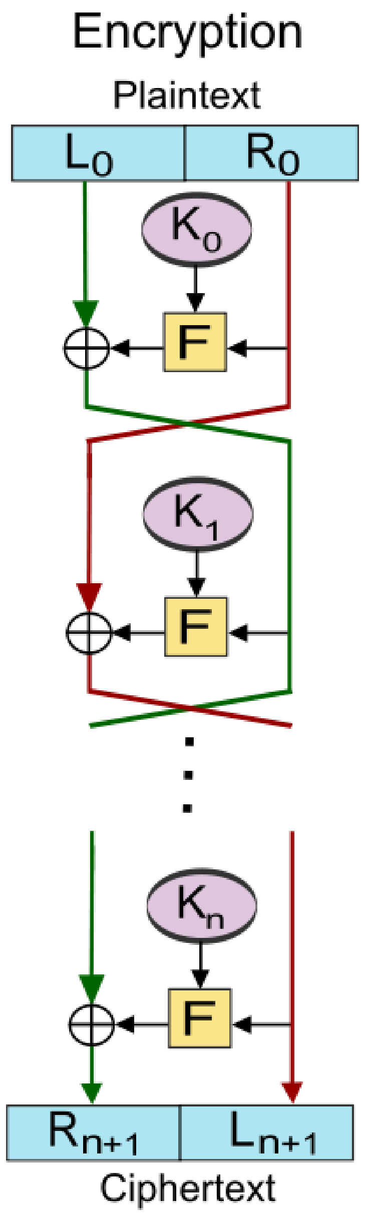
$$f_1(\mathbf{R_0}) \oplus f_3(\mathbf{R_2}) = (L_0 \oplus L_2) \oplus (L_2 \oplus R_3)$$
$$= L_0 \oplus R_3$$

We **know all of the values** L_0,R_0, R_3 and $L_3 = R_2$.

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**

$$F_{k_1}(x) = F_{k_2}^{-1}(\mathrm{c})$$

  - **Compute $F_K^{-1}(\mathrm{c})$ and $F_K(x)$ for each potential n-bit 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

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

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

Triple DES Variant 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 $\Omega(2^{2n})$
  - Brute force is more efficient: time is still $\Omega(2^{2n})$, but space usage is constant

- Key length is still just 112 bits (NIST recommends 128+ bits)

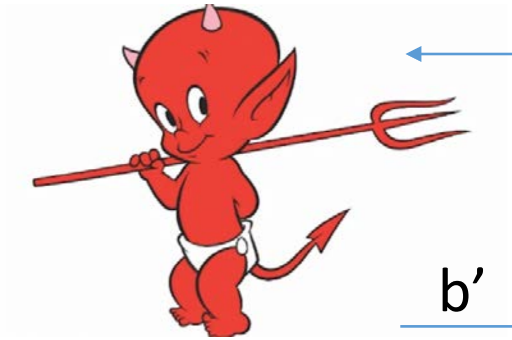
Triple DES Variant 1

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

- Standardized in 1999
- Still widely used, but it is relatively slow (three block cipher operations)
- Current gold standard: AES

CS 555:Week 6: Topic 2
Stream Ciphers

PRG Security as a Game



R

b'

Random bit b

If b=1

$r \leftarrow \{0,1\}^n$

R = G(r)

Else

$0,1\}^{\ell(n)}$

ppt attacker

negligible function

$\forall$  $\Pr\Big[$  $Guesses\ b' = b\Big] \leq \frac{1}{2} + \mu(n)$

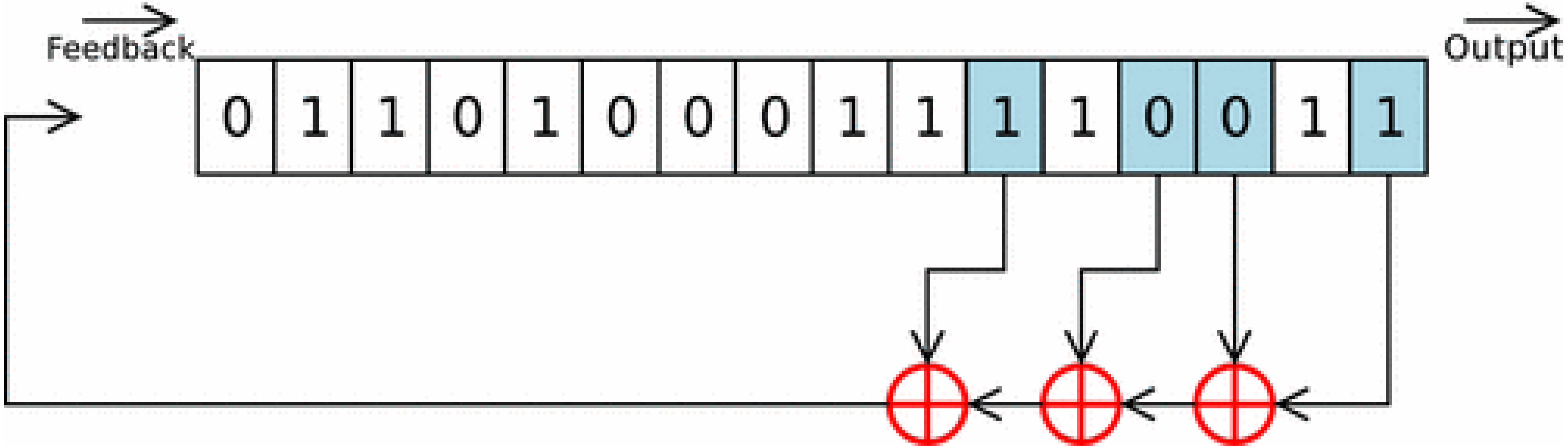
Stream Cipher vs PRG

- PRG pseudorandom bits output all at once

- Stream Cipher
  - Pseudorandom bits can be output as a stream
  - RC4, RC5 (Ron's Code)

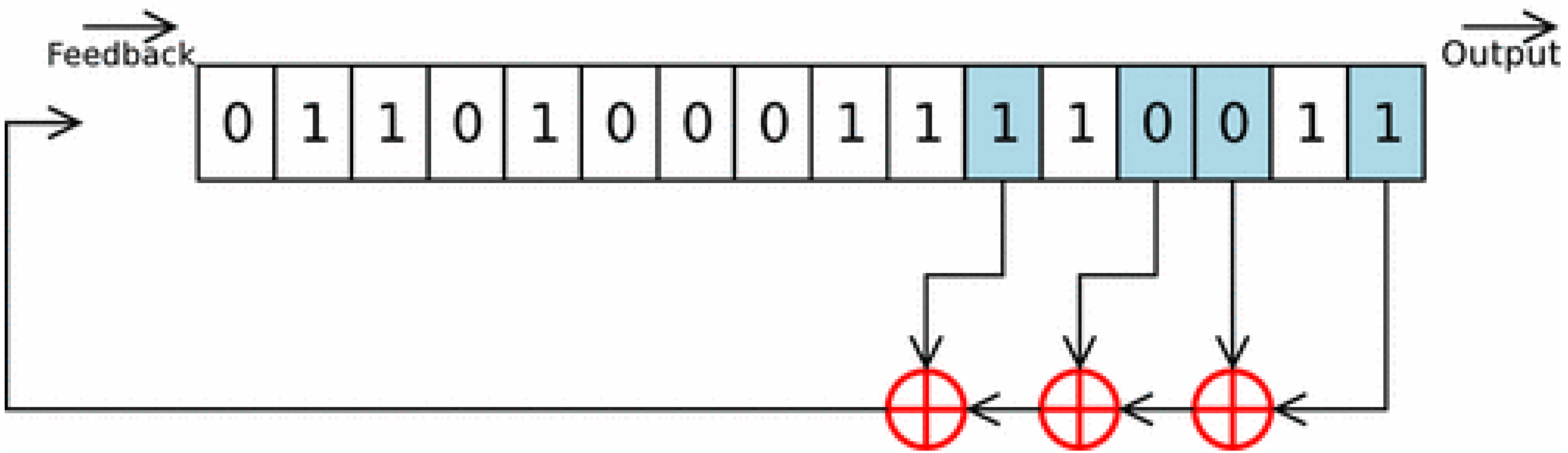
st_0 := Init(s)
For i=1 to ℓ:
    (y_i, st_i):=GetBits(st_{i-1})
Output: $y_1,\ldots,y_\ell$

Linear Feedback Shift Register



Linear Feedback Shift Register

- State at time t: $s_{n-1}^t, \ldots, s_1^t, s_0^t$ (n registers)
- Feedback Coefficients: $\mathbf{S} \subseteq \{0, \ldots, n\}$

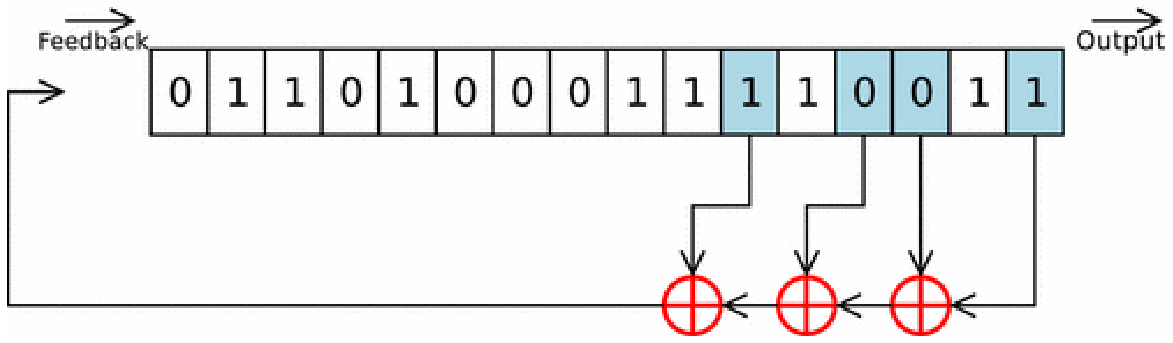


Linear Feedback Shift Register

- State at time t: $s_{n-1}^t, \ldots, s_1^t, s_0^t$ (n registers)
- Feedback Coefficients: $\mathbf{S} \subseteq \{0, \ldots, n-1\}$
- **State at time t+1:** $\bigoplus_{i \in S} s_i^t, s_{n-1}^t, \ldots, s_1^t,$

$$s_{n-1}^{t+1} = \bigoplus_{i \in S} s_i^t, \qquad \text{and} \qquad s_i^{t+1} = s_{i+1}^t \text{ for i} < \text{n} - 1$$

Output at time t+1: $y_{t+1} = s_0^t$



Linear Feedback Shift Register

- **Observation 1**: First n bits of output reveal initial state

$$y_1, \dots, y_n = s_0^0, s_1^0, \dots, s_{n-1}^0$$

- **Observation 2**: Next n bits allow us to solve for n unknowns

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & otherwise \end{cases}$$

$$y_{n+1} = y_n x_{n-1} + \cdots + y_1 x_0$$

Linear Feedback Shift Register

- **Observation 1**: First n bits of output reveal initial state

$$y_1, \dots, y_n = s_0^0, s_1^0, \dots, s_{n-1}^0$$

- **Observation 2**: Next n bits allow us to solve for n unknowns

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & otherwise \end{cases}$$

$$y_{n+1} = y_n x_{n-1} + \cdots + y_1 x_0 \text{ mod } 2$$

Linear Feedback Shift Register

- **Observation 2**: Next n bits allow us to solve for n unknowns

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & otherwise \end{cases}$$

$$y_{n+1} = y_n x_{n-1} + \cdots + y_1 x_0 \text{ mod } 2$$

$$\vdots$$

$$y_{2n} = y_{2n-1} x_{n-1} + \cdots + y_n x_0 \text{ mod } 2$$

N unknowns &
N linear independent constraints

Removing Linearity

- Attacks exploited linear relationship between state and output bits

- **Nonlinear Feedback:**

Non linear function

$$\cancel{s_{n-1}^{t+1} = \bigoplus_{i \in S} s_i^t ,}$$

$$s_{n-1}^{t+1} = g(s_0^t, s_1^t, \dots, s_{n-1}^t)$$

Removing Linearity

- Attacks exploited linear relationship between state and output bits

- **Nonlinear Combination:**

Non linear function

$$\cancel{\boldsymbol{y_{t+1} = s_0^t}}$$

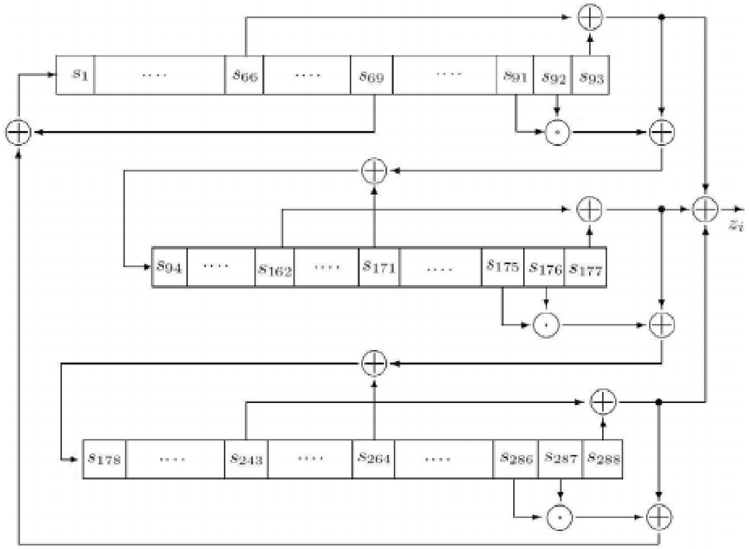
$$y_{t+1} = f(s_0^t, s_1^t, \ldots, s_{n-1}^t)$$

- **Important**: f must be balanced!

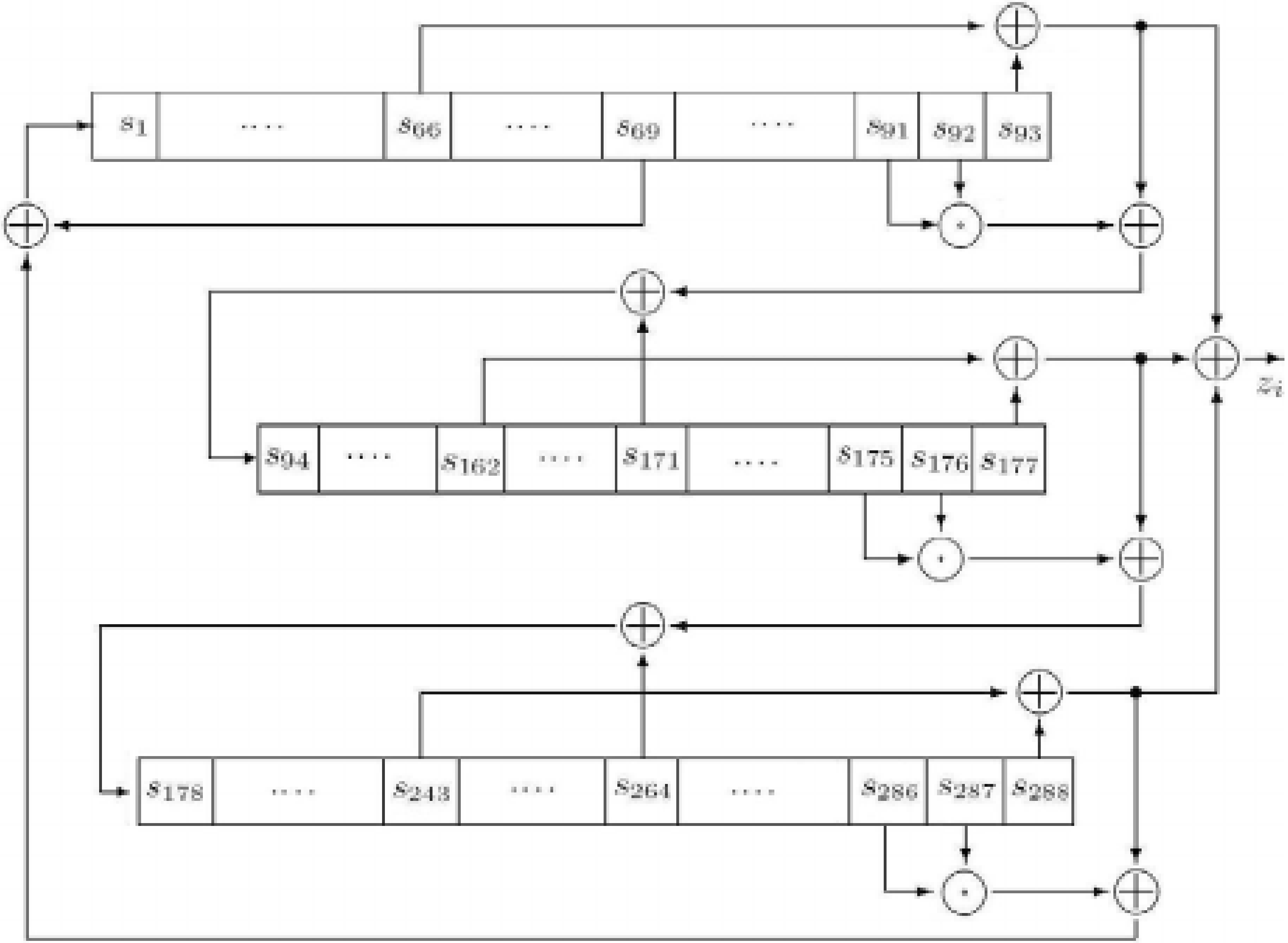
$$\Pr[f(x) = 1] \approx \frac{1}{2}$$

Trivium (2008)

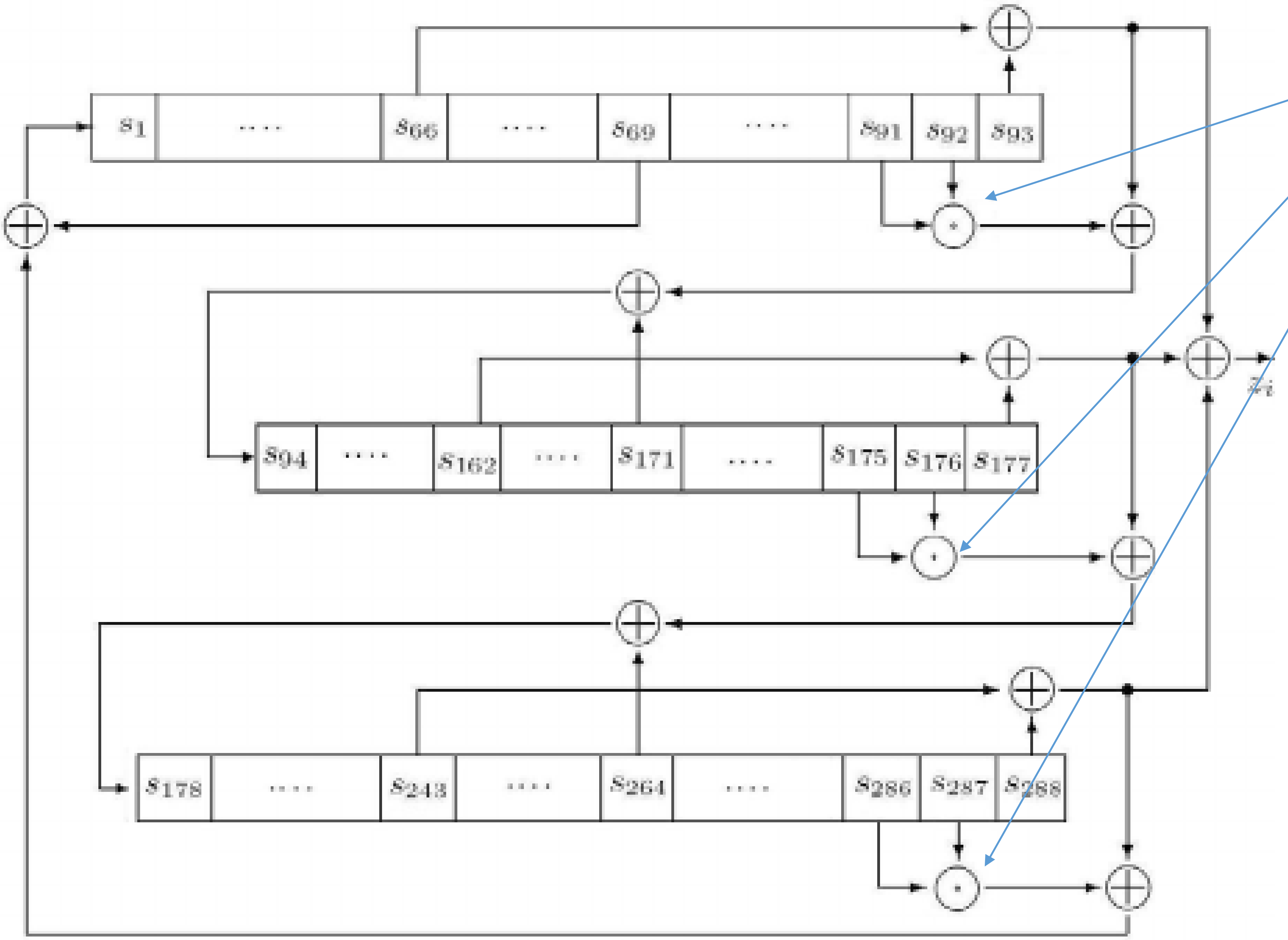
- Won the eSTREAM competition
- Currently, no known attacks are better than brute force
- Couples Output from three nonlinear Feedback Shift Registers
- First 4*288 “output bits” are discared



Trivium (2008)

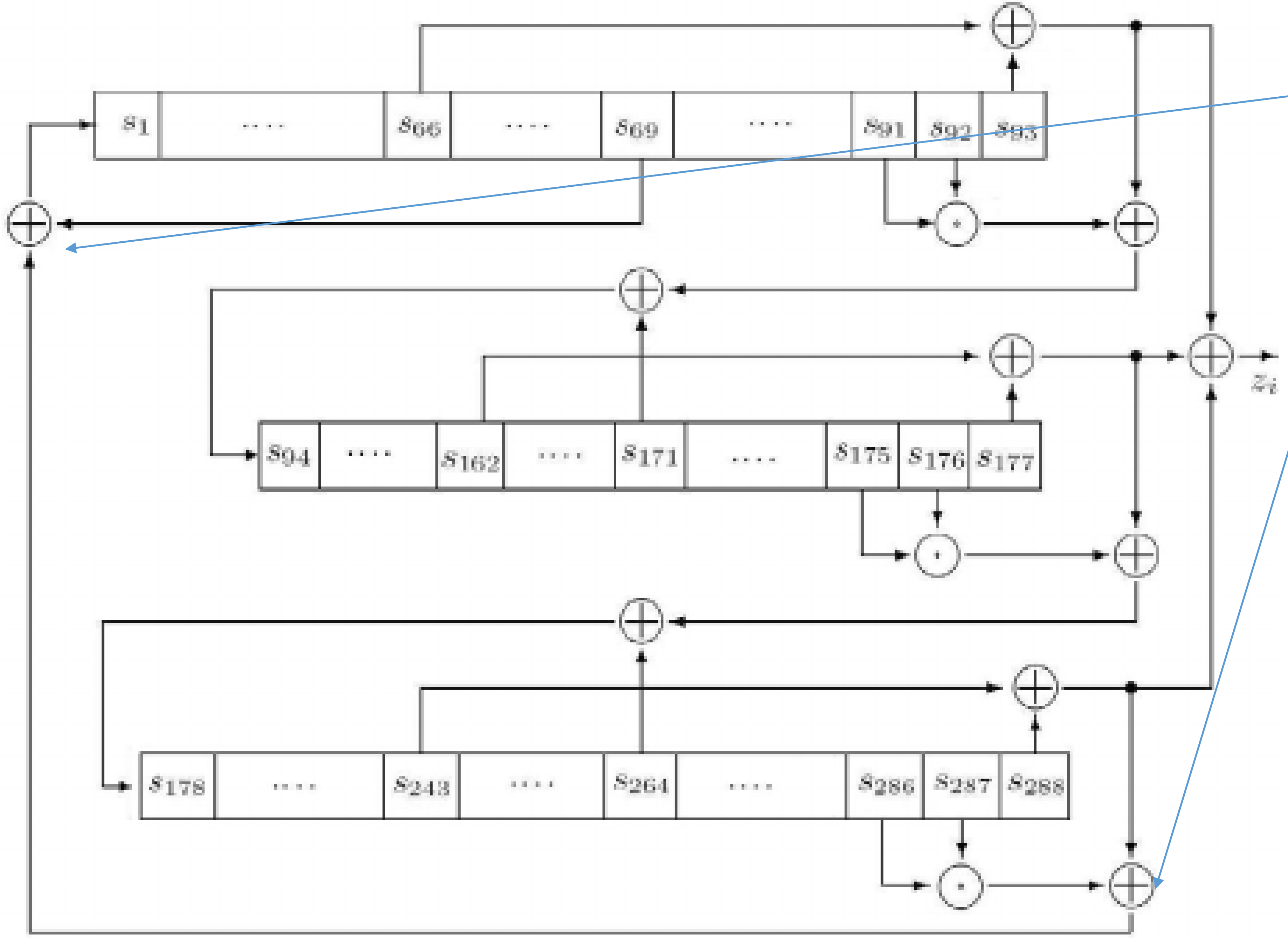


Trivium (2008)



AND (Non-linear)

Trivium (2008)



(Non-linear) Feedback

Combination Generator

- Attacks exploited linear relationship between state and output bits

- **Nonlinear Combination:**

Non linear function

$$\cancel{\boldsymbol{y_{t+1} = s_0^t}}$$

$$y_{t+1} = f(s_0^t, s_1^t, \ldots, s_{n-1}^t)$$

- **Important**: f must be balanced!

$$\Pr[f(x) = 1] \approx \frac{1}{2}$$

Feedback Shift Registers

- Good performance in hardware

- Performance is less ideal for software

Stream Ciphers

- RC4
  - A proprietary cipher owned by RSA, designed by Ron Rivest in 1987 (public 1994)
  - Widely used (web SSL/TLS, wireless WEP).
  - Distinguishable from random stream
    - Second byte of output is 0 with probability $\approx \frac{2}{256}$ (vs. $\frac{1}{256}$ for a truly random stream)

- **Newer Versions**: RC5 and RC6
- **Salsa20**
- **Rijndael** selected by NIST as AES in 2000

RC4 Attacks

- Wired Equivalent Privacy (WEP) encryption used RC4 with an initialization vector

- Description of RC4 doesn’t involve initialization vector…
  - But WEP imposes an initialization vector
  - K=IV || K’
  - Since IV is transmitted attacker may have first few bytes of the secret key K!
  - Giving the attacker partial knowledge of K often allows recovery of the entire key K’ over time!

Hash Functions from Block Ciphers

- Davies-Meyer Construction from block cipher F_K

$$H(K, x) = F_K(x) \oplus x$$

Theorem: If $F: \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is modeled as an ideal block cipher then Davies-Meyer construction is a collision-resistant hash function

(**Concrete:** Need roughly $\mathrm{q} \approx 2^{\lambda/2}$ queries to find collision)

Ideal Cipher Model: For each key K model F_K as a truly random permutation which may only be accessed in black box manner.

- (Equivalent to Random Oracle Model)

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

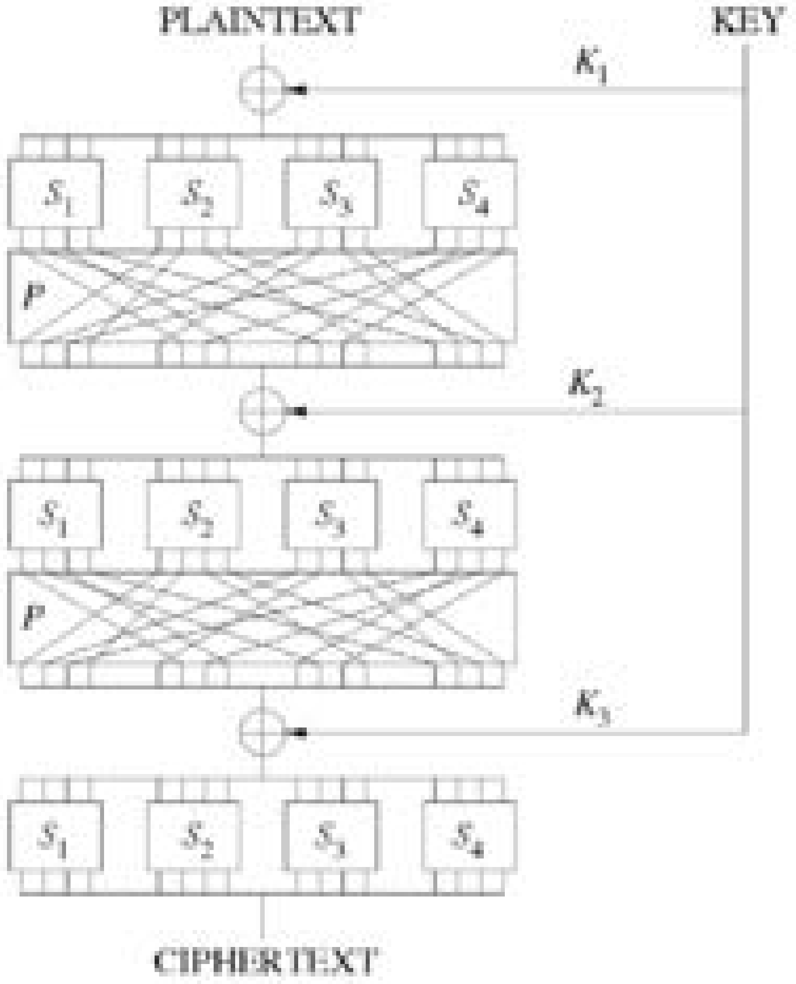
$$f(x) = S(x \oplus k)$$

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

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

1. **Key Mixing**: Set $x := x \oplus k$
2. **Substitution**: $x := S_1(x_1) \parallel S_2(x_2) \parallel \cdots \parallel S_8(x_8)$
3. **Bit Mixing Permutation**: permute the bits of x to obtain the round output

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)

Announcements

- Homework 2 Solutions Posted (See Piazza).
  - Please read through carefully and make sure you understand the solutions to each problem.
  - Grading in progress

- No Class on Tuesday (October Break)

- Look for Practice Midterm Next Week

Recap

- 2DES, Meet in the Middle Attack
- 3DES
- Stream Ciphers
  - Breaking Linear Feedback Shift Registers
  - Trivium
- AES

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) Related-Key Attack on 11 round version of AES
  - Related Key Attack: Attacker convinces Alice to use two related (but unknown) keys
  - 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) Related Key Attack on 192-bit and 256 bit version of AES
  - recovers 256-bit key in time $2^{99.5}$.
- (2011) Key Recovery attack on AES-128 in time $2^{126.2}$.
  - Improved to $2^{126.0}$ for AES-128, $2^{189.9}$ for AES-192 and $2^{254.3}$ for AES-256
- First public cipher approved by NSA for Top Secret information
  - SECRET level (AES-128,AES-192 & AES-256), TOP SECRET level (~~AES-128~~,AES-192 & AES-256)

NIST Recommendations

Ok, as CRHF and in Digital Signatures

Ok, to use for HMAC, Key Derivation and as PRG

80 bits-security is no longer acceptable

Date	Minimum of Strength	Symmetric Algorithms	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash (A)	Hash (B)
(Legacy)	80	2TDEA*	1024	160	1024	160	SHA-1**	
2016 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-512/224 SHA3-224	
2016 - 2030 & beyond	128	AES-128	3072	256	3072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1
2016 - 2030 & beyond	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224
2016 - 2030 & beyond	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-512

Recommendations from Other Groups (Including NIST): www.keylength.com

Linear Cryptanalysis

$$y = F_K(x)$$

Definition: Fixed set of input bits $i_1, \dots, i_{in}$ and output bits $i_1', \dots, i_{out}'$ are said to have ε-linear bias if the following holds

$$\left|Pr\left[x_{i_1} \oplus x_{i_2} \dots \oplus x_{i_{in}} \oplus y_{i_1'} \oplus y_{i_2'} \dots \oplus y_{i_{out}'}\right]\right| = \varepsilon$$

(randomness taken over the selection of input x and secret key K)

Linear Cryptanalysis

Definition: Fixed set of input bits $i_1, \ldots, i_{in}$ and output bits $i_1', \ldots, i_{out}'$ are said to have ε-linear bias if the following holds

$$\left|\Pr\left[x_{i_1} \oplus x_{i_2} \ldots \oplus x_{i_{in}} \oplus y_{i_1'}, \oplus y_{i_2'}, \ldots \oplus y_{i_{out}'}\right] - \frac{1}{2}\right| = \varepsilon$$

(randomness taken over the selection of input x and secret key K, $y = F_K(x)$)

Matsui: DES can be broken with just 2^{43} *known* plaintext/ciphertext pairs.

- Lots of examples needed!
- But the examples do not need to be chosen plaintext/ciphertext pairs…
- One encrypted file can provide a large amounts of known plaintext

Differential Cryptanalysis

Definition: We say that the differential $\left(\triangle_x,\triangle_y\right)$ occurs with probability p in the keyed block cipher F if

$$\Pr\left[F_K(x_1)\oplus F_K(x_1\oplus \triangle_x) =\triangle_y\right] \geq p$$

Can Lead to Efficient (Round) Key Recovery Attacks

Exploiting Weakness Requires: well over $\frac{1}{p}$ chosen plaintext-ciphertext pairs

Differentials in S-box can lead to (weaker) differentials in SPN.