

# Cryptography

## Lecture 15

Arkady Yerukhimovich

October 21, 2024

- 1 Exam 1 and the rest of the semester
- 2 Lecture 13 Review
- 3 Constructing Practical Block Ciphers – Overview (Chapters 6.Intro, 6.2.Intro)
- 4 Substitution-Permutation Networks (SPN) and AES (Chapters 6.2.1, 6.2.5)

# Exam 1 Grades

## Exam 1 Stats:

- Max – 89
- Mean – 61.7
- Median – 65.5

# Research Project

## Important Dates:

- Oct. 25 – Project Proposals Due
- Nov. 22 – Presentations Due
- Dec. 4 – Workshop Day

# Research Project

## Important Dates:

- Oct. 25 – Project Proposals Due
- Nov. 22 – Presentations Due
- Dec. 4 – Workshop Day

## Next Steps:

- 1 Form your groups (3 people max) – Do this ASAP
- 2 Choose your topic – I am happy to help, but you need to start the conversation
- 3 Do the research
- 4 Record 20-minute presentation by due date
- 5 Use Slack to start discussions about projects
- 6 Live discussion of all projects on Dec. 4

# Outline

- 1 Exam 1 and the rest of the semester
- 2 Lecture 13 Review**
- 3 Constructing Practical Block Ciphers – Overview (Chapters 6.Intro, 6.2.Intro)
- 4 Substitution-Permutation Networks (SPN) and AES (Chapters 6.2.1, 6.2.5)

# Lecture 13 Review

- Collision-resistant Hash Functions
- Merkle-Damgard Domain Extension for CRHF
- Applications of Hash Functions
  - Hash-and-MAC
  - Password-based authentication
  - and more

# Outline

- 1 Exam 1 and the rest of the semester
- 2 Lecture 13 Review
- 3 Constructing Practical Block Ciphers – Overview (Chapters 6.Intro, 6.2.Intro)
- 4 Substitution-Permutation Networks (SPN) and AES (Chapters 6.2.1, 6.2.5)



# And Now For Something Completely Different

- This week, we will step away from asymptotic notions of security

# And Now For Something Completely Different

- This week, we will step away from asymptotic notions of security
- Focus on constructing *practically efficient* pseudorandom objects

# And Now For Something Completely Different

- This week, we will step away from asymptotic notions of security
- Focus on constructing *practically efficient* pseudorandom objects
- Constructions will be largely combinatorial, and proofs informal

# What Is a Block Cipher?

## Block Cipher

*Fixed input size* strong PRP

$$F : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$$

- $n$  - key size
- $\ell$  - block size

# What Is a Block Cipher?

## Block Cipher

*Fixed input size* strong PRP

$$F : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$$

- $n$  - key size
  - $\ell$  - block size
- 
- $n$  and  $\ell$  are both fixed (e.g., 256 bits)

# What Is a Block Cipher?

## Block Cipher

*Fixed input size* strong PRP

$$F : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$$

- $n$  - key size
  - $\ell$  - block size
- 
- $n$  and  $\ell$  are both fixed (e.g., 256 bits)
  - Security: Should take (approximately)  $2^n$  time to attack

# What Is a Block Cipher?

## Block Cipher

*Fixed input size* strong PRP

$$F : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$$

- $n$  - key size
  - $\ell$  - block size
- 
- $n$  and  $\ell$  are both fixed (e.g., 256 bits)
  - Security: Should take (approximately)  $2^n$  time to attack
    - Can't do much better than exhaustive search

# What Is a Block Cipher?

## Block Cipher

*Fixed input size* strong PRP

$$F : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$$

- $n$  - key size
  - $\ell$  - block size
- 
- $n$  and  $\ell$  are both fixed (e.g., 256 bits)
  - Security: Should take (approximately)  $2^n$  time to attack
    - Can't do much better than exhaustive search
    - A cipher with  $n = 256$  that can be broken in time  $2^{128}$  is insecure



# Block Cipher Security

Block cipher security goals:

- Indistinguishability from a random permutation - PRP

# Block Cipher Security

Block cipher security goals:

- Indistinguishability from a random permutation - PRP
- Security against key-recovery

# Block Cipher Security

Block cipher security goals:

- Indistinguishability from a random permutation - PRP
- Security against key-recovery
- Hard to invert

# Block Cipher Security

Block cipher security goals:

- Indistinguishability from a random permutation - PRP
- Security against key-recovery
- Hard to invert
- Many additional (informal) goals:

Block cipher security goals:

- Indistinguishability from a random permutation - PRP
- Security against key-recovery
- Hard to invert
- Many additional (informal) goals:
  - Security against related-key attacks -  $F_k(\cdot)$  should tell you nothing about  $F_{k'}(\cdot)$  even if  $k$  and  $k'$  differ in only 1 bit

Block cipher security goals:

- Indistinguishability from a random permutation - PRP
- Security against key-recovery
- Hard to invert
- Many additional (informal) goals:
  - Security against related-key attacks -  $F_k(\cdot)$  should tell you nothing about  $F_{k'}(\cdot)$  even if  $k$  and  $k'$  differ in only 1 bit
  - No weak keys -  $F_k$  should look random for all  $k \in \{0, 1\}^n$

Block cipher security goals:

- Indistinguishability from a random permutation - PRP
- Security against key-recovery
- Hard to invert
- Many additional (informal) goals:
  - Security against related-key attacks -  $F_k(\cdot)$  should tell you nothing about  $F_{k'}(\cdot)$  even if  $k$  and  $k'$  differ in only 1 bit
  - No weak keys -  $F_k$  should look random for all  $k \in \{0, 1\}^n$
  - For many block ciphers these properties are not well defined, not needed by all constructions

# Block Cipher Adversary Models

We consider several adversary types:



# Block Cipher Adversary Models

We consider several adversary types:

- Known-plaintext attack:  $\{(x_i, F_k(x_i))\}$  with  $x_i$  known to  $\mathcal{A}$

# Block Cipher Adversary Models

We consider several adversary types:

- Known-plaintext attack:  $\{(x_i, F_k(x_i))\}$  with  $x_i$  known to  $\mathcal{A}$
- Chosen-plaintext attack:  $\{F_k(x_i)\}$  for  $x_i$  chosen by  $\mathcal{A}$

# Block Cipher Adversary Models

We consider several adversary types:

- Known-plaintext attack:  $\{(x_i, F_k(x_i))\}$  with  $x_i$  known to  $\mathcal{A}$
- Chosen-plaintext attack:  $\{F_k(x_i)\}$  for  $x_i$  chosen by  $\mathcal{A}$
- Chosen-ciphertext attack:  $\{F_k(x_i)\}, \{F_k^{-1}(y_i)\}$  for  $x_i, y_i$  chosen by  $\mathcal{A}$

# Block Cipher Adversary Models

We consider several adversary types:

- Known-plaintext attack:  $\{(x_i, F_k(x_i))\}$  with  $x_i$  known to  $\mathcal{A}$
- Chosen-plaintext attack:  $\{F_k(x_i)\}$  for  $x_i$  chosen by  $\mathcal{A}$
- Chosen-ciphertext attack:  $\{F_k(x_i)\}, \{F_k^{-1}(y_i)\}$  for  $x_i, y_i$  chosen by  $\mathcal{A}$

## Important

Remember that a block cipher is not an encryption scheme

# Building a Block Cipher

## The Challenge

There are  $2^\ell!$  different permutations on  $\ell$  bits

# Building a Block Cipher

## The Challenge

There are  $2^\ell!$  different permutations on  $\ell$  bits

- Writing down a permutation takes  $\log 2^\ell! \approx \ell \cdot 2^\ell$  bits

We use  $\ell \geq 128$ , so this is close to number of atoms in the universe

# Building a Block Cipher

## The Challenge

There are  $2^\ell!$  different permutations on  $\ell$  bits

- Writing down a permutation takes  $\log 2^\ell! \approx \ell \cdot 2^\ell$  bits  
We use  $\ell \geq 128$ , so this is close to number of atoms in the universe
- Need a short description  
Otherwise, how do you evaluate efficiently?

# Building a Block Cipher

## The Challenge

There are  $2^\ell!$  different permutations on  $\ell$  bits

- Writing down a permutation takes  $\log 2^\ell! \approx \ell \cdot 2^\ell$  bits  
We use  $\ell \geq 128$ , so this is close to number of atoms in the universe
- Need a short description  
Otherwise, how do you evaluate efficiently?

Solution Idea: Instead of one permutation on  $\ell$  bits, compose many small (e.g., 8-bit) permutations

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



# Building a Block Cipher

## The Challenge

There are  $2^\ell!$  different permutations on  $\ell$  bits

- Writing down a permutation takes  $\log 2^\ell! \approx \ell \cdot 2^\ell$  bits  
We use  $\ell \geq 128$ , so this is close to number of atoms in the universe
- Need a short description  
Otherwise, how do you evaluate efficiently?

Solution Idea: Instead of one permutation on  $\ell$  bits, compose many small (e.g., 8-bit) permutations

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

- Pro:  $|F| = 16 \cdot \log(2^8!) \approx 16 \cdot 600 \text{ bits} \approx 3KB \ll 128 \cdot 2^{128}$

# Building a Block Cipher

## The Challenge

There are  $2^\ell!$  different permutations on  $\ell$  bits

- Writing down a permutation takes  $\log 2^\ell! \approx \ell \cdot 2^\ell$  bits  
We use  $\ell \geq 128$ , so this is close to number of atoms in the universe
- Need a short description  
Otherwise, how do you evaluate efficiently?

Solution Idea: Instead of one permutation on  $\ell$  bits, compose many small (e.g., 8-bit) permutations

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

- Pro:  $|F| = 16 \cdot \log(2^8!) \approx 16 \cdot 600 \text{ bits} \approx 3KB \ll 128 \cdot 2^{128}$
- Con: If  $x$  only changes in one bit, only one block is affected

# Avalanche Effect

A critical goal in design of block-ciphers is to achieve something called the *avalanche effect*.

## Avalanche Effect

Small change in the input must “affect” every bit of the output.

# Avalanche Effect

A critical goal in design of block-ciphers is to achieve something called the *avalanche effect*.

## Avalanche Effect

Small change in the input must “affect” every bit of the output.

What this means:

# Avalanche Effect

A critical goal in design of block-ciphers is to achieve something called the *avalanche effect*.

## Avalanche Effect

Small change in the input must “affect” every bit of the output.

What this means:

- This does not require that all bits of output change, only that the bits of the output are not statistically independent of the change in input.

# Avalanche Effect

A critical goal in design of block-ciphers is to achieve something called the *avalanche effect*.

## Avalanche Effect

Small change in the input must “affect” every bit of the output.

What this means:

- This does not require that all bits of output change, only that the bits of the output are not statistically independent of the change in input.
- Even a 1-bit change to input must have statistical impact on all bits.

# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

- Confusion:  $F_k(x) = f_1(x_1) || f_2(x_2) || \cdots || f_{16}(x_{16})$ 
  - $|x_i| = 8$  bits
  - $f_i$  can be a random permutation on 8 bits
  - Goal: Introduce random local changes



# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

- Confusion:  $F_k(x) = f_1(x_1) || f_2(x_2) || \cdots || f_{16}(x_{16})$ 
  - $|x_i| = 8$  bits
  - $f_i$  can be a random permutation on 8 bits
  - Goal: Introduce random local changes
- Diffusion: Shuffle the bits using a *mixing permutation*
  - This just moves bits around
  - Goal: Spread confusion across all 128 bits

# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

- Confusion:  $F_k(x) = f_1(x_1) || f_2(x_2) || \dots || f_{16}(x_{16})$ 
  - $|x_i| = 8$  bits
  - $f_i$  can be a random permutation on 8 bits
  - Goal: Introduce random local changes
- Diffusion: Shuffle the bits using a *mixing permutation*
  - This just moves bits around
  - Goal: Spread confusion across all 128 bits
- Repeat many times

Intuition:

# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

- Confusion:  $F_k(x) = f_1(x_1) || f_2(x_2) || \cdots || f_{16}(x_{16})$ 
  - $|x_i| = 8$  bits
  - $f_i$  can be a random permutation on 8 bits
  - Goal: Introduce random local changes
- Diffusion: Shuffle the bits using a *mixing permutation*
  - This just moves bits around
  - Goal: Spread confusion across all 128 bits
- Repeat many times

Intuition:

- Each confusion step introduces local changes

# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

- Confusion:  $F_k(x) = f_1(x_1) || f_2(x_2) || \dots || f_{16}(x_{16})$ 
  - $|x_i| = 8$  bits
  - $f_i$  can be a random permutation on 8 bits
  - Goal: Introduce random local changes
- Diffusion: Shuffle the bits using a *mixing permutation*
  - This just moves bits around
  - Goal: Spread confusion across all 128 bits
- Repeat many times

Intuition:

- Each confusion step introduces local changes
- Each diffusion step distributes those changes across blocks

# Confusion-Diffusion Paradigm

Many block-ciphers follow the following general approach:

- Confusion:  $F_k(x) = f_1(x_1) || f_2(x_2) || \dots || f_{16}(x_{16})$ 
  - $|x_i| = 8$  bits
  - $f_i$  can be a random permutation on 8 bits
  - Goal: Introduce random local changes
- Diffusion: Shuffle the bits using a *mixing permutation*
  - This just moves bits around
  - Goal: Spread confusion across all 128 bits
- Repeat many times

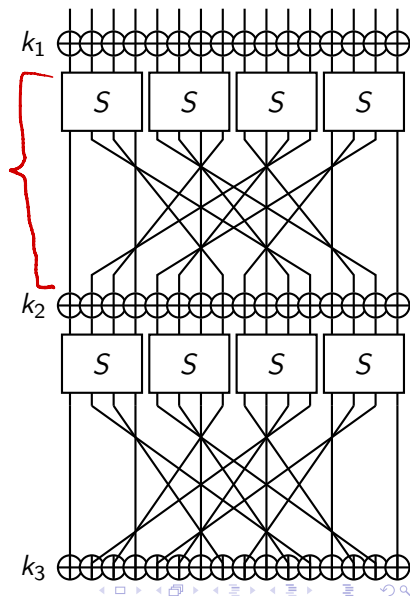
Intuition:

- Each confusion step introduces local changes
- Each diffusion step distributes those changes across blocks
- After many iterations, all bits are affected – avalanche effect

# Outline

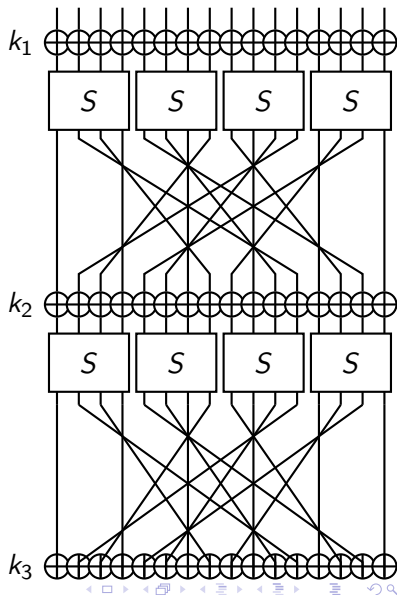
- 1 Exam 1 and the rest of the semester
- 2 Lecture 13 Review
- 3 Constructing Practical Block Ciphers – Overview (Chapters 6.Intro, 6.2.Intro)
- 4 Substitution-Permutation Networks (SPN) and AES (Chapters 6.2.1, 6.2.5)

# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)



# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

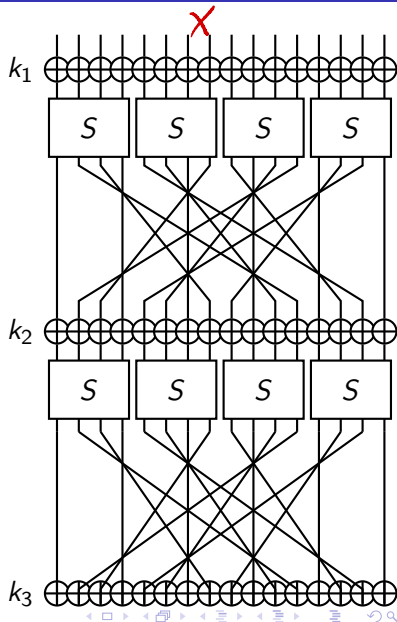




# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

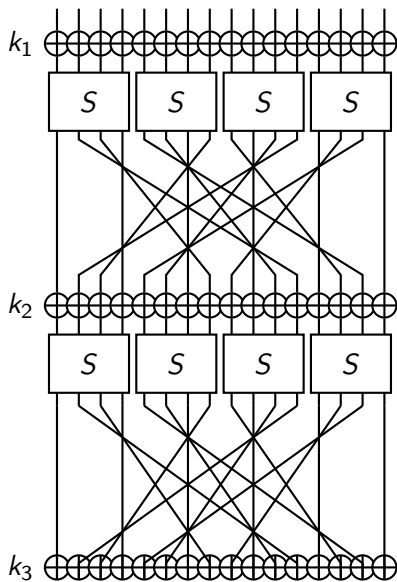
- 1 Key mixing: Set  $x = x \oplus k_i$



# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

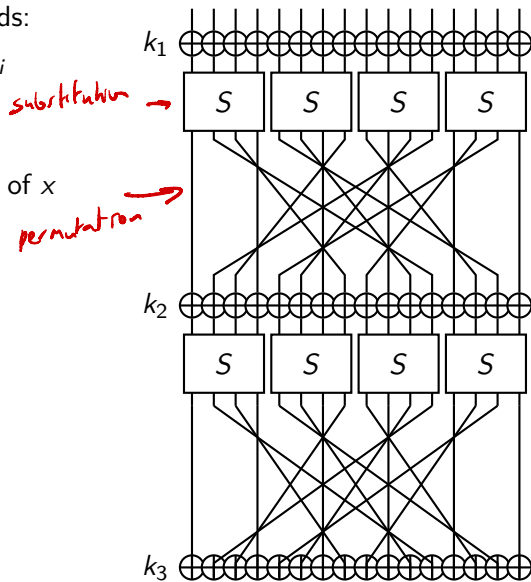
- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$



# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

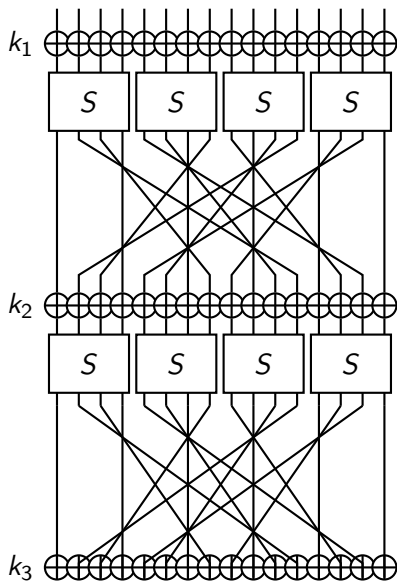


# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Observations:



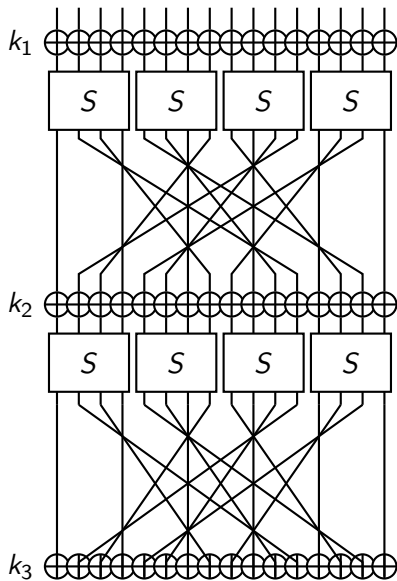
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Observations:

- $S_1, \dots, S_8$  are public (S-boxes)



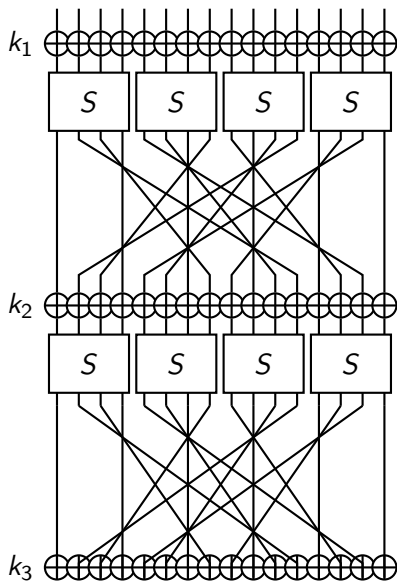
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Observations:

- $S_1, \dots, S_8$  are public (S-boxes)
- Mixing perm. is also public



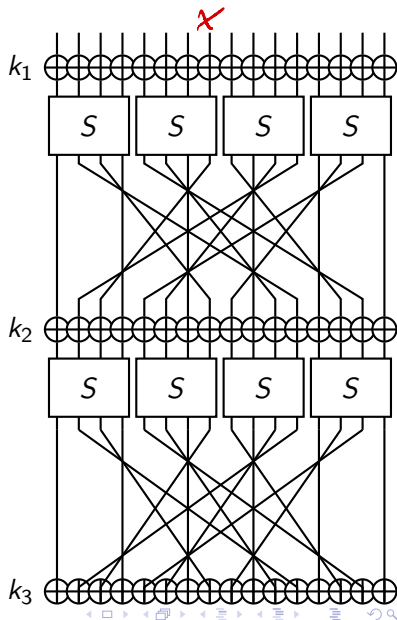
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Observations:

- $S_1, \dots, S_8$  are public (S-boxes)
- Mixing perm. is also public
- Key  $k$  only used in key-mixing



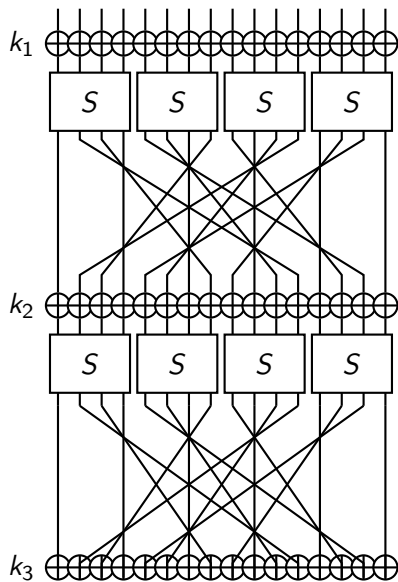
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Observations:

- $S_1, \dots, S_8$  are public (S-boxes)
- Mixing perm. is also public
- Key  $k$  only used in key-mixing
- Need one more key-mixing step after last round to prevent  $\mathcal{A}$  from inverting round





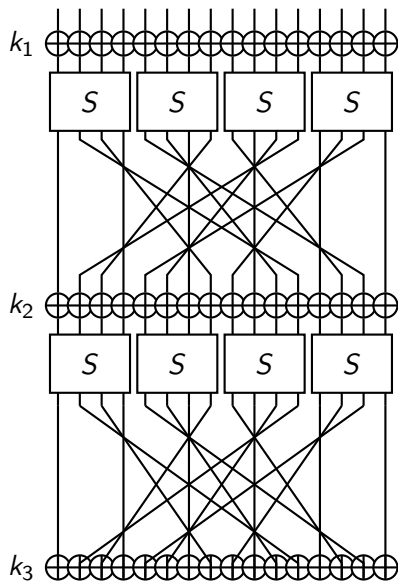
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Observations:

- $S_1, \dots, S_8$  are public (S-boxes)
- Mixing perm. is also public
- Key  $k$  only used in key-mixing
- Need one more key-mixing step after last round to prevent  $\mathcal{A}$  from inverting round
- Direct application of confusion-diffusion paradigm

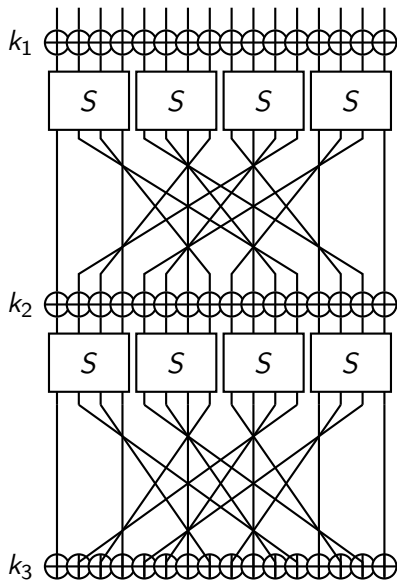


# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Key Schedule:



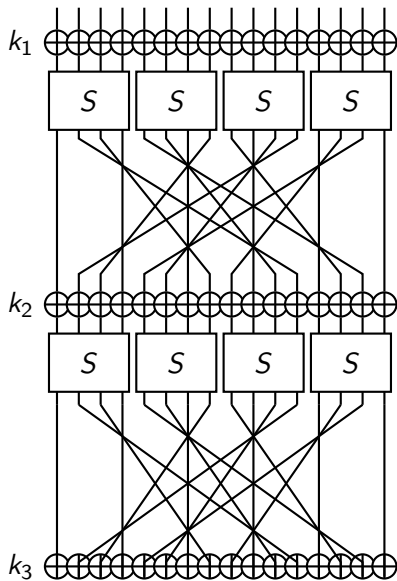
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Key Schedule:

- Each round uses different key  $k_i$



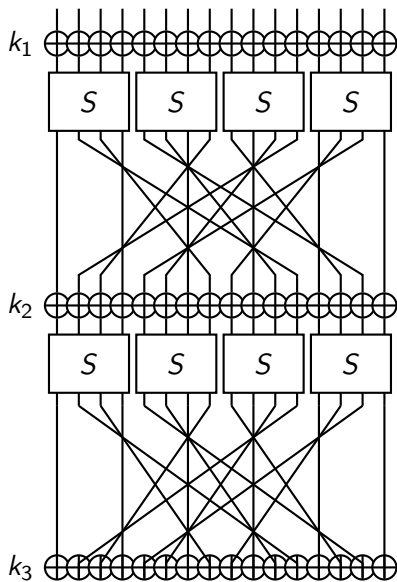
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Key Schedule:

- Each round uses different key  $k_i$
- $k_i$  are derived from *master key*  $k$  via *key schedule*



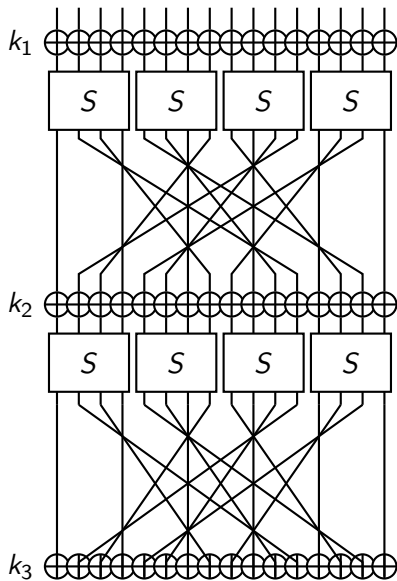
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Key Schedule:

- Each round uses different key  $k_i$
- $k_i$  are derived from *master key*  $k$  via *key schedule*
- Key schedule often very simple –  
E.g., take a subset of the bits of  $k$



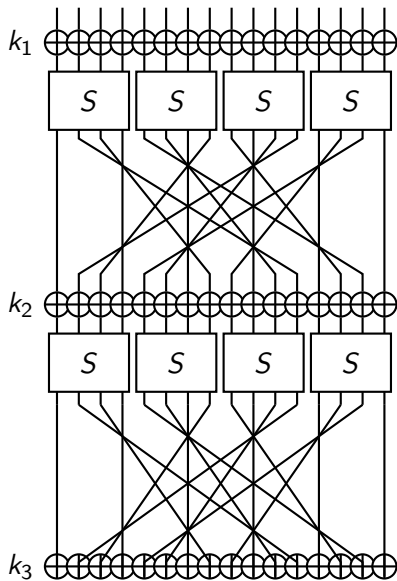
# Substitution-Permutation Network ( $\ell = 64$ using 8 Perms)

Repeat following for many rounds:

- 1 Key mixing: Set  $x = x \oplus k_i$
- 2 Substitution:  
 $x = S_1(x_1) || \dots || S_8(x_8)$
- 3 Permutation: Permute bits of  $x$

Key Schedule:

- Each round uses different key  $k_i$
- $k_i$  are derived from *master key*  $k$  via *key schedule*
- Key schedule often very simple –  
E.g., take a subset of the bits of  $k$
- $r$ -round SPN needs  $r + 1$  keys



# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

To achieve avalanche effect, we guarantee that:

- ① S-boxes: Changing 1 bit of input changes  $\geq 2$  bits of output



# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

To achieve avalanche effect, we guarantee that:

- 1 S-boxes: Changing 1 bit of input changes  $\geq 2$  bits of output
- 2 Mixing permutations: Output bits of any S-box are input to  $> 1$  S-boxes at next round

# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

To achieve avalanche effect, we guarantee that:

- 1 S-boxes: Changing 1 bit of input changes  $\geq 2$  bits of output
- 2 Mixing permutations: Output bits of any S-box are input to  $> 1$  S-boxes at next round

Analysis:

# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

To achieve avalanche effect, we guarantee that:

- 1 S-boxes: Changing 1 bit of input changes  $\geq 2$  bits of output
- 2 Mixing permutations: Output bits of any S-box are input to  $> 1$  S-boxes at next round

Analysis:

- If 1 bit change affects 2 bits, need 6 rounds to affect all  $2^6 = 64$  bits

# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

To achieve avalanche effect, we guarantee that:

- 1 S-boxes: Changing 1 bit of input changes  $\geq 2$  bits of output
- 2 Mixing permutations: Output bits of any S-box are input to  $> 1$  S-boxes at next round

Analysis:

- If 1 bit change affects 2 bits, need 6 rounds to affect all  $2^6 = 64$  bits
- Also want avalanche effect for inverse

# Avalanche Effect of SPN

## Avalanche Effect

Small change in input must affect every bit of output.

To achieve avalanche effect, we guarantee that:

- 1 S-boxes: Changing 1 bit of input changes  $\geq 2$  bits of output
- 2 Mixing permutations: Output bits of any S-box are input to  $> 1$  S-boxes at next round

Analysis:

- If 1 bit change affects 2 bits, need 6 rounds to affect all  $2^6 = 64$  bits
- Also want avalanche effect for inverse
- Hence, run for  $\gg 6$  rounds

## Advanced Encryption Standard (AES)

## Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen

## Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000



## Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000
- Now, standardized and very widely used

## Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000
- Now, standardized and very widely used
- Hardware support (e.g., AES-NI) makes this very fast:  $10^8$  per second

## Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000
- Now, standardized and very widely used
- Hardware support (e.g., AES-NI) makes this very fast:  $10^8$  per second
- This is the right block-cipher to use for most applications

Available Versions:  $\ell = 128$  (16 Bytes),  $n = 128, 192, \underline{256}$

Available Versions:  $\ell = 128$  (16 Bytes),  $n = 128, 192, 256$

- Change key schedule for different key lengths

Available Versions:  $\ell = 128$ (16 Bytes),  $n = 128, 192, 256$

- Change key schedule for different key lengths
- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key

Available Versions:  $\ell = 128(16 \text{ Bytes}), n = 128, 192, 256$

- Change key schedule for different key lengths
- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key

State:  $4 \times 4$  array of Bytes

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{pmatrix}$$

- ① Add Round Key: View  $k_i = k_{11} || k_{12} || \dots || k_{44}$  with  $|k_{ij}| = 1$  Byte

$$\begin{pmatrix} B_{11} \oplus k_{11} & B_{12} \oplus k_{12} & B_{13} \oplus k_{13} & B_{14} \oplus k_{14} \\ B_{21} \oplus k_{21} & B_{22} \oplus k_{22} & B_{23} \oplus k_{23} & B_{24} \oplus k_{24} \\ B_{31} \oplus k_{31} & B_{32} \oplus k_{32} & B_{33} \oplus k_{33} & B_{34} \oplus k_{34} \\ B_{41} \oplus k_{41} & B_{42} \oplus k_{42} & B_{43} \oplus k_{43} & B_{44} \oplus k_{44} \end{pmatrix}$$



- ① Add Round Key: View  $k_i = k_{i1} || k_{i2} || \dots || k_{i4}$  with  $|k_{ij}| = 1$  Byte

$$\begin{pmatrix} B_{11} \oplus k_{11} & B_{12} \oplus k_{12} & B_{13} \oplus k_{13} & B_{14} \oplus k_{14} \\ B_{21} \oplus k_{21} & B_{22} \oplus k_{22} & B_{23} \oplus k_{23} & B_{24} \oplus k_{24} \\ B_{31} \oplus k_{31} & B_{32} \oplus k_{32} & B_{33} \oplus k_{33} & B_{34} \oplus k_{34} \\ B_{41} \oplus k_{41} & B_{42} \oplus k_{42} & B_{43} \oplus k_{43} & B_{44} \oplus k_{44} \end{pmatrix}$$

- ② SubBytes: Uses a single 8-bit S-box

$$\begin{pmatrix} S(B_{11}) & S(B_{12}) & S(B_{13}) & S(B_{14}) \\ S(B_{21}) & S(B_{22}) & S(B_{23}) & S(B_{24}) \\ S(B_{31}) & S(B_{32}) & S(B_{33}) & S(B_{34}) \\ S(B_{41}) & S(B_{42}) & S(B_{43}) & S(B_{44}) \end{pmatrix}$$

- ③ Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & (0 \text{ shift}) \\ B_{22} & B_{23} & B_{24} & B_{21} & (1 \text{ shift}) \\ B_{33} & B_{34} & B_{31} & B_{32} & (2 \text{ shift}) \\ B_{44} & B_{41} & B_{42} & B_{43} & (3 \text{ shift}) \end{pmatrix}$$

- ③ Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & (0 \text{ shift}) \\ B_{22} & B_{23} & B_{24} & B_{21} & (1 \text{ shift}) \\ B_{33} & B_{34} & B_{31} & B_{32} & (2 \text{ shift}) \\ B_{44} & B_{41} & B_{42} & B_{43} & (3 \text{ shift}) \end{pmatrix}$$

- ④ MixColumns: Apply invertible transformation to Bytes in each column

- ③ Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & (0 \text{ shift}) \\ B_{22} & B_{23} & B_{24} & B_{21} & (1 \text{ shift}) \\ B_{33} & B_{34} & B_{31} & B_{32} & (2 \text{ shift}) \\ B_{44} & B_{41} & B_{42} & B_{43} & (3 \text{ shift}) \end{pmatrix}$$

- ④ MixColumns: Apply invertible transformation to Bytes in each column

Observations:

- 3 Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & (0 \text{ shift}) \\ B_{22} & B_{23} & B_{24} & B_{21} & (1 \text{ shift}) \\ B_{33} & B_{34} & B_{31} & B_{32} & (2 \text{ shift}) \\ B_{44} & B_{41} & B_{42} & B_{43} & (3 \text{ shift}) \end{pmatrix}$$

- 4 MixColumns: Apply invertible transformation to Bytes in each column

Observations:

- Steps 3 and 4 correspond to permutation step

- 3 Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & (0 \text{ shift}) \\ B_{22} & B_{23} & B_{24} & B_{21} & (1 \text{ shift}) \\ B_{33} & B_{34} & B_{31} & B_{32} & (2 \text{ shift}) \\ B_{44} & B_{41} & B_{42} & B_{43} & (3 \text{ shift}) \end{pmatrix}$$

- 4 MixColumns: Apply invertible transformation to Bytes in each column

Observations:

- Steps 3 and 4 correspond to permutation step
- Even this very structured permutation seems enough