CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. Qi Wang
Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Private-key schemes

- We have seen how to construct schemes based on various lower-level primitives
  - Stream ciphers / PRGs
  - Block ciphers / PRFs
  - Hash functions
- How do we construct these primitives?

Two approaches

- Construct from even lower-level assumptions
  - Can prove secure (given lower-level assumption)
  - Typically inefficient

Two approaches

- Construct from even lower-level assumptions
  - Can prove secure (given lower-level assumption)
  - Typically inefficient

- Build directly
  - Much more efficient!
  - Need to assume security, but
    - We have formal definitions to aim for
    - We can concentrate our analysis on these primitives
    - We can develop/analyze various design principles

Terminology

- *Init* algorithm
  - Takes as input a key + initialization vector (IV)
  - Outputs initial state

- *GetBits* algorithm
  - Takes as input the current state
  - Outputs next bit/byte/chunk and updated state
  - Allows generation of as many bits as needed

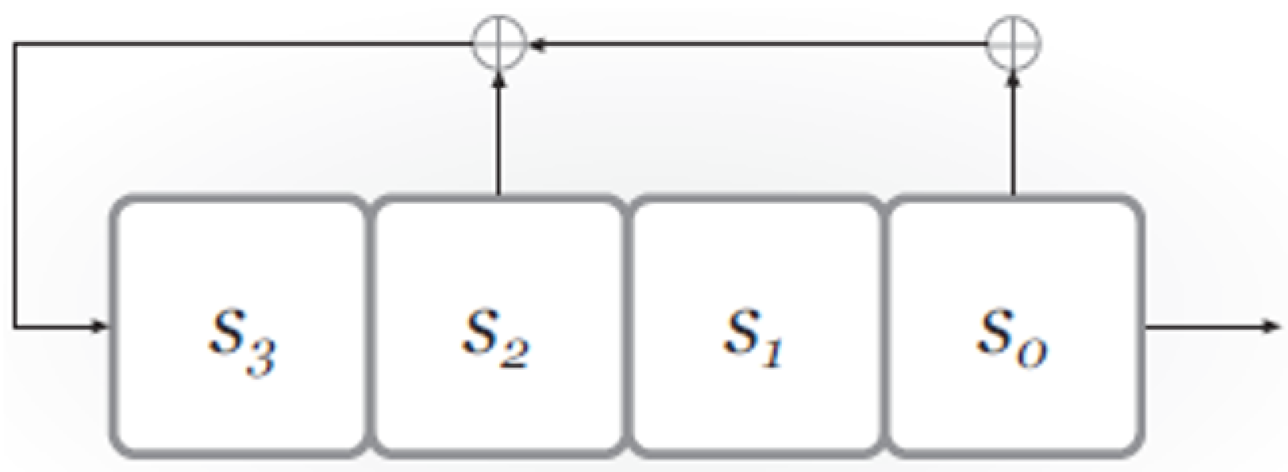
Security requirements

- If there is no *IV*, then (for a uniform key) the output of *GetBits* should be indistinguishable from a uniform, independent stream of bits

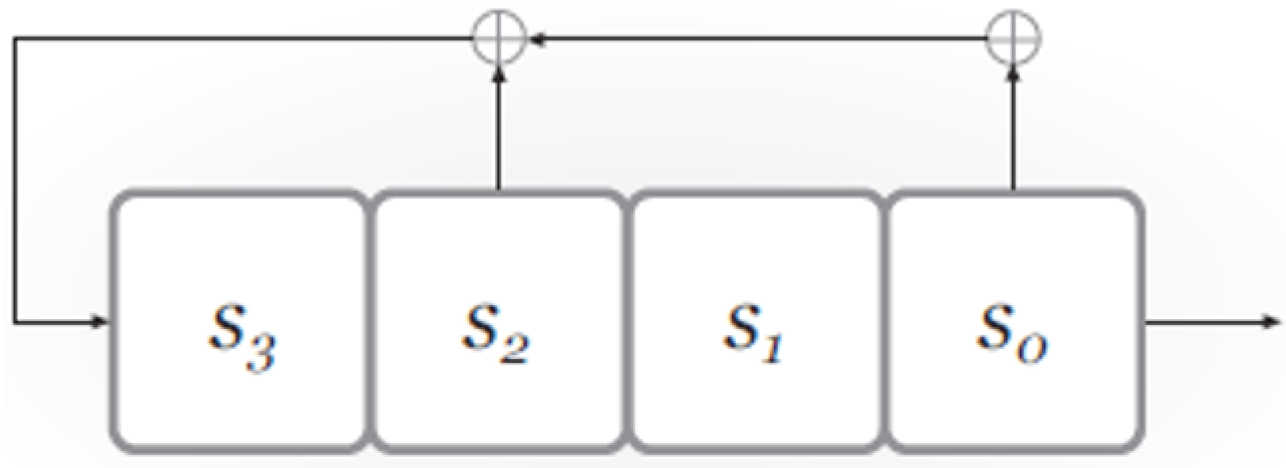
- If there is an *IV*, then (for a uniform key) the output of *GetBits* on multiple, uniform *IV*s should be indistinguishable from multiple uniform, independent streams of bits
  - Even if the attacker is given the *IV*s

LFSRs

- Degree $n \Rightarrow n$ registers
- State: bits $s_{n-1}, \ldots, s_0$ (contents of the registers)
- Feedback coefficients $c_{n-1}, \ldots, c_0$ (view as part of state; do not change)
- State updated and output generated in each "clock tick"



Example



- Assume initial content of registers is 0100
- First 4 state transitions:
  $0100 \rightarrow 1010 \rightarrow 0101 \rightarrow 0010 \rightarrow \ldots$
- First 3 output bits:
  $0\ 0\ 1\ \ldots$

LFSRs as stream ciphers

- Key + *IV* used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated

LFSRs as stream ciphers

- Key + *IV* used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated
- State (and output) "cycles" if state ever repeated
- *Maximal-length LFSR* cycles through all $2^n - 1$ nonzero states
  - Known how to set feedback coefficients so as to achieve maximal length

LFSRs as stream ciphers

- Key + *IV* used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated
- State (and output) "cycles" if state ever repeated
- *Maximal-length LFSR* cycles through all $2^n - 1$ nonzero states
  - Known how to set feedback coefficients so as to achieve maximal length
- *Maximal-length LFSR*s have good statistical properties, but they are not cryptographically secure!

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!

- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!

- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)

$$y_i = s^{(0)}_{i-1}, \qquad i = 1, \ldots, n$$
$$y_i = \bigoplus_{j=0}^{n-1} c_j y_{i-n+j-1}, \qquad i > n$$

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!

- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)

$$y_i = s_{i-1}^{(0)}, \qquad i = 1, \ldots, n$$

$$y_i = \bigoplus_{j=0}^{n-1} c_j y_{i-n+j-1}, \qquad i > n$$

$$\begin{aligned} y_{n+1} &= c_{n-1} y_n \oplus \cdots \oplus c_0 y_1 \\ &\vdots \\ y_{2n} &= c_{n-1} y_{2n-1} \oplus \cdots \oplus c_0 y_n \end{aligned}$$

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!

- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)

$$y_i = s_{i-1}^{(0)}, \qquad i = 1, \ldots, n$$
$$y_i = \bigoplus_{j=0}^{n-1} c_j y_{i-n+j-1}, \qquad i > n$$

$$\begin{aligned} y_{n+1} &= c_{n-1} y_n \oplus \cdots \oplus c_0 y_1 \\ &\vdots \\ y_{2n} &= c_{n-1} y_{2n-1} \oplus \cdots \oplus c_0 y_n \end{aligned}$$

- Linearity is *bad* for cryptography (because linear algebra is so powerful)

Nonlinear FSRs

- Add *nonlinearity* to prevent attacks
  - Nonlinear feedback
  - Output is a nonlinear function of the state
  - Multiple (coupled) LFSRs
  - or any combination of the above

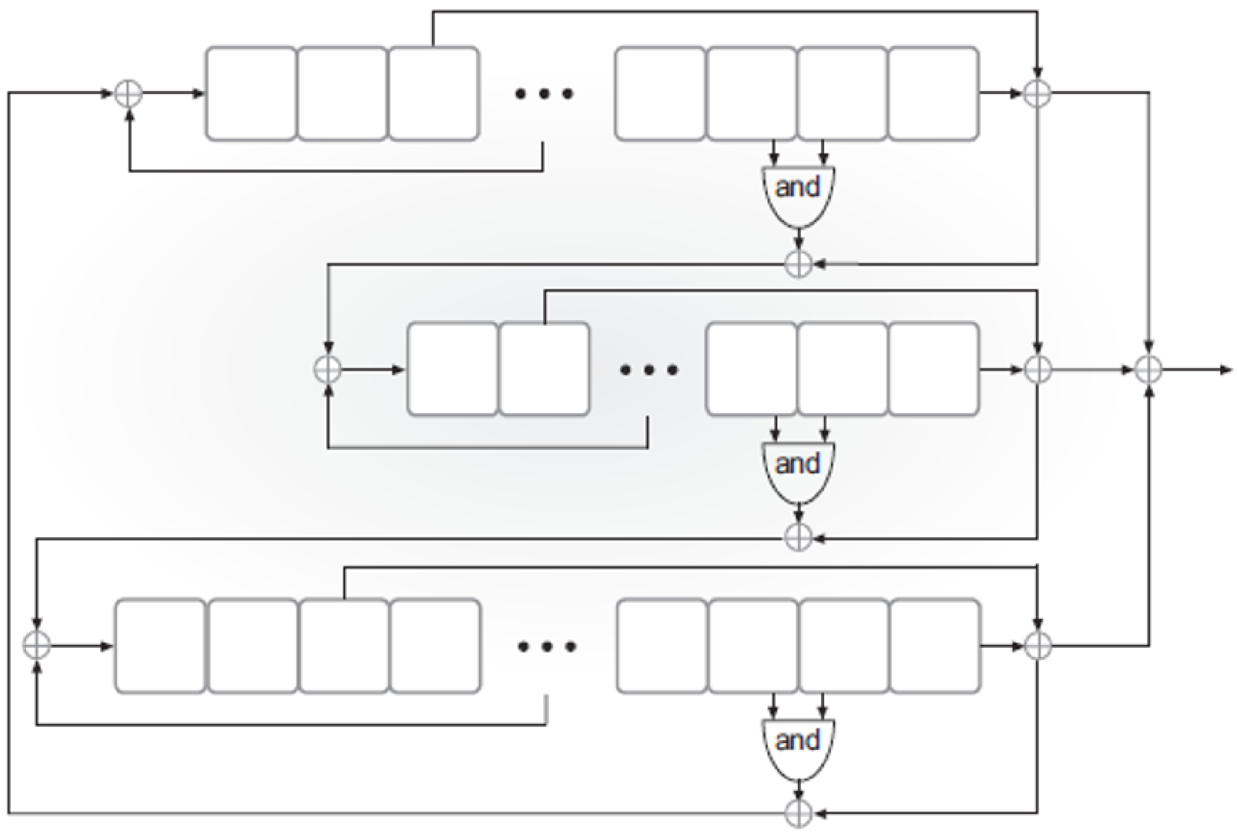
Nonlinear FSRs

- Add *nonlinearity* to prevent attacks
  - Nonlinear feedback
  - Output is a nonlinear function of the state
  - Multiple (coupled) LFSRs
  - or any combination of the above

- Still want to preserve statistical properties of the output, and long cycle length

Example: Trivium

- Designed by De Canniere and Preneel in 2006 as part of eSTREAM competition
- Intended to be simple and efficient (especially in hardware)
- Essentially no attacks better than brute-force search are known



Example: Trivium

- Three FSRs of degree 93, 84, and 111

Example: Trivium

- Three FSRs of degree 93, 84, and 111
- Initialization:
  - 80-bit key in left-most registers of first FSR
  - 80-bit IV in left-most registers of second FSR
  - Remaining registers set to 0, except for three right-most registers of third FSR
  - Run for 4×288 clock ticks

Block ciphers

- Want keyed permutation

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

  – n = key length, ℓ = block length

- Want F_k (for *uniform*, unknown key k) to be *indistinguishable* from a *uniform* permutation over $\{0,1\}^\ell$

Block ciphers

- Want keyed permutation

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

  – n = key length, ℓ = block length

- Want F_k (for *uniform*, unknown key k) to be *indistinguishable* from a *uniform* permutation over $\{0,1\}^\ell$

The security provided by an algorithm is the most important factor. ... Algorithms will be judged on the following factors ...

- *The extent to which the algorithm output is indistinguishable from a random permutation ...*

Attack models

- A *block cipher* is *not* an encryption scheme !!

Attack models

- A *block cipher* is *not* an encryption scheme !!

- Nevertheless, some of the terminology used in the same (for historical reasons)
  - "*known-plaintext attack*": attacker given $\{x, F_k(x)\}$ for random x (outside control of the attacker)

Attack models

- A *block cipher* is *not* an encryption scheme !!

- Nevertheless, some of the terminology used in the same (for historical reasons)
  - "*known-plaintext attack*": attacker given $\{x, F_k(x)\}$ for random x (outside control of the attacker)
  - "*chosen-plaintext attack*": attacker can query $F_k(\cdot)$ (this is the default model we have been using)

Attack models

- A *block cipher* is *not* an encryption scheme !!

- Nevertheless, some of the terminology used in the same (for historical reasons)
  - "*known-plaintext attack*": attacker given $\{x, F_k(x)\}$ for random x (outside control of the attacker)
  - "*chosen-plaintext attack*": attacker can query $F_k(\cdot)$ (this is the default model we have been using)
  - "*chosen-ciphertext attack*": attacker can query $F_k(\cdot)$ and $F_k^{-1}(\cdot)$

Concrete security

- As in the case of stream ciphers, we are interested in *concrete security* for a given key length n
  - Best attack should take time $\approx 2^n$
  - If there is an attack taking time $2^{n/2}$ then the cipher is considered *insecure*

Concrete security

- As in the case of stream ciphers, we are interested in *concrete security* for a given key length n
  - Best attack should take time $\approx 2^n$
  - If there is an attack taking time $2^{n/2}$ then the cipher is considered *insecure*
- Designing block ciphers: Want F_k (for uniform, unknown key k) to be *indistinguishable* from a uniform permutation over $\{0,1\}^\ell$. If x and x' differ in one bit, what should be the relation between $F_k(x)$ and $F_k(x')$?

Concrete security

- As in the case of stream ciphers, we are interested in *concrete security* for a given key length n
  - Best attack should take time $\approx 2^n$
  - If there is an attack taking time $2^{n/2}$ then the cipher is considered *insecure*

- Designing block ciphers: Want F_k (for uniform, unknown key k) to be *indistinguishable* from a uniform permutation over $\{0,1\}^\ell$. If x and x' differ in one bit, what should be the relation between $F_k(x)$ and $F_k(x')$?
  - How many bits should change (on average)?
  - Which bits should change?
  - How to achieve this?

Confusion/diffusion

- "*Confusion*"
  - Small change in input should result in local, "random" change in output

- "*Diffusion*"
  - Local change in input should be propagated to entire output

Design paradigms

- Two design paradigms
  - *Substitution-permutation networks* (SPNs)
  - *Feistel networks*

Design paradigms

- Two design paradigms
  - *Substitution-permutation networks* (SPNs)
  - *Feistel networks*

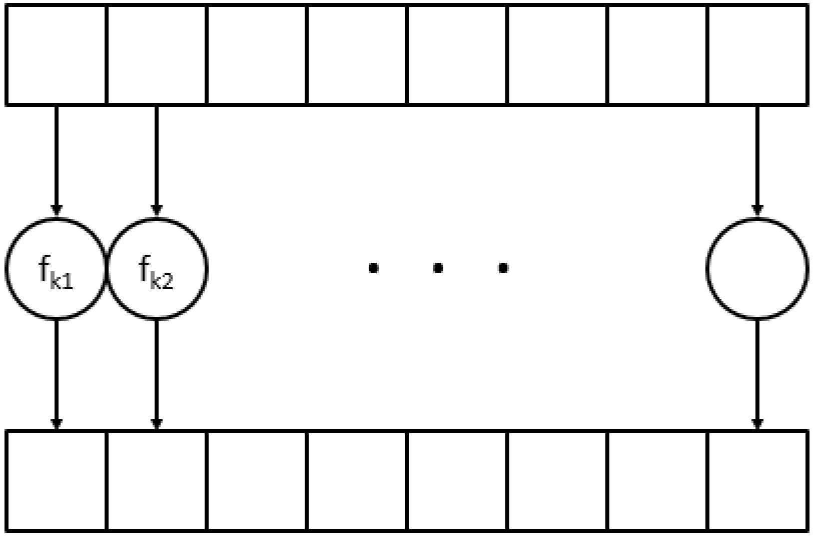
- SPNs: build "random-looking" permutation on large input from random permutations on small inputs

Design paradigms

- Two design paradigms
  - *Substitution-permutation networks* (SPNs)
  - *Feistel networks*
- SPNs: build "random-looking" permutation on large input from random permutations on small inputs
  - E.g., assume 8-byte block length
    $$F_k(x) = f_{k_1}(x_1) f_{k_2}(x_2) \cdots f_{k_8}(x_8),$$
    where each f is a random permutation
  - How long is k?

Design paradigms

- Two design paradigms
  - *Substitution-permutation networks* (SPNs)
  - *Feistel networks*
- SPNs: build "random-looking" permutation on large input from random permutations on small inputs



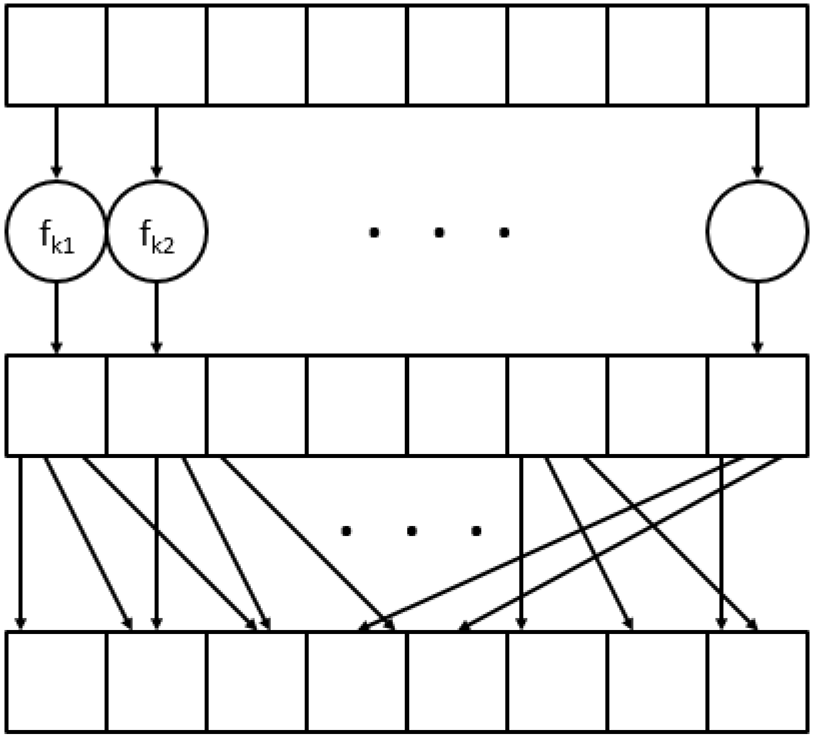
Is this a pseudorandom function?

SPN

- This has *confusion* but no diffusion
  - Add a *mixing permutation*

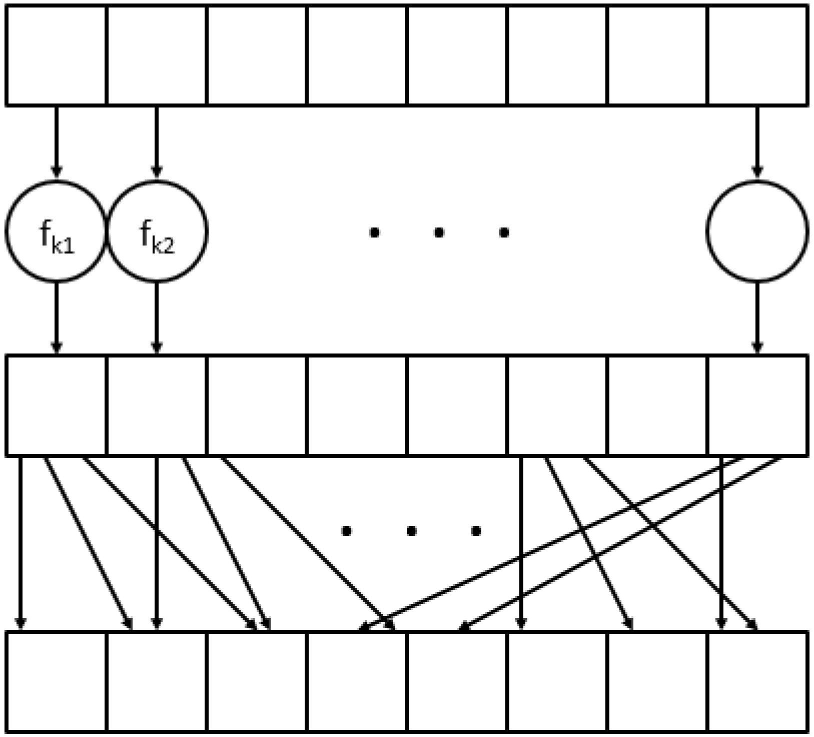
SPN

- This has *confusion* but no diffusion
  - Add a *mixing permutation*



SPN

- This has *confusion* but no diffusion
  - Add a *mixing permutation*



Note that the structure is *invertible* (given the key) since the f's are permutations

SPN

- Mixing permutation is public
  - Chosen to ensure good diffusion

SPN

- Mixing permutation is public
  - Chosen to ensure good diffusion

- Does this give a pseudorandom function?

SPN

- Mixing permutation is public
  - Chosen to ensure good diffusion

- Does this give a pseudorandom function?

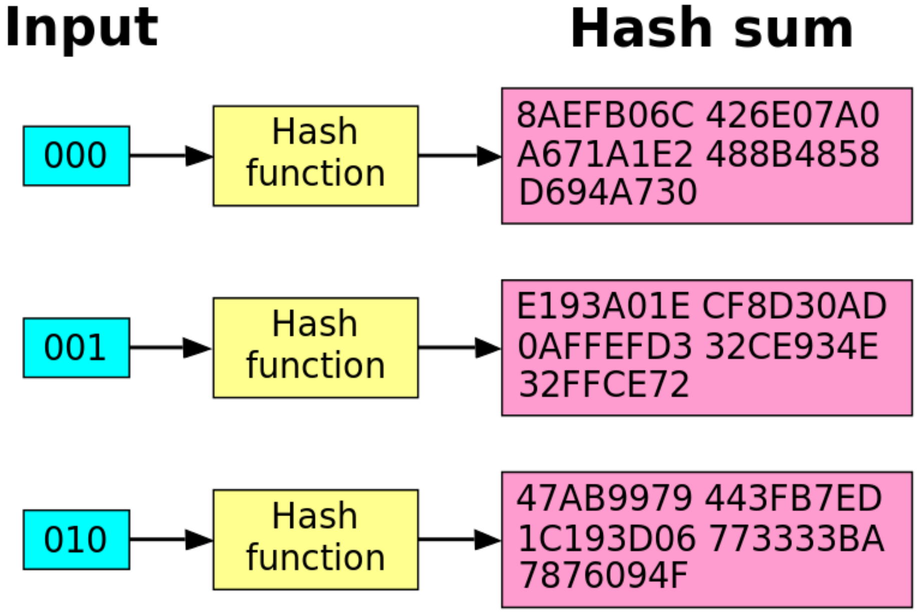
- What if we repeat for another round (with independent, random functions)?
  - What is the *minimal* # of rounds we need?
  - *Avalanche effect*

SPN

- Mixing permutation is public
  - Chosen to ensure good diffusion

- Does this give a pseudorandom function?

- What if we repeat for another round (with independent, random functions)?
  - What is the *minin*
  - *Avalanche effect*



SPN

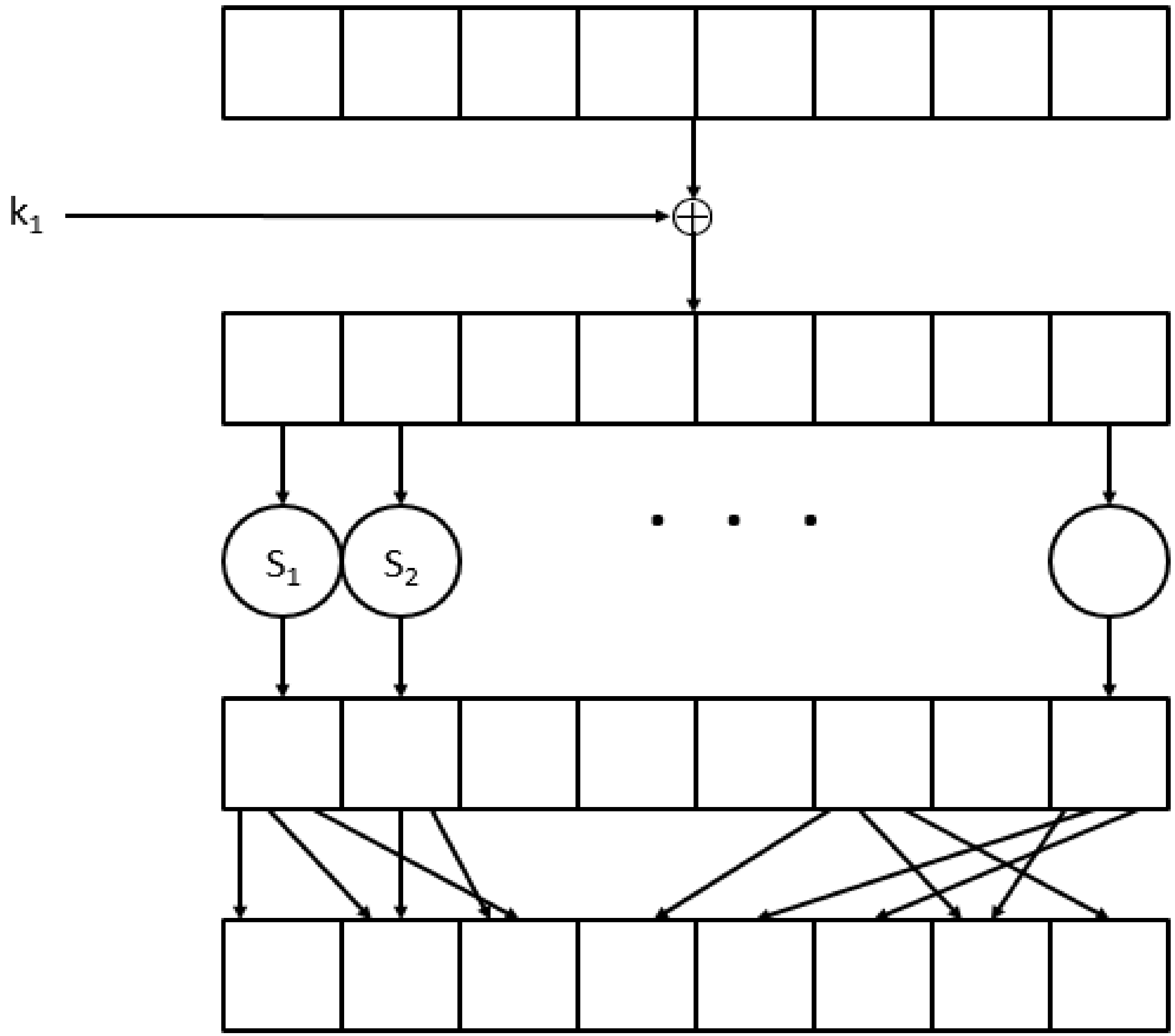
- Using random f's is not practical
  - Key would be too large

SPN

- Using random f's is not practical
  - Key would be too large

- Instead, use f's of a particular form
  - $f_{k_i}(x) = S_i(k_i \oplus x)$, where S_i is a public permutation
  - S_i are called "*S-boxes*" (*substitution boxes*)
  - XORing the key is called "*key mixing*"
  - Note that this is still invertible (given the key)

SPN



Avalanche effect

- Design S-boxes and mixing permutation to ensure *avalanche effect*
  - Small differences should eventually propagate to entire output

Avalanche effect

- Design S-boxes and mixing permutation to ensure *avalanche effect*
  - Small differences should eventually propagate to entire output
- S-boxes: 1-bit change in input causes $\geq$ 2-bit change in output
  - Not so easy to ensure!

Avalanche effect

- Design S-boxes and mixing permutation to ensure *avalanche effect*
  - Small differences should eventually propagate to entire output
- S-boxes: 1-bit change in input causes $\geq$ 2-bit change in output
  - Not so easy to ensure!
- Mixing permutation
  - Each bit output from a given S-box should feed into a *different* S-box in the next round

SPN

- One round of an SPN involves
  - Key mixing
    - Ideally, round keys are independent
    - In practice, derived from a master key via *key schedule*
  - Substitution (S-boxes)
  - Permutation (mixing permutation)

SPN

- One round of an SPN involves
  - Key mixing
    - Ideally, round keys are independent
    - In practice, derived from a master key via *key schedule*
  - Substitution (S-boxes)
  - Permutation (mixing permutation)
- r-round SPN has r rounds as above, plus a final key-mixing step
  - Why?

SPN

- One round of an SPN involves
  - Key mixing
    - Ideally, round keys are independent
    - In practice, derived from a master key via *key schedule*
  - Substitution (S-boxes)
  - Permutation (mixing permutation)
- r-round SPN has r rounds as above, plus a final key-mixing step
  - Why?
- *Invertible* regardless of how many rounds

Key-recovery attacks

- *Key-recovery attacks* are even more damaging than distinguishing attacks
  - As before, a cipher is *secure* only if the best key-recovery attack takes time $\approx 2^n$
  - A fast key-recovery attack represents a "complete attack" of the cipher

Key-recovery attack, 1-round SPN

- Consider first the case where there is no final key-mixing step
  - Possible to get the key immediately!

Key-recovery attack, 1-round SPN

- Consider first the case where there is no final key-mixing step
  - Possible to get the key immediately!
- What about a full 1-round SPN?
  - Attack 1: for each possible 1^{st}-round key, get corresponding 2^{nd}-round key
    - Continue process of elimination
    - Complexity $\approx 2^{\ell}$ for key of length 2ℓ

Key-recovery attack, 1-round SPN

- Consider first the case where there is no final key-mixing step
  - Possible to get the key immediately!
- What about a full 1-round SPN?
  - Attack 1: for each possible 1^{st}-round key, get corresponding 2^{nd}-round key
    - Continue process of elimination
    - Complexity $\approx 2^{\ell}$ for key of length 2ℓ
- Better attack: work S-box-by-S-box
  - Assume 8-bit S-box
  - For each 8 bits of 1^{st}-round key, get corresponding 8 bits of 2^{nd}-round key
    - Continue process of elimination
    - Complexity?

Key-recovery attack, 1-round SPN

- Consider first the case where there is no final key-mixing step
  - Possible to get the key immediately!
- What about a full 1-round SPN?
  - Attack 1: for each possible 1^{st}-round key, get corresponding 2^{nd}-round key
    - Continue process of elimination
    - Complexity $\approx 2^{\ell}$ for key of length 2ℓ
- Better attack: work S-box-by-S-box
  - Assume 8-bit S-box
  - For each 8 bits of 1^{st}-round key, get corresponding 8 bits of 2^{nd}-round key
    - Continue process of elimination
    - Complexity?

Lindell & Katz p.209-211

Feistel networks

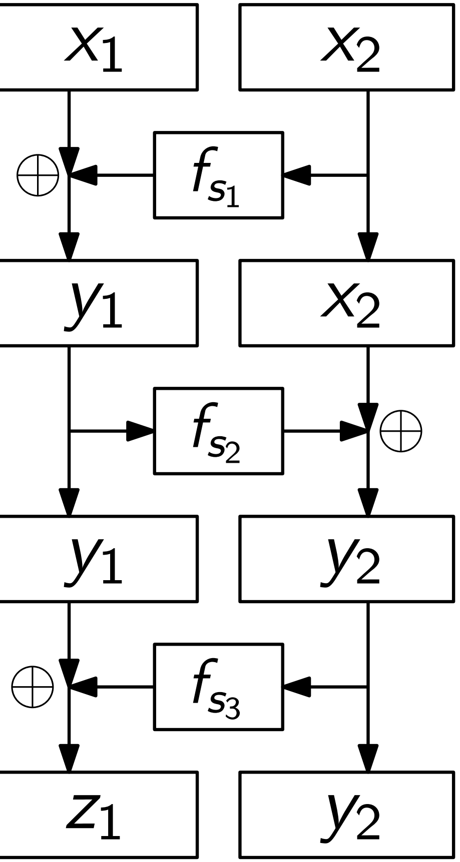
- Build (invertible) permutation from non-invertible components

- One round:
  - Keyed round function f: $\{0,1\}^n \times \{0,1\}^{\ell/2} \to \{0,1\}^{\ell/2}$
  - $F_{k_1}(L0, R0) \to (L1, R1)$
    where $L1 = R0$; $R1 = L0 \oplus f_{k_1}(R0)$

- Always invertible!

Luby-Rackoff construction

- This is so-called *Luby-Rackoff construction*, using several rounds of *Feistel Transformation*.



We build a PRP p on $2n$ bits from three PRFs $f_{s_1}, f_{s_2}, f_{s_3}$ on n bits by letting

$$p_{s_1,s_2,s_3}(x_1, x_2) = (z_1, y_2)$$

where $y_1 = x_1 \oplus f_{s_1}(x_2)$, $y_2 = x_2 \oplus f_{s_2}(y_1)$, and $z_1 = f_{s_3}(y_2) \oplus y_1$.

Security

- Security of 1-round Feistel?

- Security of 2-round Feistel (with independent keys)?

- Security of 3/4-round Feistel?

Security

- Security of 1-round Feistel?

- Security of 2-round Feistel (with independent keys)?

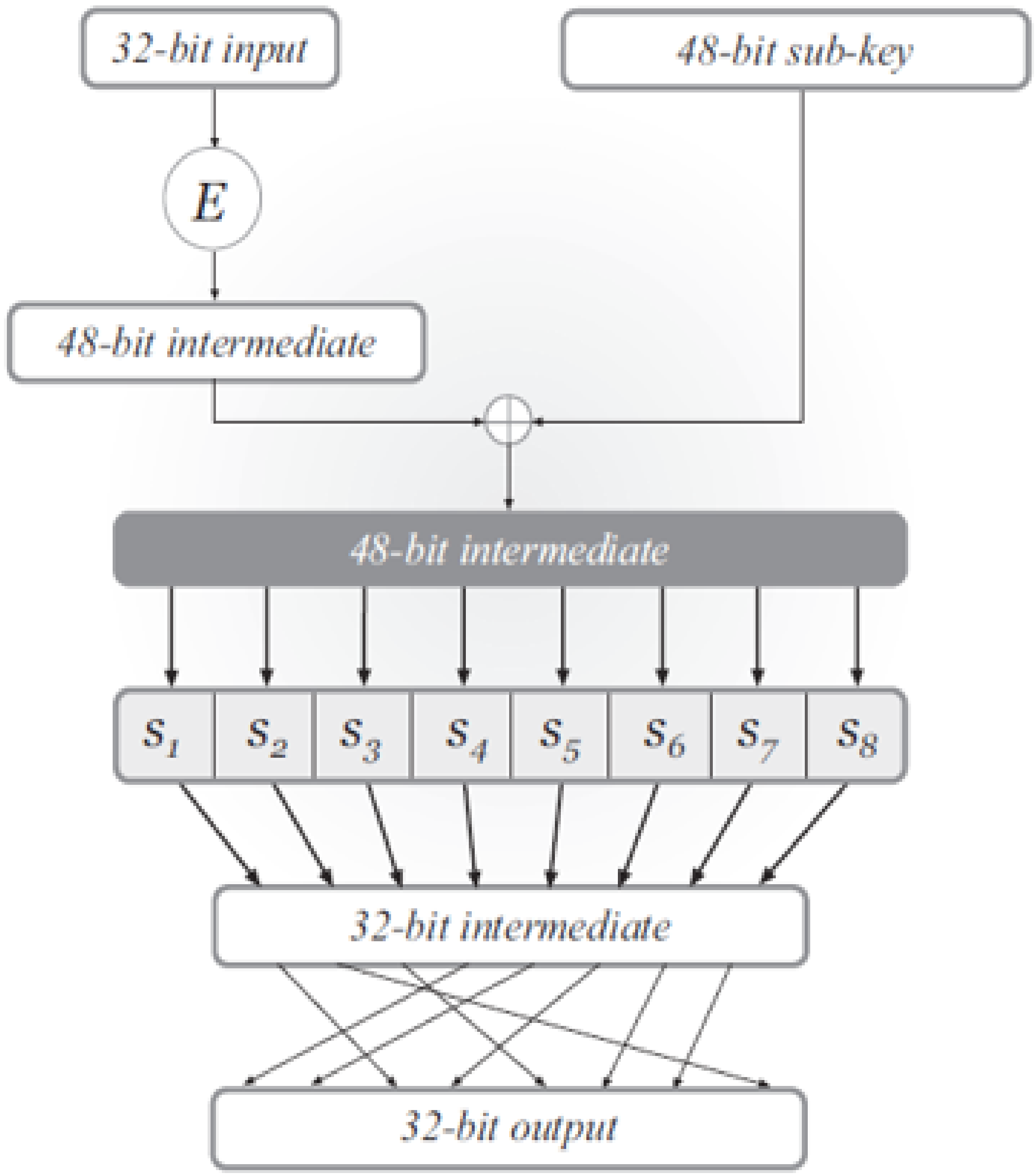
- Security of 3/4-round Feistel?

  Lindell & Katz p.216-218

Data Encryption Standard (DES)

- Standardized in 1977
- 56-bit keys, 64-bit block length
- 16-round Feistel network
  - Same round function in all rounds (but different sub-keys)
  - Basically an SPN design!

DES mangler function



DES mangler function

- S-boxes
  - Each S-box is 4-to-1
  - Changing 1 bit of input changes at least 2-bits of output
- Mixing permutation
  - The 4 bits of output from any S-box affect the input to 6 S-boxes in the next round

Key schedule + Avalanche effect

- 56-bit master key, 48-bit subkey in each round
  - Each subkey takes 24 bits from the left half of the master key, and 24 bits from the right half of the master key

Key schedule + Avalanche effect

- 56-bit master key, 48-bit subkey in each round
  – Each subkey takes 24 bits from the left half of the master key, and 24 bits from the right half of the master key

- Consider 1-bit difference in left half of input
  – After 1 round, 1-bit difference in right half
  – S-boxes cause a 2-bit difference, implying a 3-bit difference overall after 2 rounds
  – Mixing permutation spreads differences into different S-boxes

Security of DES

- DES is extremely well-designed
  - Except for some attacks that require large amounts of plaintext, no attacks better than brute-force are known

Security of DES

- DES is extremely well-designed
  – Except for some attacks that require large amounts of plaintext, no attacks better than brute-force are known

- But, parameters are too small!
  – I.e., brute-force search is feasible

56-bit key length

- A concern as soon as DES was released
- Brute-force search over 2^{56} keys is possible
  - 1997: 1000s of computers, 96 days
  - 1998: distributed.net, 41 days
  - 1999: Deep Crack ($\$250,000$), 56 hours
  - Today: 48 FPGAs, about 1 day

64-bit block length

- Birthday collisions relatively likely

- E.g., encrypt 2^{30} (≈ 1 billion) records using CTR mode; chances of a collision are

$$\approx 2^{60}/2^{64} = 1/16$$

Increasing key length?

- DES has key that is too short

- How to fix?
  - Design new cipher
  - Tweak DES so that it takes a larger key
  - Build new cipher using DES as a black box

Double encryption

- Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$
  - i.e., $n = 56$, $\ell = 64$ for DES

Double encryption

- Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$
  – i.e., $n = 56$, $\ell = 64$ for DES

- Define $F^2 : \{0,1\}^{2n} \times \{0,1\}^\ell \to \{0,1\}^\ell$ as follows:
  $$F^2_{k_1,k_2}(x) = F_{k_1}(F_{k_2}(x))$$
  (still invertible)

Double encryption

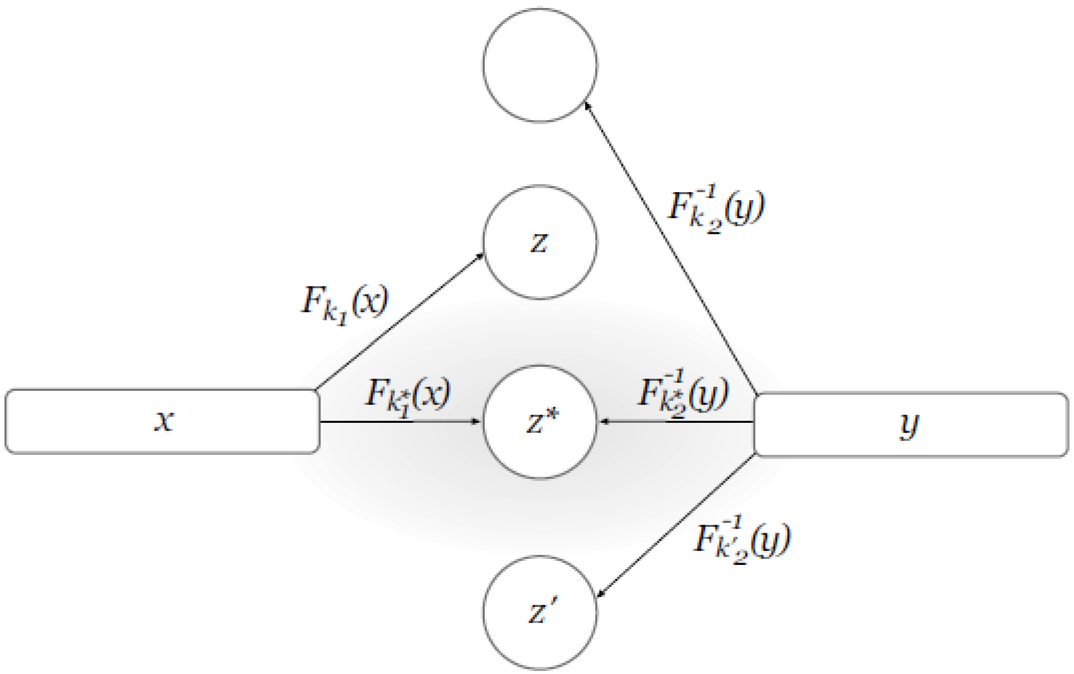
- Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$
  - i.e., $n = 56$, $\ell = 64$ for DES

- Define $F^2 : \{0,1\}^{2n} \times \{0,1\}^\ell \to \{0,1\}^\ell$ as follows:
$$F^2_{k_1,k_2}(x) = F_{k_1}(F_{k_2}(x))$$
(still invertible)

- If best attack on F takes time 2^n, is it reasonable to assume that the best attack on F^2 takes time 2^{2n}?

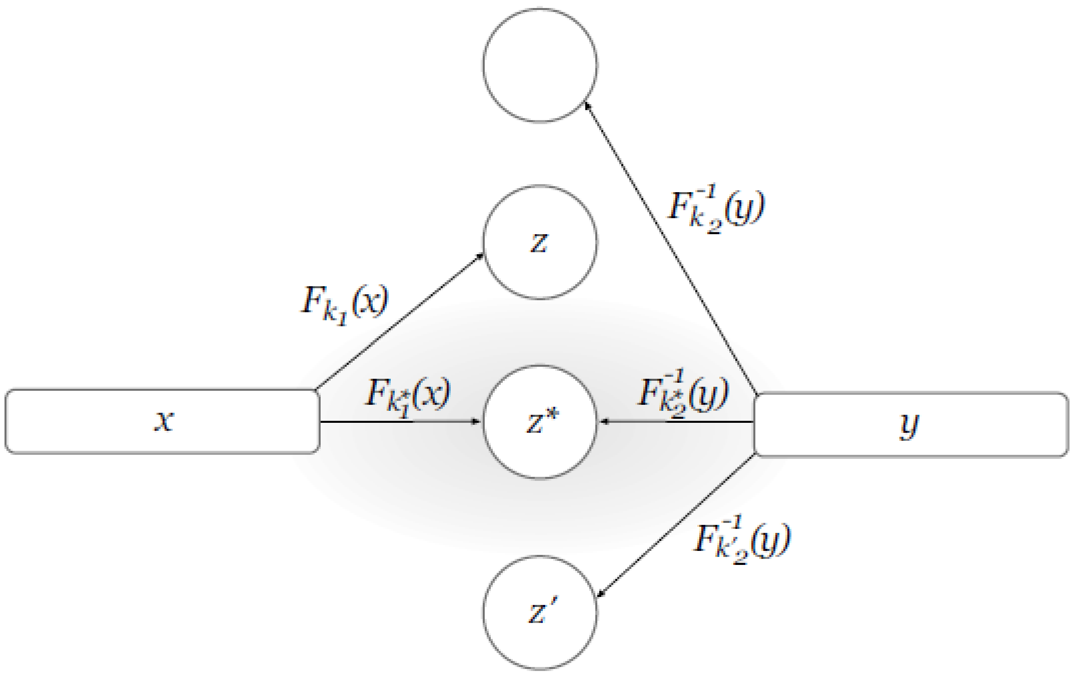
Meet-in-the-middle attack

- No! There is an attack taking 2^n time
  - And 2^n memory



Meet-in-the-middle attack

- No! There is an attack taking 2^n time
  - And 2^n memory



- The attack applies any time a block cipher can be "factored" into 2 independent components

Triple encryption

- Define $F^3 : \{0,1\}^{3n} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ as follows:
$$F^3_{k_1,k_2,k_3}(x) = F_{k_1}(F_{k_2}(F_{k_3}(x)))$$

- What is the best attack now?

Triple encryption

- Define $F^3 : \{0,1\}^{3n} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ as follows:
$$F^3_{k_1,k_2,k_3}(x) = F_{k_1}(F_{k_2}(F_{k_3}(x)))$$

- What is the best attack now?

- Best attacks take time 2^{2n} – optimal given the key length!

- This approach is taken by triple-DES

Advanced encryption standard (AES)

- Public design competition run by NIST
- Began in Jan 1997
  - 15 algorithms submitted

Advanced encryption standard (AES)

- Public design competition run by NIST
- Began in Jan 1997
  - 15 algorithms submitted
- Workshops in 1998, 1999
  - Narrowed to 5 finalists

Advanced encryption standard (AES)

- Public design competition run by NIST
- Began in Jan 1997
  - 15 algorithms submitted
- Workshops in 1998, 1999
  - Narrowed to 5 finalists
- Workshop in early 2000; winner announced in late 2000
  - Factors besides security taken into account

AES

- 128-bit block length
- 128-, 192-, and 256-bit key lengths

AES

- 128-bit block length
- 128-, 192-, and 256-bit key lengths

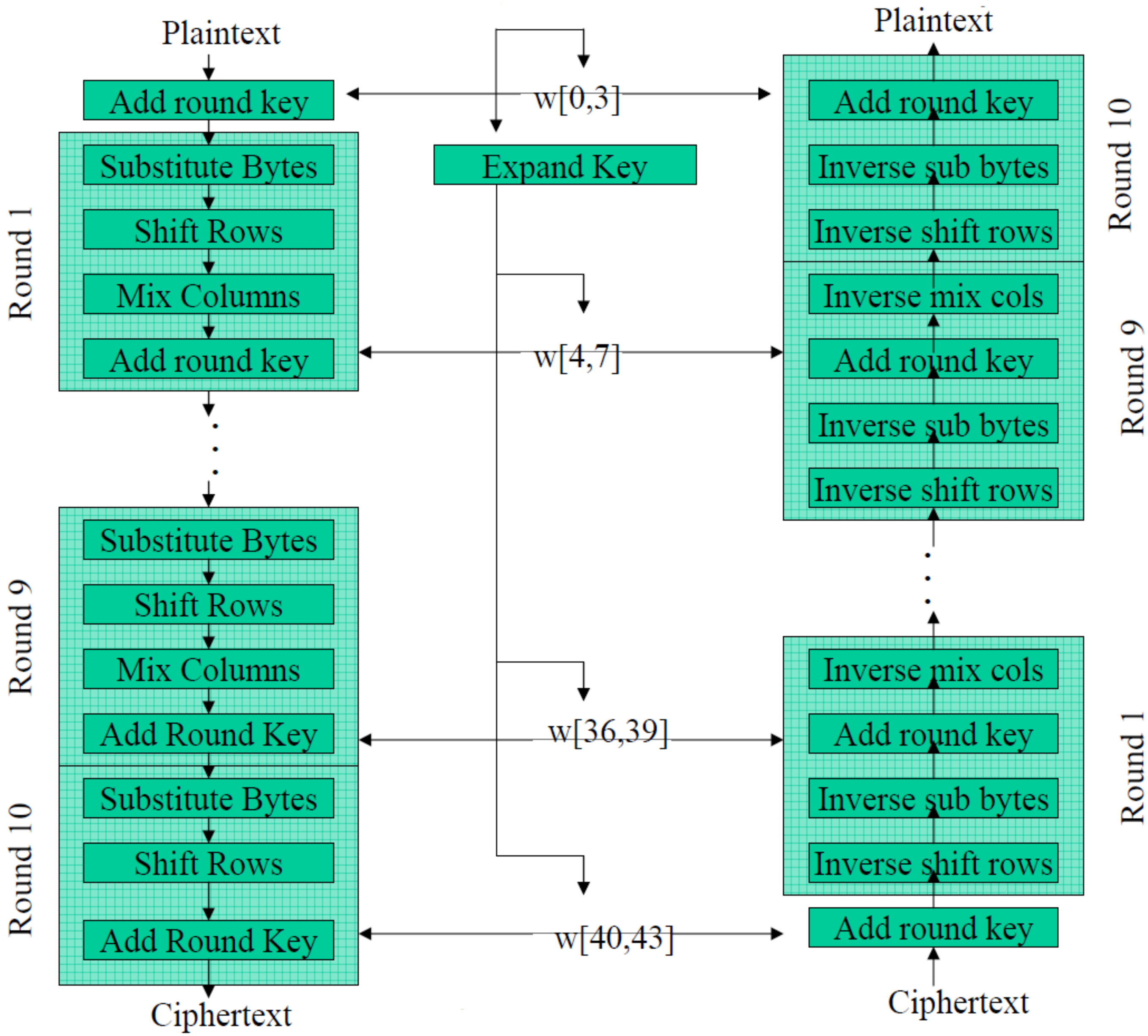
- Basically an SPN structure!
  - 1-byte S-box (same for all bytes)
  - Mixing permutation replaced by invertible linear transformation

- No attacks better than brute-force known

Rijndael: Key and Block Size

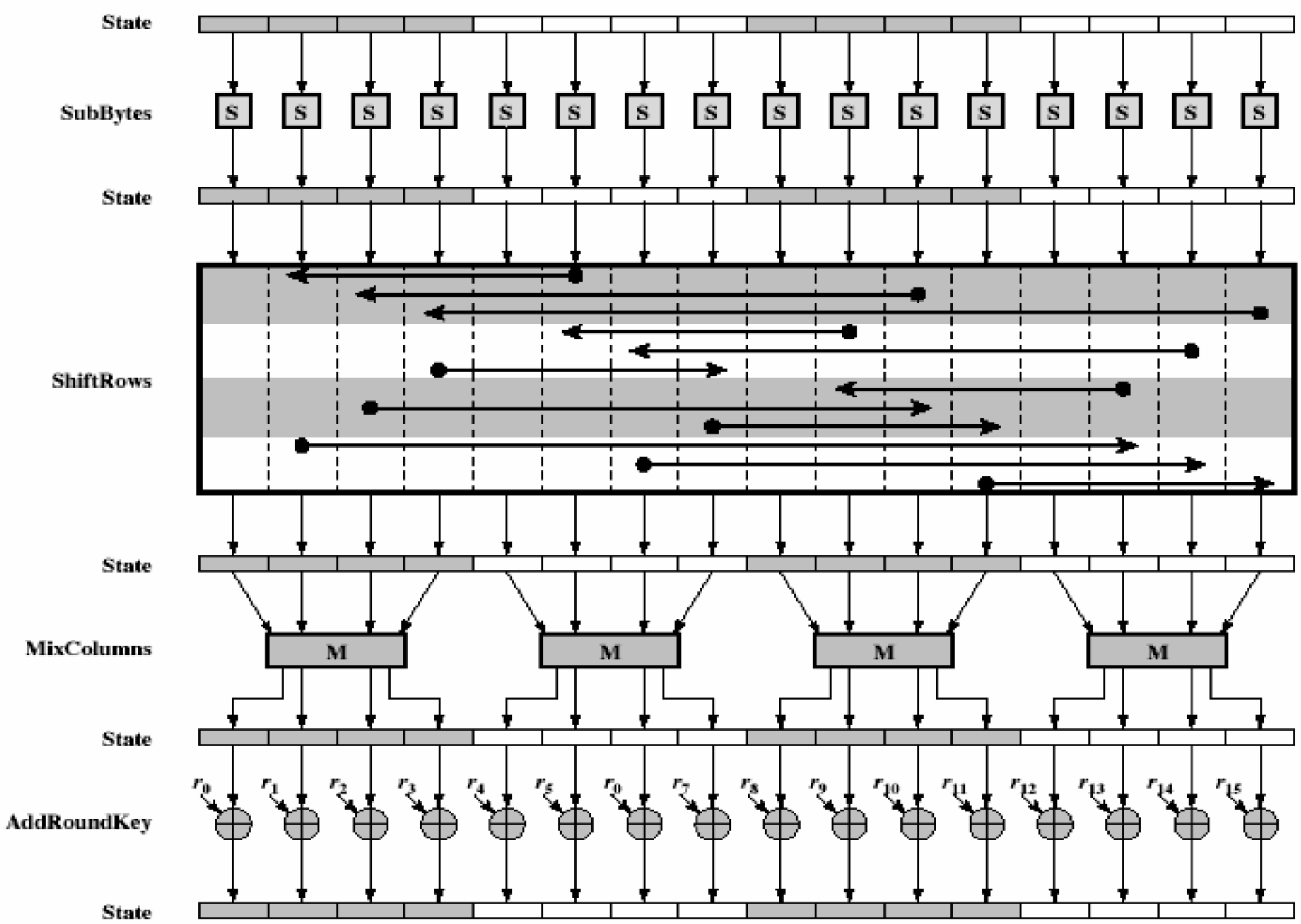
Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

AES Encryption & Decryption



- ◇ An initial round-key addition.
- ◇ 9/11/13 rounds, corresponds to 128/192/256 bit keys
- ◇ A final round, similar to other rounds, but without mixed column opertations

AES Round Function



Key and State Bytes in Rectangular Arrays

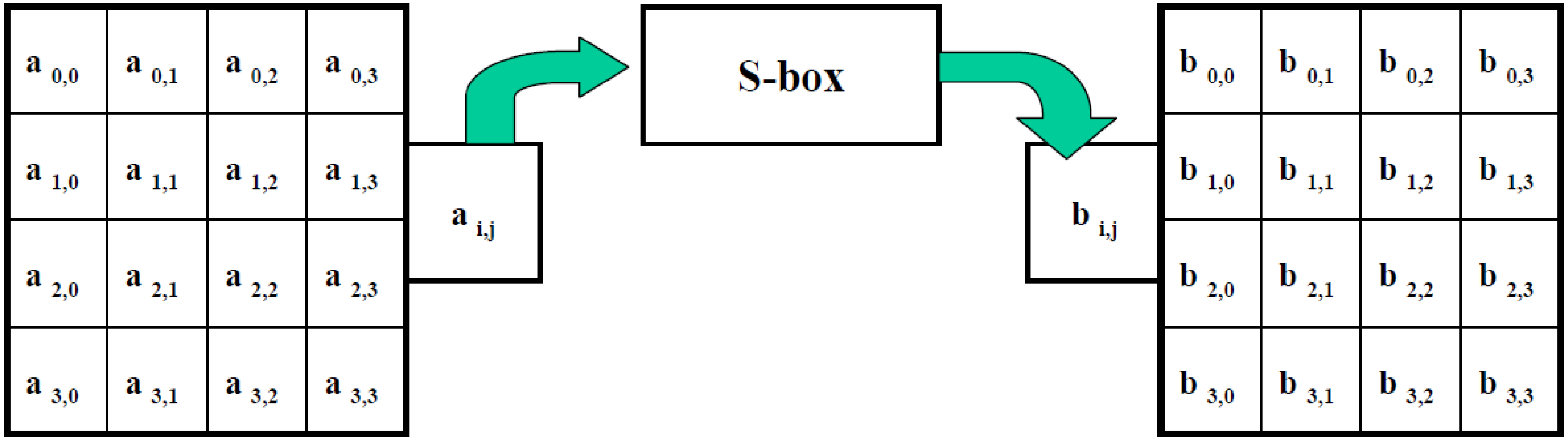
$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

Variable key size:
16/24/32 bytes

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

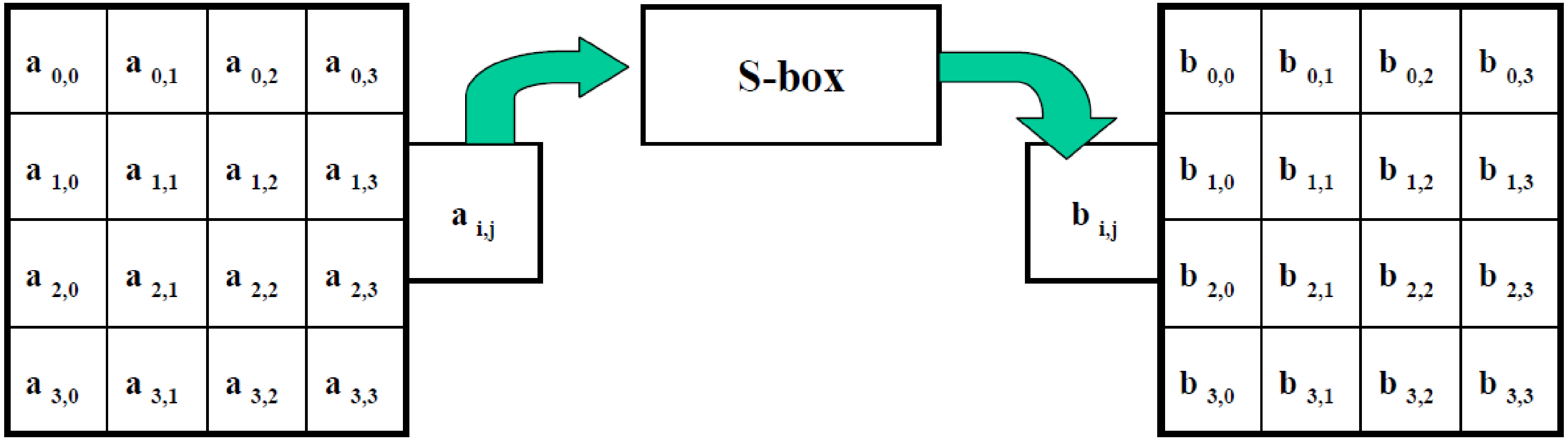
Variable State size:
16/24/32 bytes

AES Round Function: ByteSub



ByteSub acts on individual bytes of the State (only 1 S-box 8×8)

AES Round Function: ByteSub

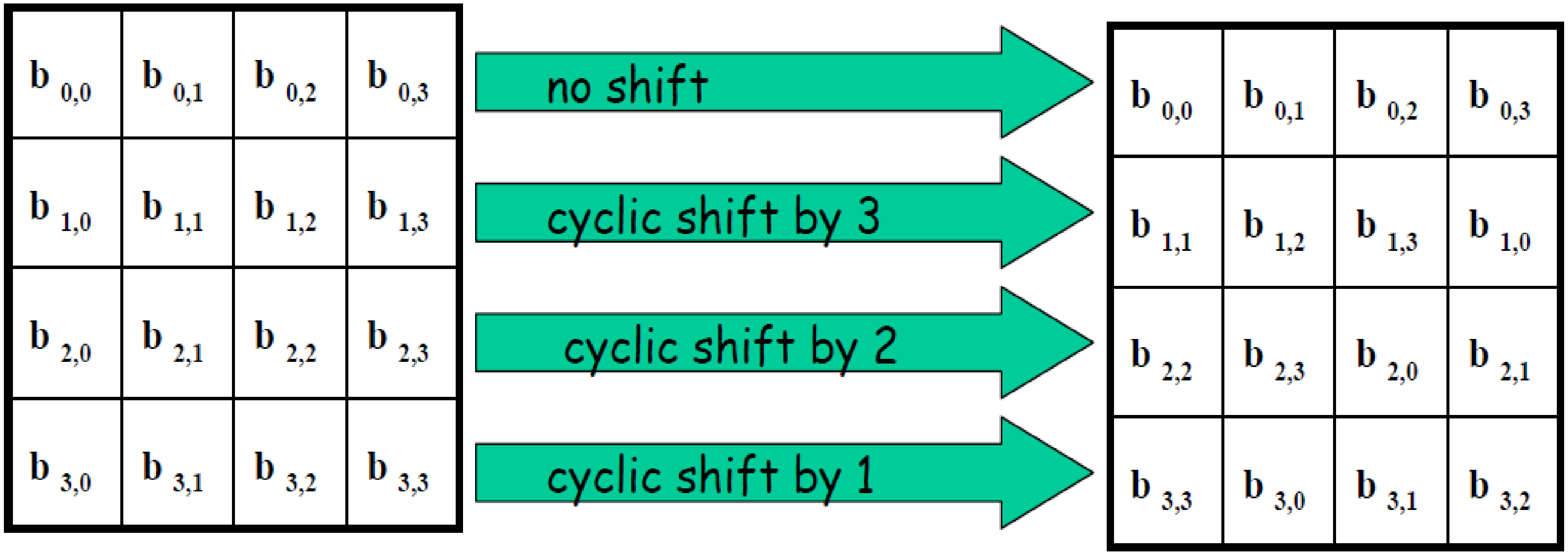


ByteSub acts on individual bytes of the State (only 1 S-box 8×8)

ByteSub is a (the only) non-linear byte substitution by the composition of two transformations:

1. take *multiplicative inverse* in $\mathbb{F}_{2^8}$ $(0 \mapsto 0)$
2. apply an *affine* (over $\mathbb{F}_2$) mapping to each byte.

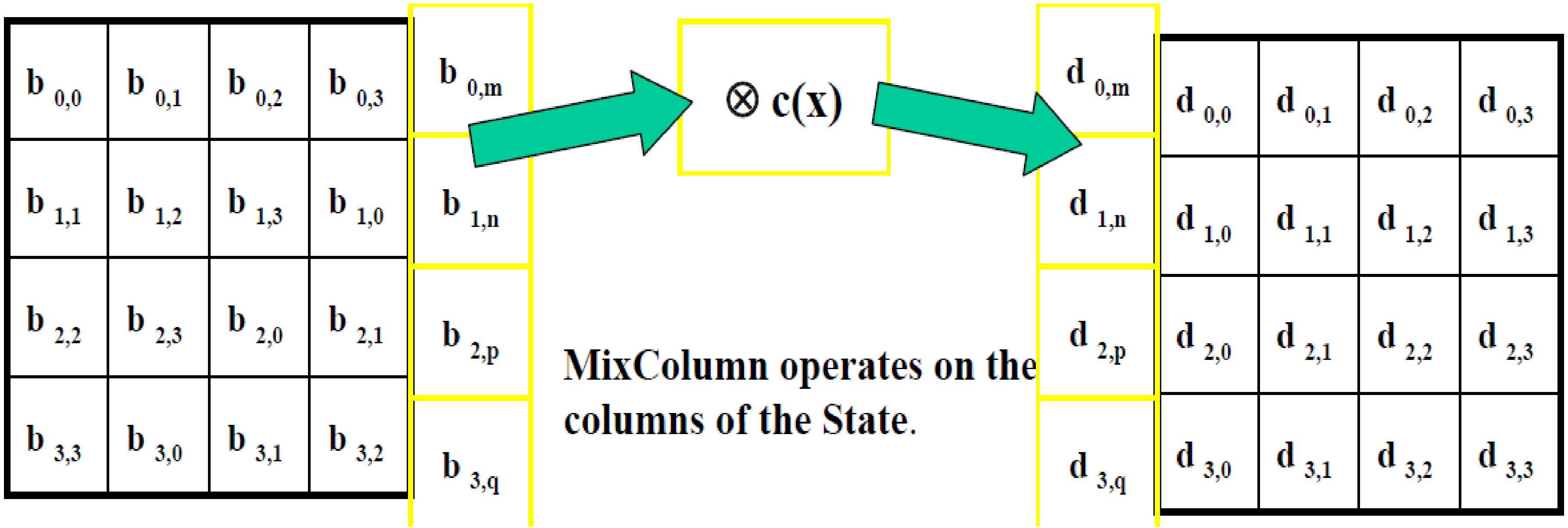
AES Round Function: ShiftRow

- 

ShiftRow operates on the rows of the State

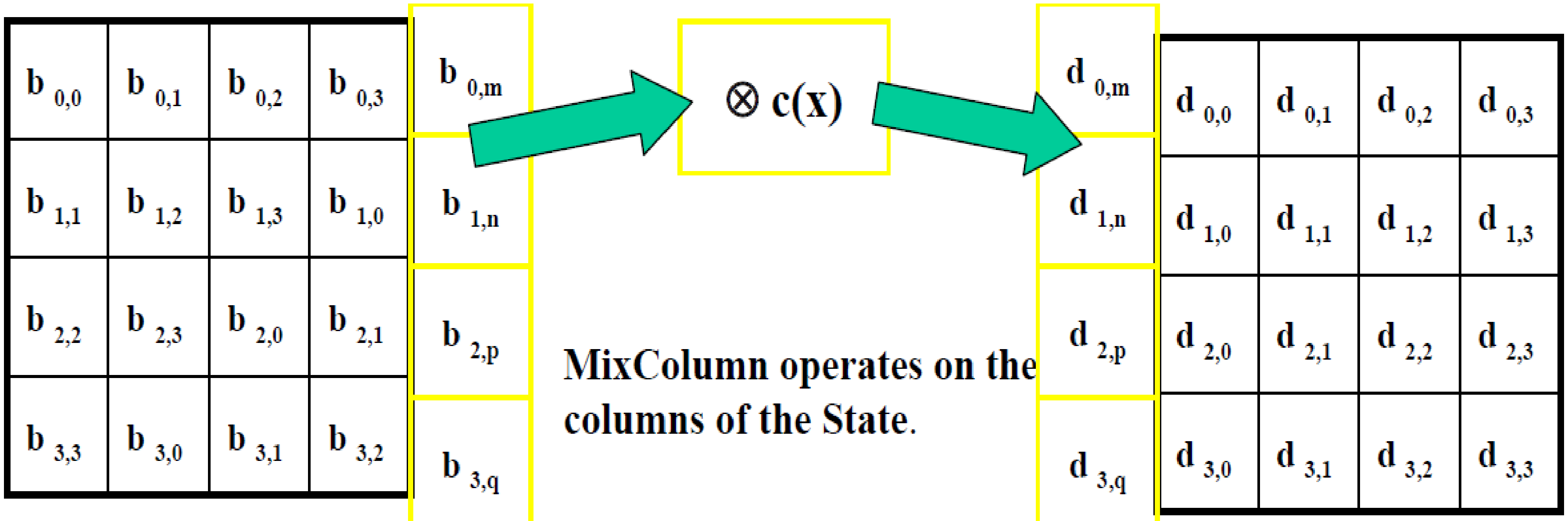
Purpose: inter-column *diffusion*

AES Round Function: MixColumn



MixColumn is implemented using XOR operations. The columns of the State are considered as polynomials of degree 3 over $\mathbb{F}_{2^8}$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$:
$c(x) = 03x^3 + 01x^2 + 01x + 02$.

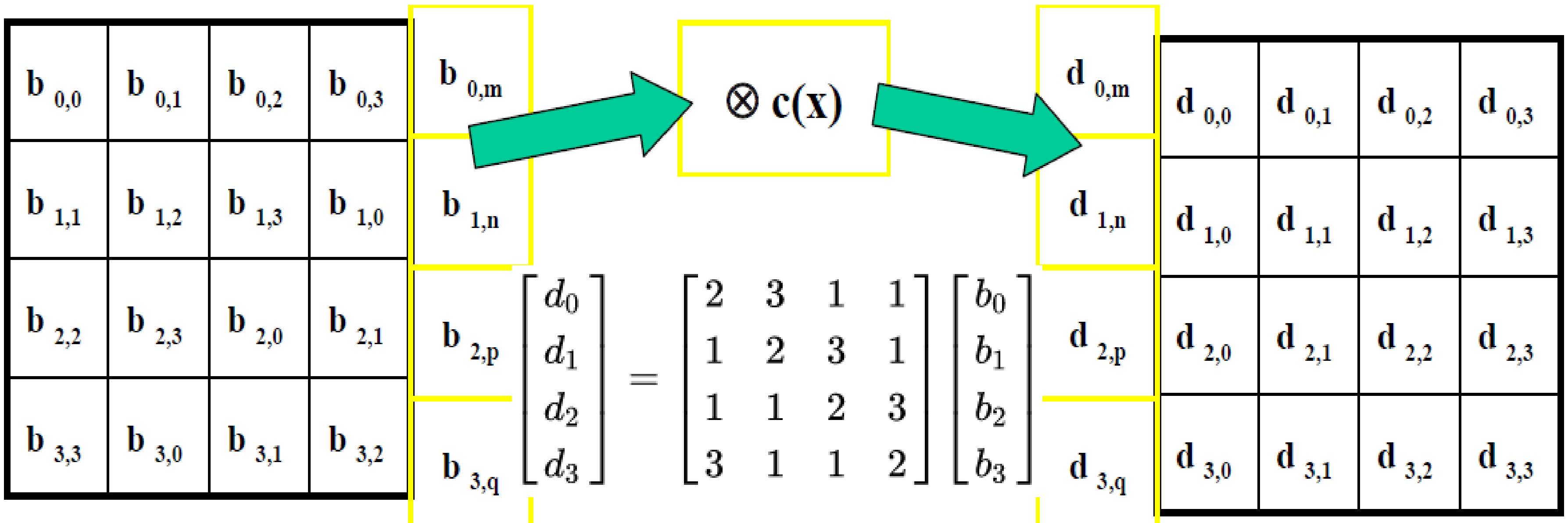
AES Round Function: MixColumn



MixColumn is implemented using XOR operations. The columns of the State are considered as polynomials of degree 3 over $\mathbb{F}_{2^8}$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$:
$c(x) = 03x^3 + 01x^2 + 01x + 02$.

Purpose: inter-byte *diffusion*. Together with ShiftRow, it ensures that after a few rounds, all output bits depend on all input bits.

AES Round Function: MixColumn



MixColumn is implemented using XOR operations. The columns of the State are considered as polynomials of degree 3 over $\mathbb{F}_{2^8}$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$: $c(x) = 03x^3 + 01x^2 + 01x + 02$.

Purpose: inter-byte *diffusion*. Together with ShiftRow, it ensures that after a few rounds, all output bits depend on all input bits.

AES Round Function: AddRoundKey

$$\begin{bmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\ e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}$$

In *AddRoundKey*, the Round Key is bitwise XORed to the State.

AES Round Function: AddRoundKey

$$\begin{bmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\ e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}$$

In *AddRoundKey*, the Round Key is bitwise XORed to the State.

Purpose: makes round function *key-dependent*.

AES Round Function: AddRoundKey

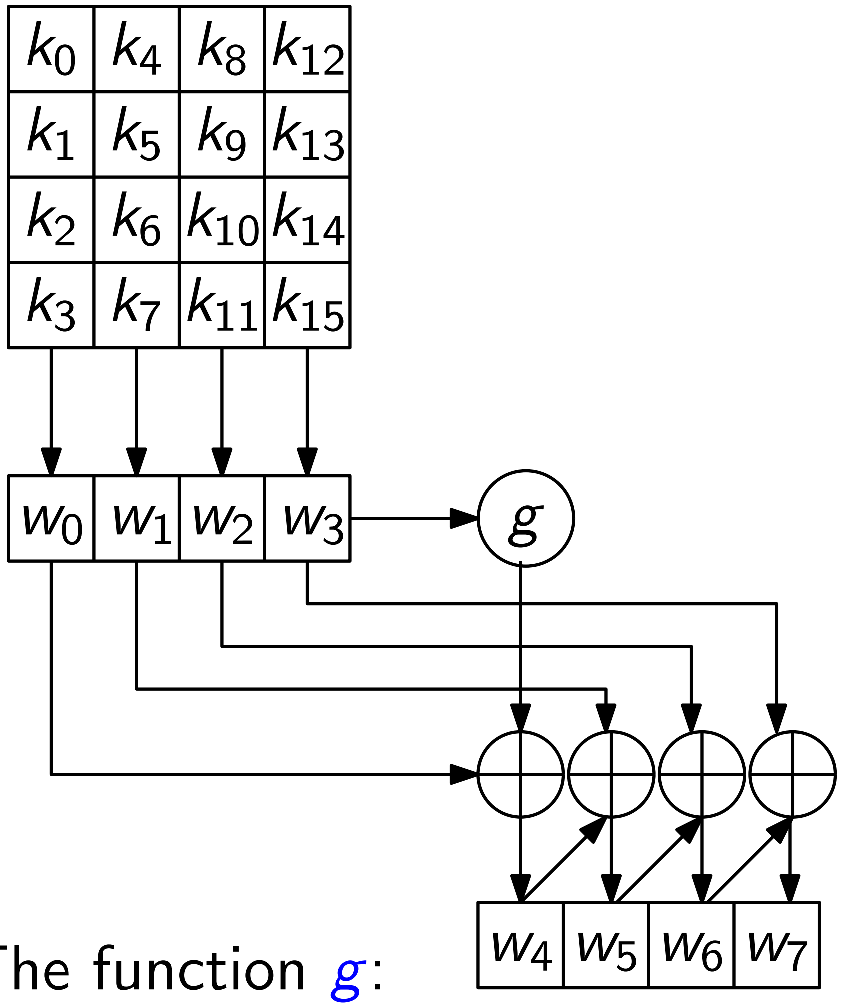
$$
\begin{bmatrix}
d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\
d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\
d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\
d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3}
\end{bmatrix}
\oplus
\begin{bmatrix}
k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\
k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\
k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\
k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3}
\end{bmatrix}
=
\begin{bmatrix}
e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\
e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\
e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\
e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3}
\end{bmatrix}
$$

In *AddRoundKey*, the Round Key is bitwise XORed to the State.

Purpose: makes round function *key-dependent*.

Key-XORing with plaintext or ciphertext is called *whitening*. This is a cheap way of adding to the security of cipher by preventing the collection of plaintext-ciphertext pairs.

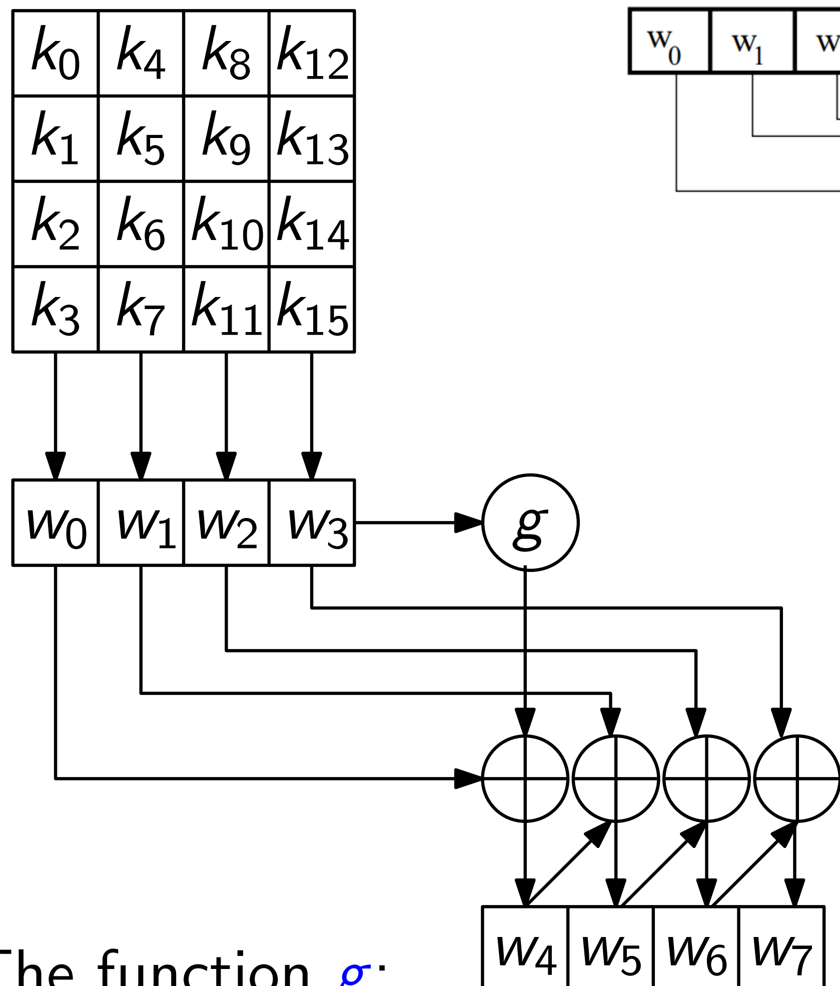
AES Round Function: Key Expansion

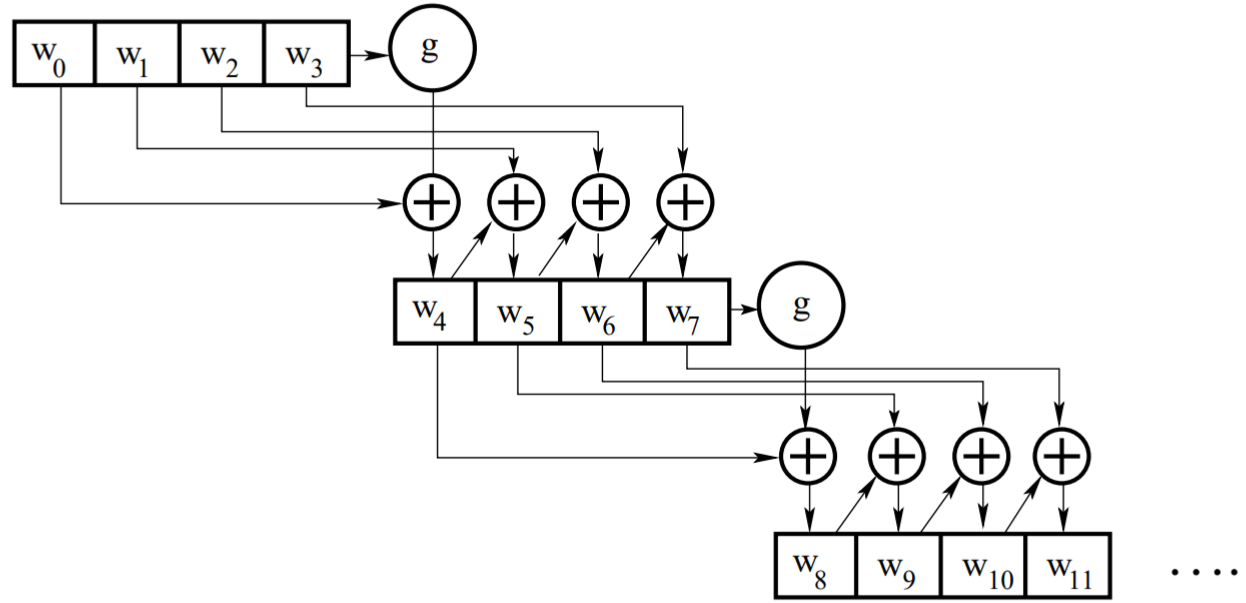
- 

The function g:

1. One-byte circular left shift by a word: $[b_0, b_1, b_2, b_3] \rightarrow [b_1, b_2, b_3, b_0]$
2. Byte substitution using S-box
3. XOR 1 & 2 with a round constant (breaks symmetry)

AES Round Function: Key Expansion

- 



The function g:

1. One-byte circular left shift by a word: $[b_0, b_1, b_2, b_3] \rightarrow [b_1, b_2, b_3, b_0]$
2. Byte substitution using S-box
3. XOR 1 & 2 with a round constant (breaks symmetry)

Block cipher competition

.关于公布全国密码算法设计竞赛 第一轮算法评选结果的通知

时间：2019年09月27日 来源：中国密码学会

分享：

根据全国密码算法设计竞赛工作安排，经公开评议、检测评估和专家评选，全国密码算法设计竞赛第一轮算法评选结果已经揭晓，现公布《全国密码算法设计竞赛分组算法第二轮入选名单》（见附件1）和《全国密码算法设计竞赛公钥算法第二轮入选名单》（含公钥加密算法、数字签名算法、密钥交换算法，见附件2）。

本次公布的密码算法可在2019年10月20日前完成非框架性修改。修改完善并按要求提交后，将在学会网站统一发布。欢迎密码科技工作者、密码研究爱好者积极参与评议。

附件1：全国密码算法设计竞赛分组算法第二轮入选名单.docx

附件2：全国密码算法设计竞赛公钥加密算法第二轮入选名单.docx

Block cipher competition

全国密码算法设计竞赛分组算法第二轮入选名单

排名	算法名称	第一设计者	参与设计者
1	uBlock	吴文玲（中国科学院软件研究所）	张　蕾（中国科学院软件研究所） 郑雅菲（中国科学院软件研究所） 李灵琛（中国科学院软件研究所）
2	Ballet	崔婷婷（杭州电子科技大学)	王美琴（山东大学） 樊燕红（山东大学） 胡　凯（山东大学） 付　勇（山东大学） 黄鲁宁（山东大学）
3	FESH	贾珂婷（清华大学）	董晓阳（清华大学） 魏淙洺（清华大学） 李　铮（山东大学） 周海波（山东大学） 丛天硕（清华大学）

Next Lecture

- Hash, RO model, Finite field ...