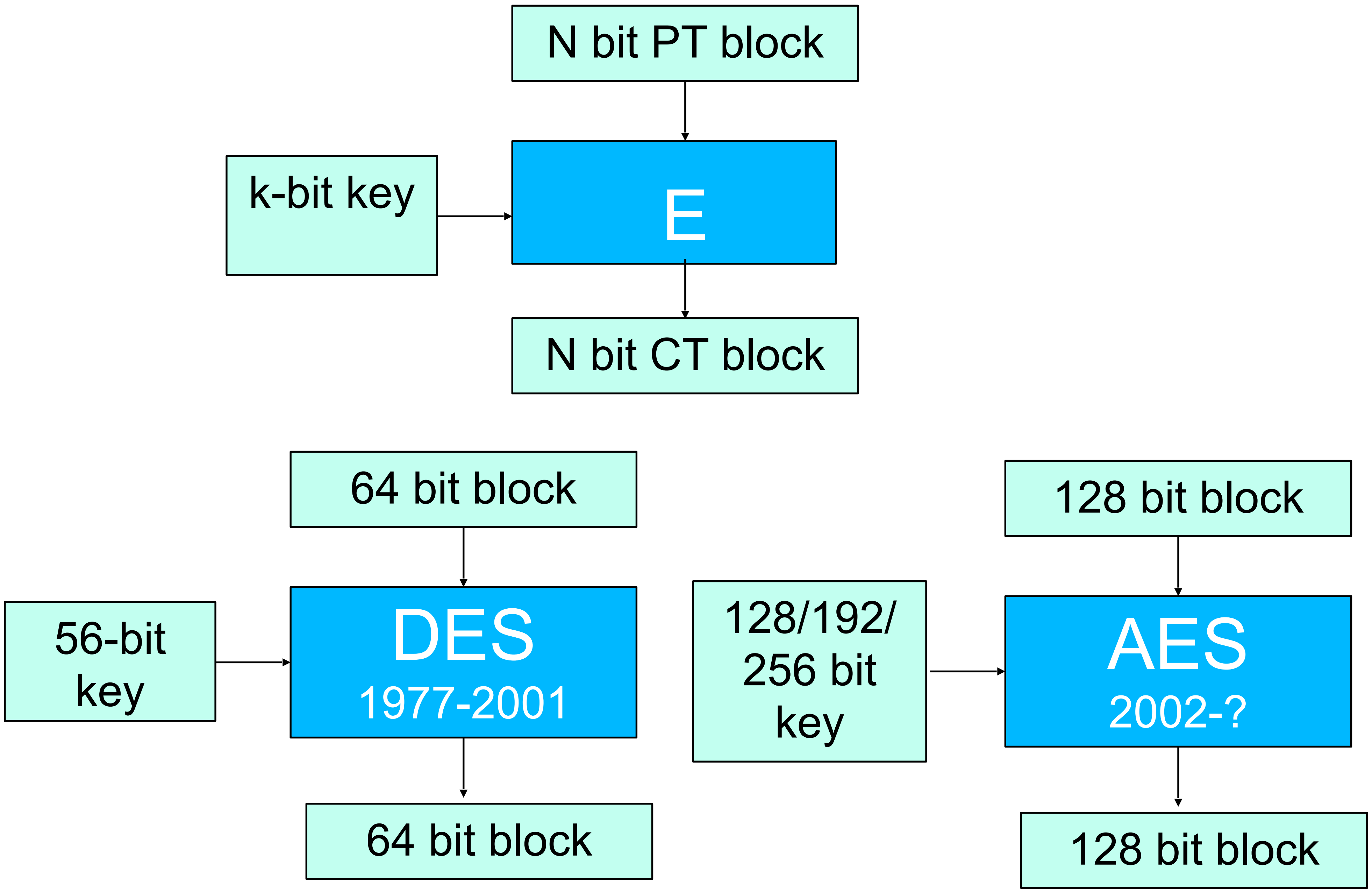
Symmetric-Key Encryption
Block Ciphers

Sushmita Ruj

Recap

- OTP and perfect secrecy of OTP
- Construction of Stream ciphers using PRG
- Statistical Tests
- Attacks when OTP is used more than once
- RC4 Stream Cipher
- Linear Feedback Shift Registers
- Stream ciphers with non-repeating nonce

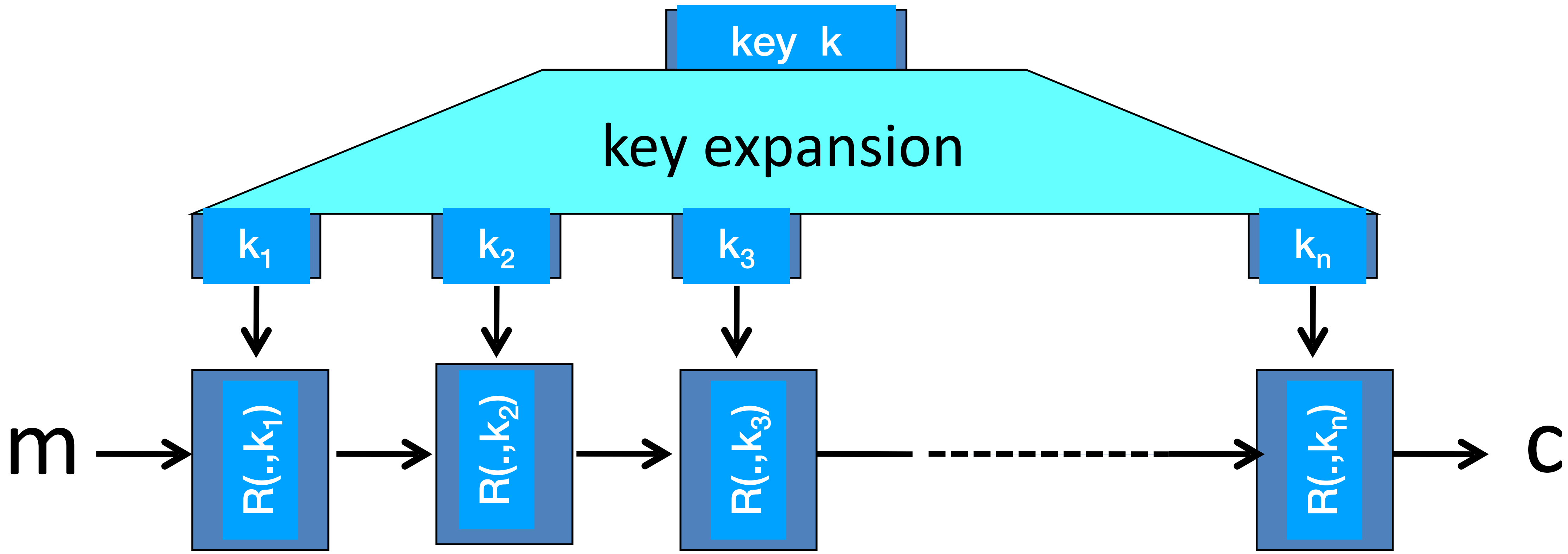
What is a Block Cipher?



DES broken by exhaustive search on keys

3DES: n= 64 bits, k = 168 bits

Block Ciphers Built by Iteration



R(m,k) is called a round function

for 3DES (n=48), for AES-128 (n=10)

Performance

AMD Opteron, 2.2 GHz (Linux)

Cipher	Block/key size	Speed (MB/s)
RC4		126
Salsa20/12		643
Sosemanuk		727
3DES	64/168	13
AES-128	128/128	109

PRPs and PRFs

- Pseudo Random Function **(PRF)** defined over (K,X,Y):

$$F: \quad X \times K \rightarrow Y$$

  such that exists "efficient" algorithm to evaluate F(k,x)

- Pseudo Random Permutation **(PRP)** defined over (K,X):

$$E: \quad X \times K \rightarrow X$$

  such that:

  1. Exists "efficient" <u>deterministic</u> algorithm to evaluate E(x,k)
  2. The function E(., K) is one-to-one
  3. Exists "efficient" inversion algorithm D(y,k)

- Functionally, any PRP is also a PRF.
  - A PRP is a PRF where X=Y and is efficiently invertible.

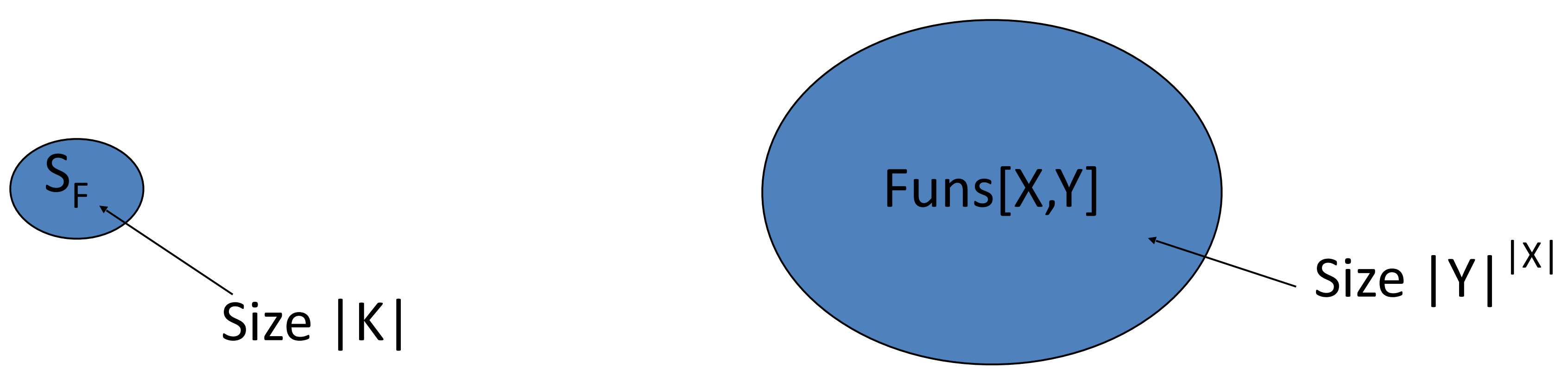
Secure PRFs

- Let $F: X \times K \rightarrow Y$ be a PRF

  Funs[X,Y]: the set of **<u>all</u>** functions from X to Y

  $S_F = \{ F(\cdot, k) \text{ s.t. } k \in K \} \subseteq \text{Funs}[X,Y]$

- <u>Intuition</u>: a PRF is **secure** if a random function in Funs[X,Y] is indistinguishable from a random function in S_F



Secure PRFs have been used in AES and 3DES

The Data Encryption Standard (DES)

- Early 1970s: Horst Feistel designs Lucifer at IBM

  key-len = 128 bits ; block-len = 128 bits

- 1973: NBS asks for block cipher proposals.

  IBM submits variant of Lucifer.

- 1976: NBS adopts DES as a federal standard

  key-len = 56 bits ; block-len = 64 bits

- 1997: DES broken by exhaustive search
- 2000: NIST adopts Rijndael as AES to replace DES

Widely deployed in banking (ACH) and commerce

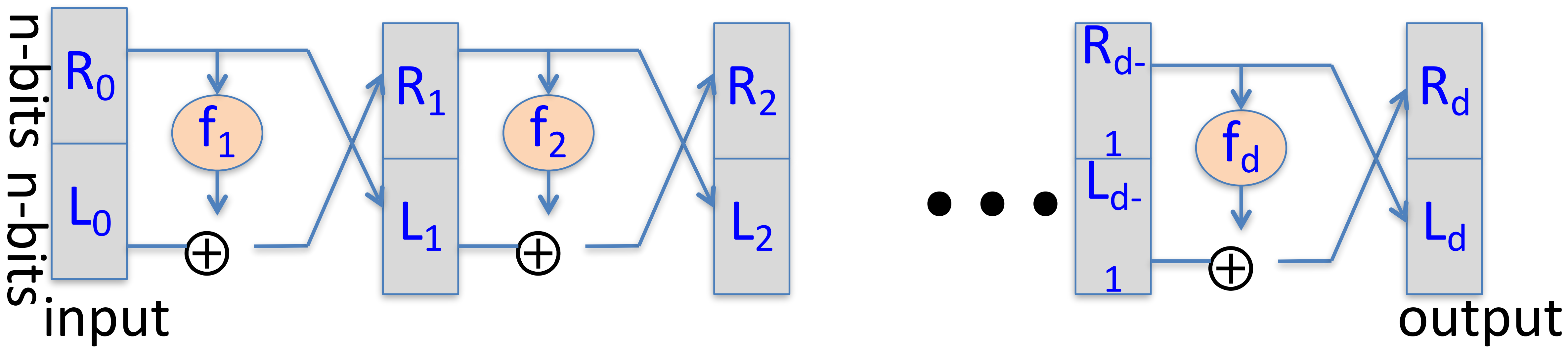
Block Ciphers from PRP

- Focus: Block cipher constructions from a PRF $f_k(\bullet)$
  - PRFs seem easier to design (less restrictions)
- Before: ‘plain’ PRP $E_k(\bullet)$ (not a block cipher)
- Now: construct block cipher (invertible PRP) $E_k,\ D_k$
- Challenge: making it invertible…
- Solution: The Feistel Construction

DES: core idea – Feistel Network

Given functions $f_1, \ldots, f_d: \{0,1\}^n \longrightarrow \{0,1\}^n$

Goal: build invertible function $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$



$$R_i = f_i(R_{i-1}) \oplus L_{i-1}$$

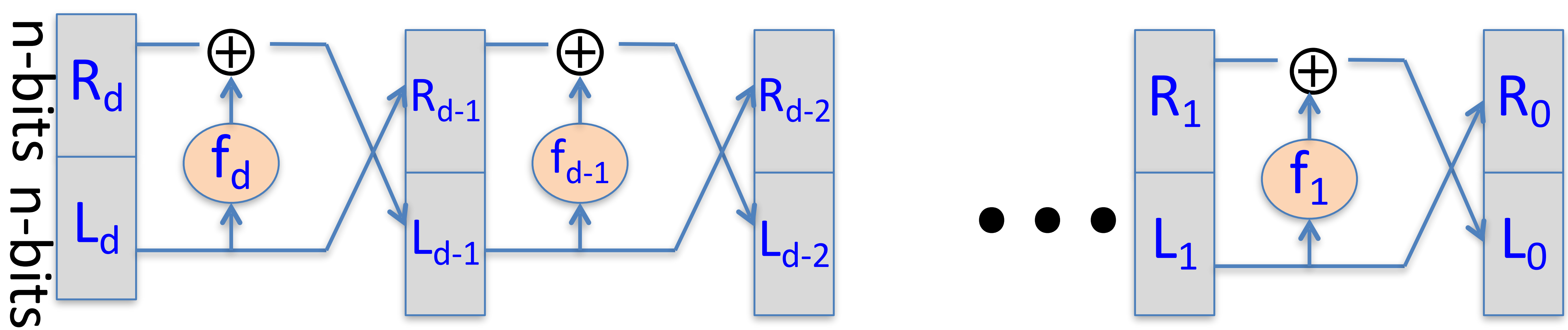
$$L_i = R_{i-1}$$

Fiestel Network is Invertible

Claim: for all $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$

Feistel network $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is invertible

Proof: construct inverse:



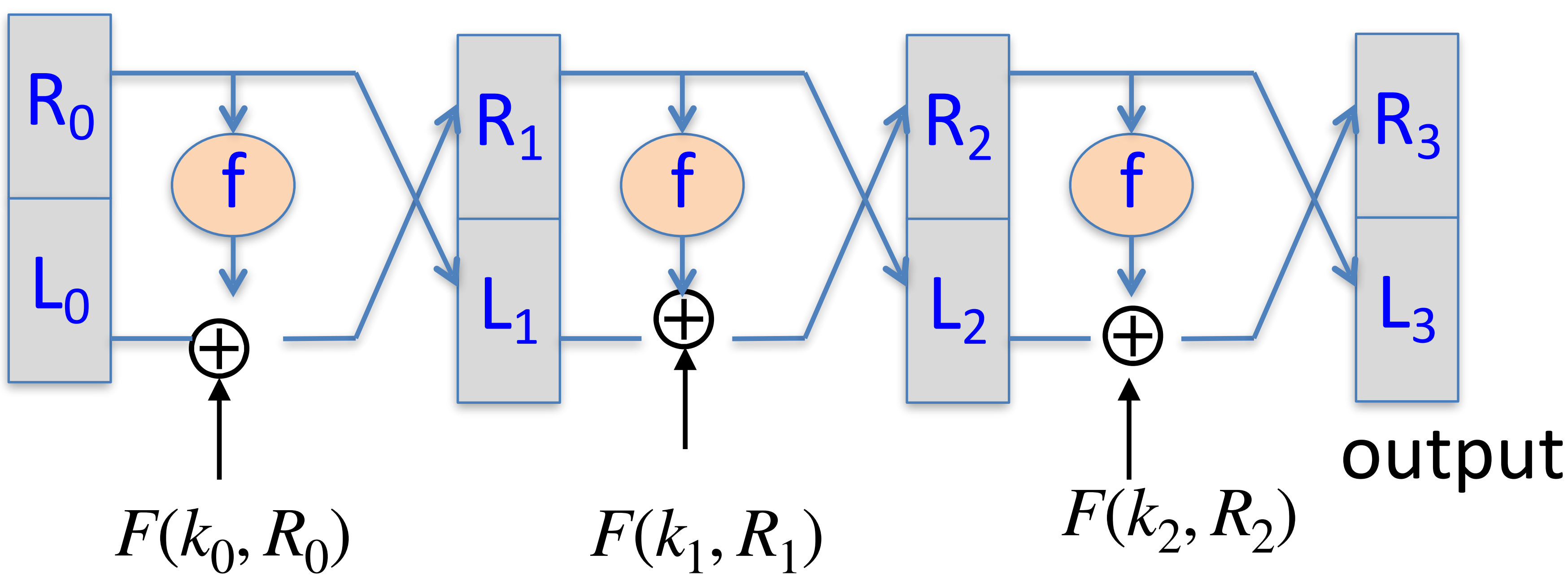
$L_{i-1} = f_i(L_i) \oplus R_i \qquad R_{i-1} = L_i$

3-Round Fiestal

“Thm:” (Luby-Rackoff ‘85):

f: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a secure PRF

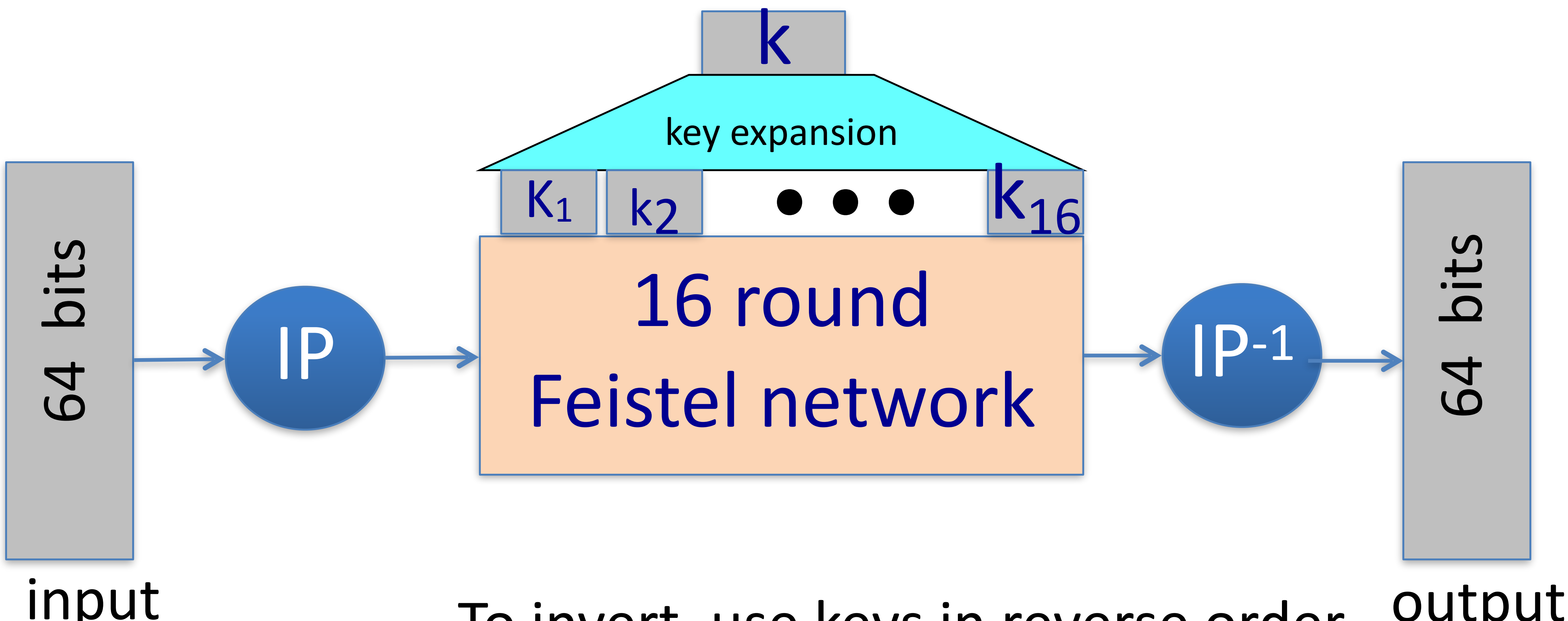
$\Rightarrow$ 3-round Feistel F: $K^3 \times \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ a secure PRP



k_0, k_1, k_2 are chosen independently

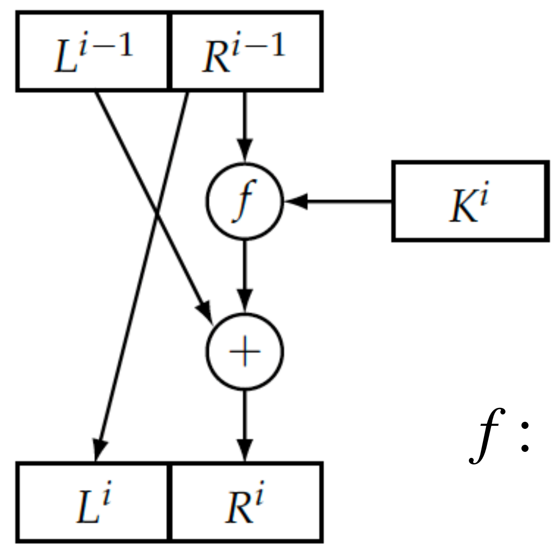
DES: 16 round Feistel network

$f_1, \ldots, f_{16}: \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$, $f_i(x) = \mathbf{F}(k_i, x)$



To invert, use keys in reverse order

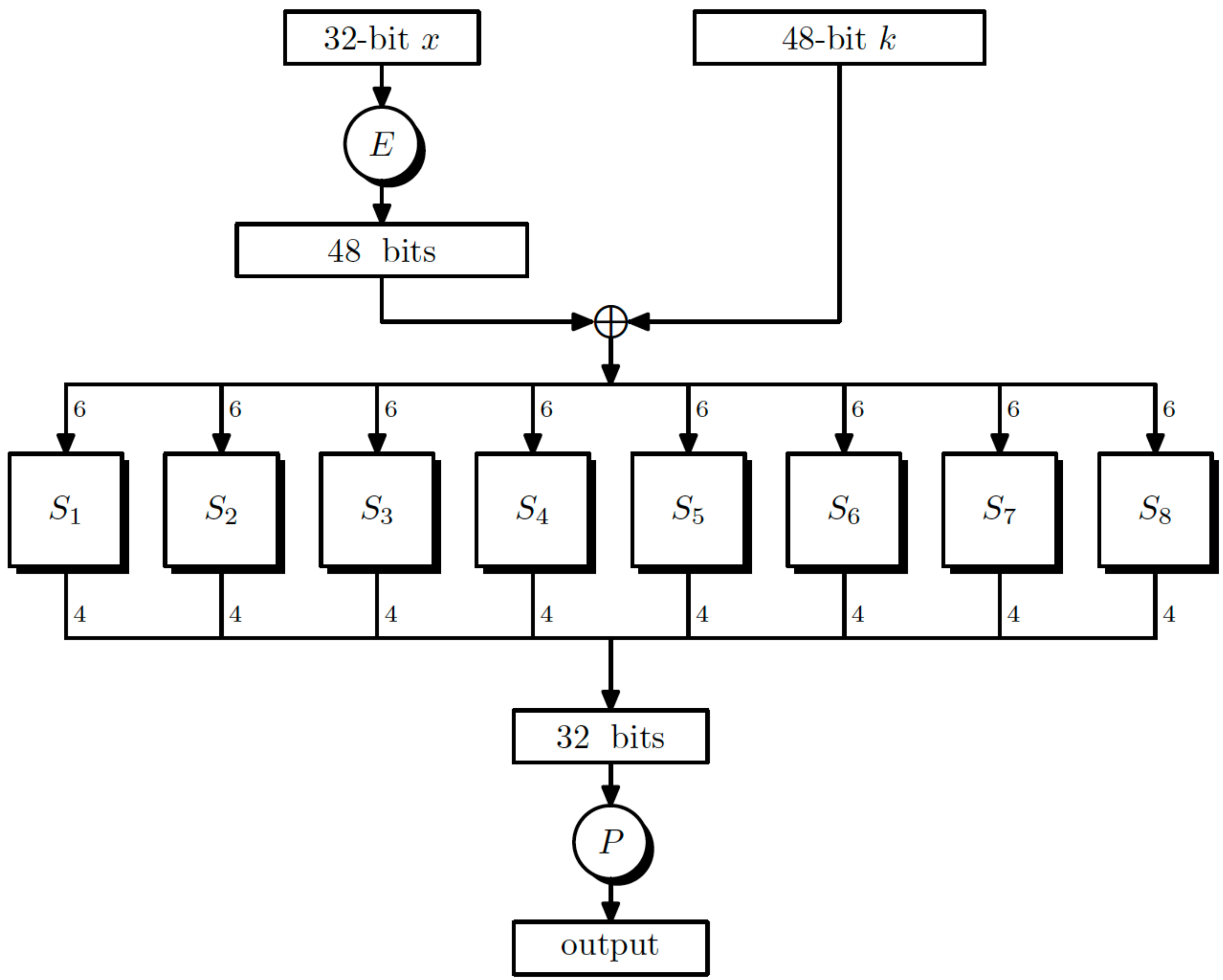
DES: 1 round



$f:\{0,1\}^{32}\times\{0,1\}^{48}\rightarrow\{0,1\}^{32}$

DES has 16 rounds
f takes 32-bit string $R^{j-1}$and a round key K^j.
The key schedule, $(K^1, K^2, \cdots, K^{16})$, consists of 48-bit round keys that are derived from the 56-bit key, K. Each K_i is a certain permuted selection of bits from K.

DES round function f



S-boxes

$S_i: \{0,1\}^6 \longrightarrow \{0,1\}^4$

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

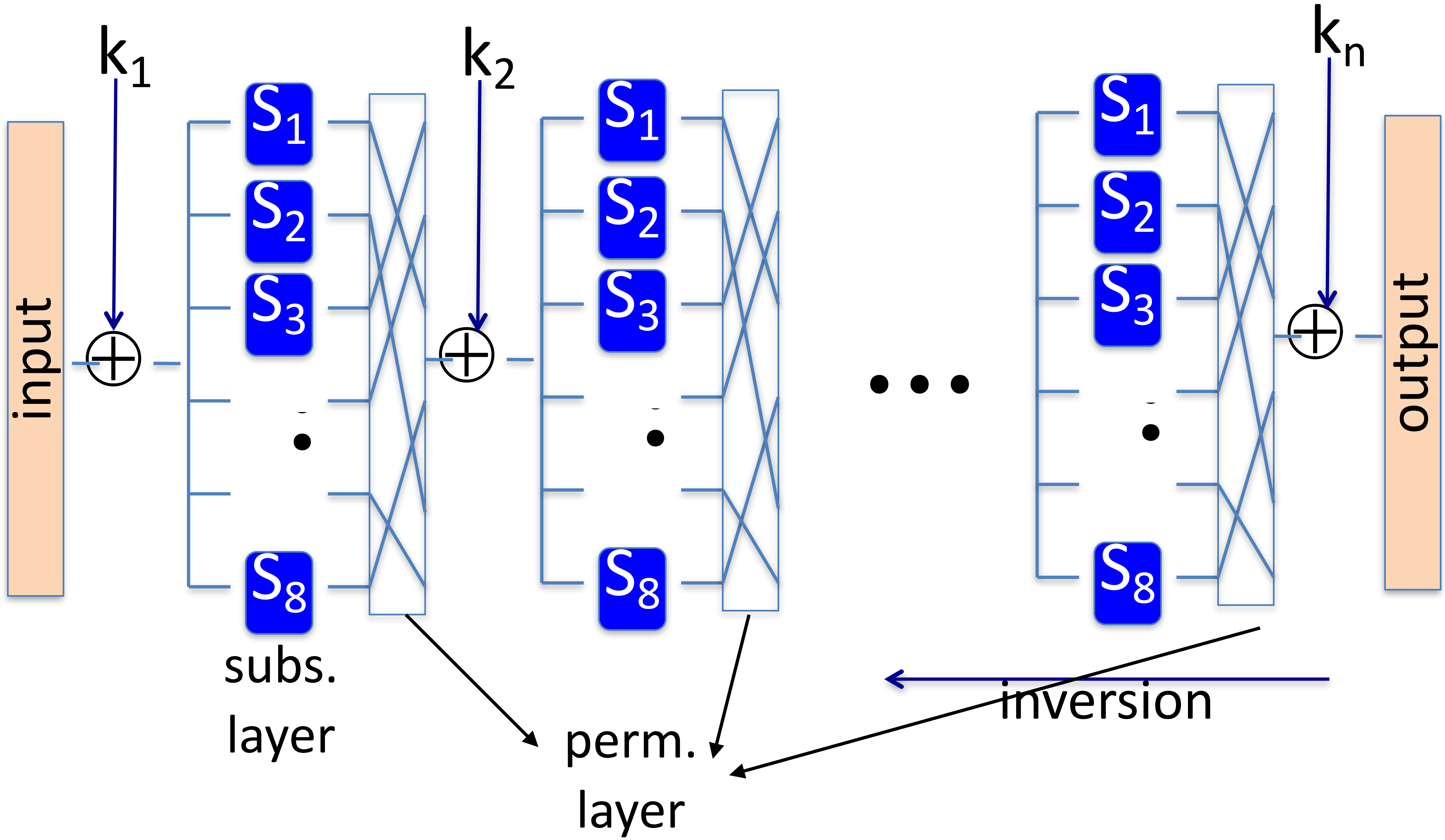
1. The size of the look-up tables, mapping 6-bits to 4-bits, was the largest that could be accommodated on a single chip using 1974 technology.
2. No output bit of an S-box should be close to a linear function of the input bits. if we select any output bit and any subset of the 6 input bits, then the fraction of inputs for which this output bit equals the XOR of these input bits should be close to 1/2.
3. If we fix the leftmost and rightmost bits of the input to an S-box then the resulting 4-bit to 4-bit function is one-to-one. In particular, this implies that each S-box is a 4-to-1 map.
4. Changing one bit of the input to an S-box changes at least two bits of the output.

AES Selection

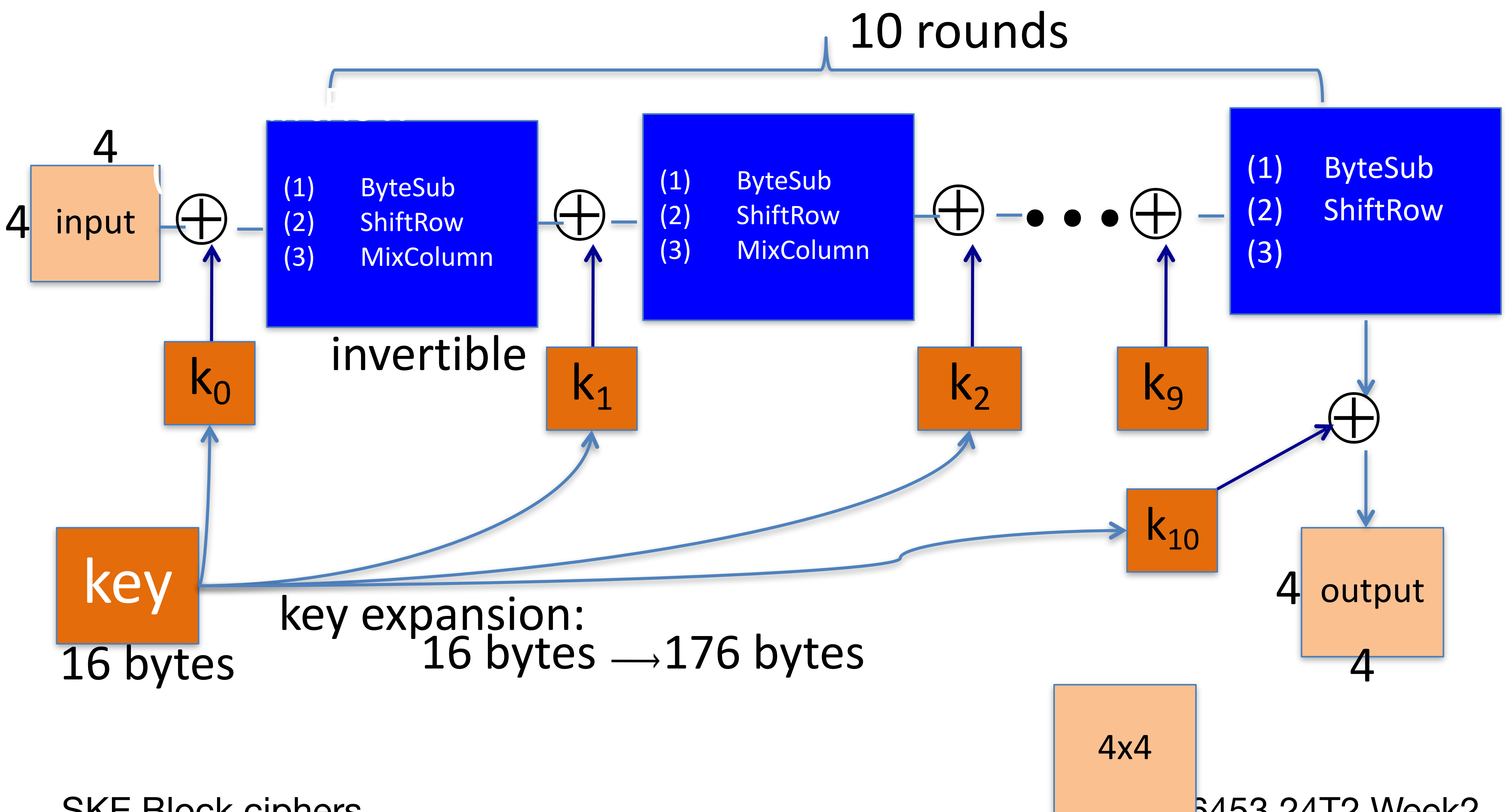
- 1997: NIST publishes request for proposal
- 1998: 15 submissions. Five claimed attacks.
- 1999: NIST chooses 5 finalists
- 2000: NIST chooses Rijndael as AES (designed in Belgium)

Key sizes: 128, 192, 256 bits. Block size: 128 bits

Sustitution-Permutation Network (SPN) in AES

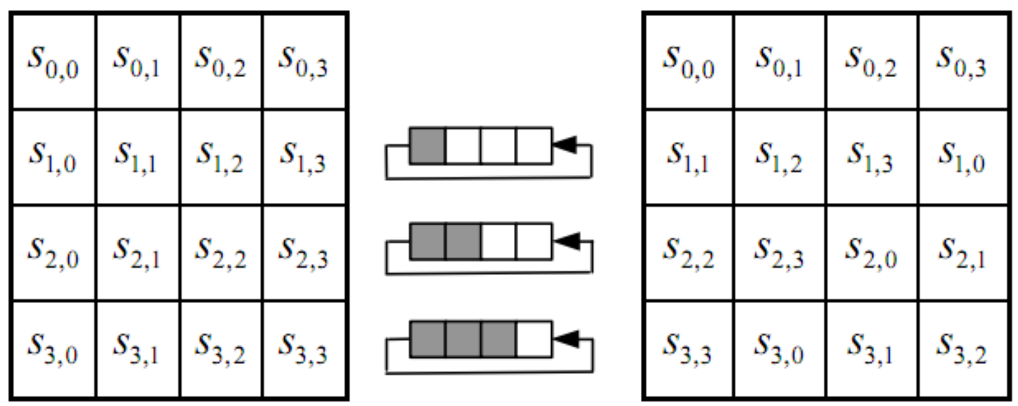


AES-128 schematic

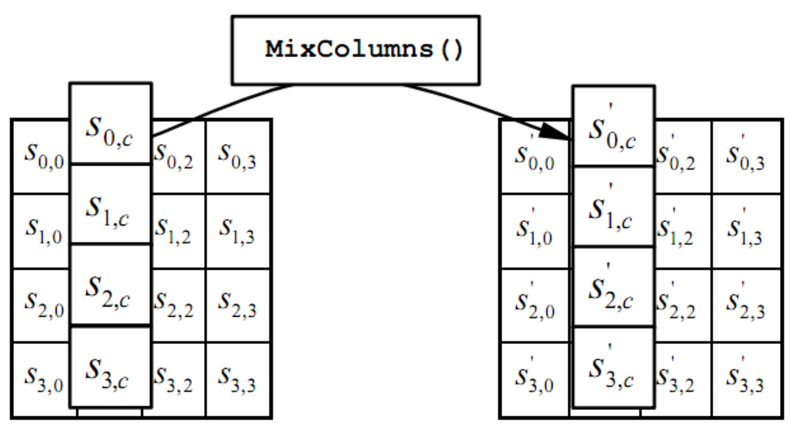


The Round Function

- **ByteSub**: 16 cells, 1 byte each, a 1 byte S-box. AES S-Box consists of 256 entries (hardcoded, with $S(x) \neq \bar{x}$
- **ShiftRows**:



- **MixColumns**:



x8 + x4 + x3 + x + 1.

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_5 & a_6 & a_7 & a_4 \\ a_{10} & a_{11} & a_8 & a_9 \\ a_{15} & a_{12} & a_{13} & a_{14} \end{pmatrix} \Longrightarrow \begin{pmatrix} a'_0 & a'_1 & a'_2 & a'_3 \\ a'_4 & a'_5 & a'_6 & a'_7 \\ a'_8 & a'_9 & a'_{10} & a'_{11} \\ a'_{12} & a'_{13} & a'_{14} & a'_{15} \end{pmatrix}$$

Performance of AES

	Code size	Performance
Pre-compute round functions (24KB or 4KB)	largest	fastest: table lookups and xors
Pre-compute S-box only (256 bytes)	smaller	slower
No pre-computation	smallest	slowest

AES can be implemented in Hardware in both resource constrained and normal devices.
Read Boneh-Shoup Section 4.2.4.1

Attacks on AES

Key recovery attacks :

- Adversary who is given multiple plaintext/ciphertext pairs is able to recover the secret key from these pairs, as in an exhaustive search attack.
- The best-known key recovery attack on AES-128 takes $2^{126.1}$ evaluations of AES.
- This is about four times faster than exhaustive search.
- No Danger to AES-128

Related key attack on AES-256:

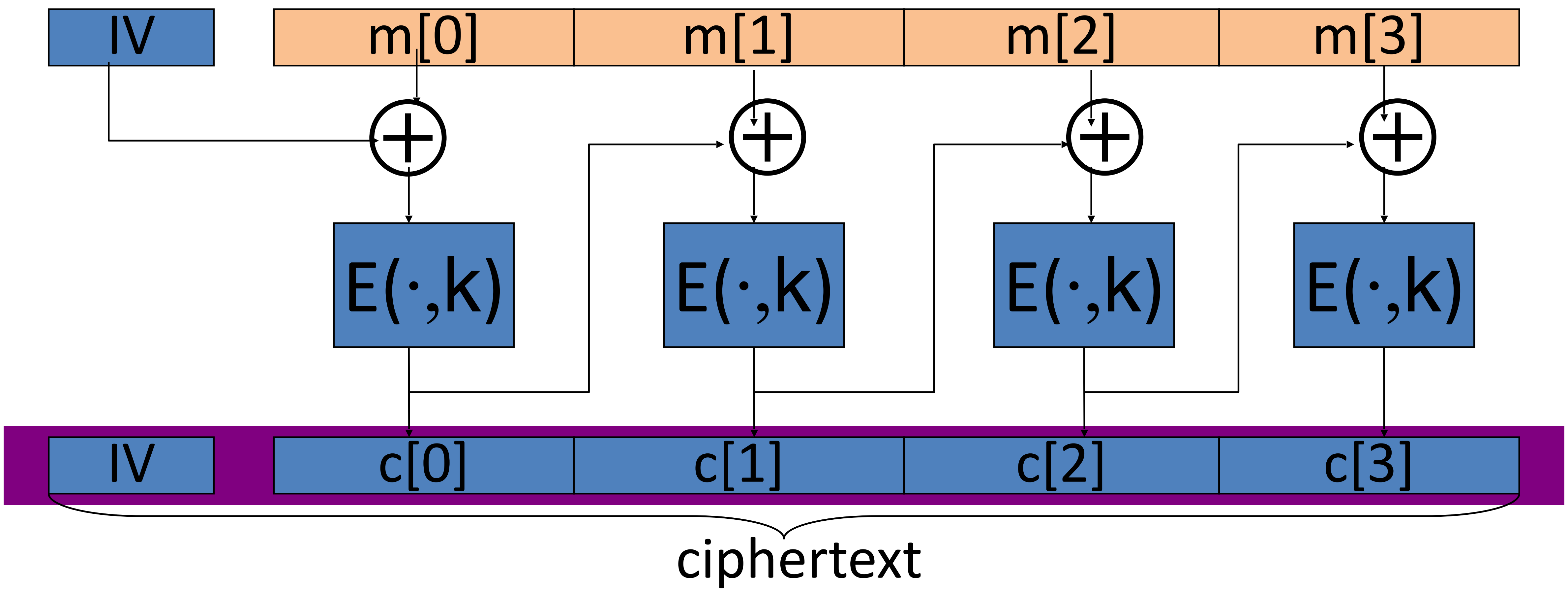
Given 2^{99} inp/out pairs from **four related keys (set to specific values)** in AES-256 can recover keys in time $\approx 2^{99}$

Modes of Operation
How to encrypt many blocks of messages

Cipher Block Chaining (CBC)
Mode CBC with random IV

Let (E,D) be a PRP. E_{CBC}(m,k): choose **random** IV∈X and do:

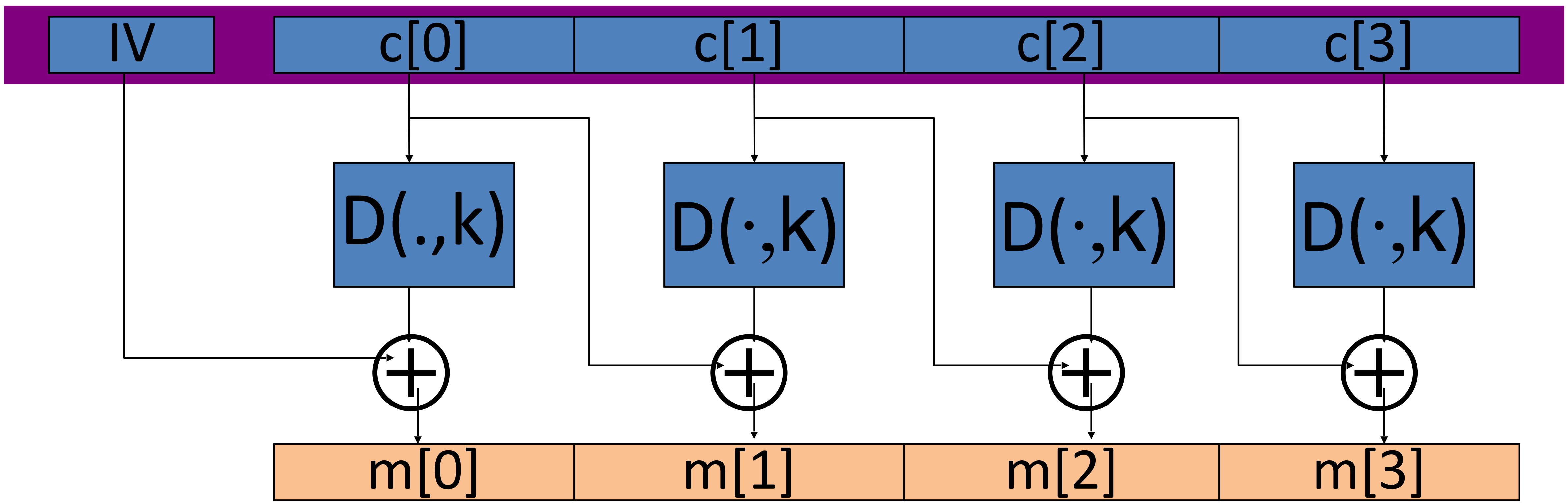
$E : \{0,1\}^n \times \mathscr{K} \to \{0,1\}^n$ $IV \in \{0,1\}^n$



If message mi changes then all subsequent cipher texts have to be recomputed.
So, CBC is used for authentication.

Decryption circuit

IV is not secret and should NOT be used more than once

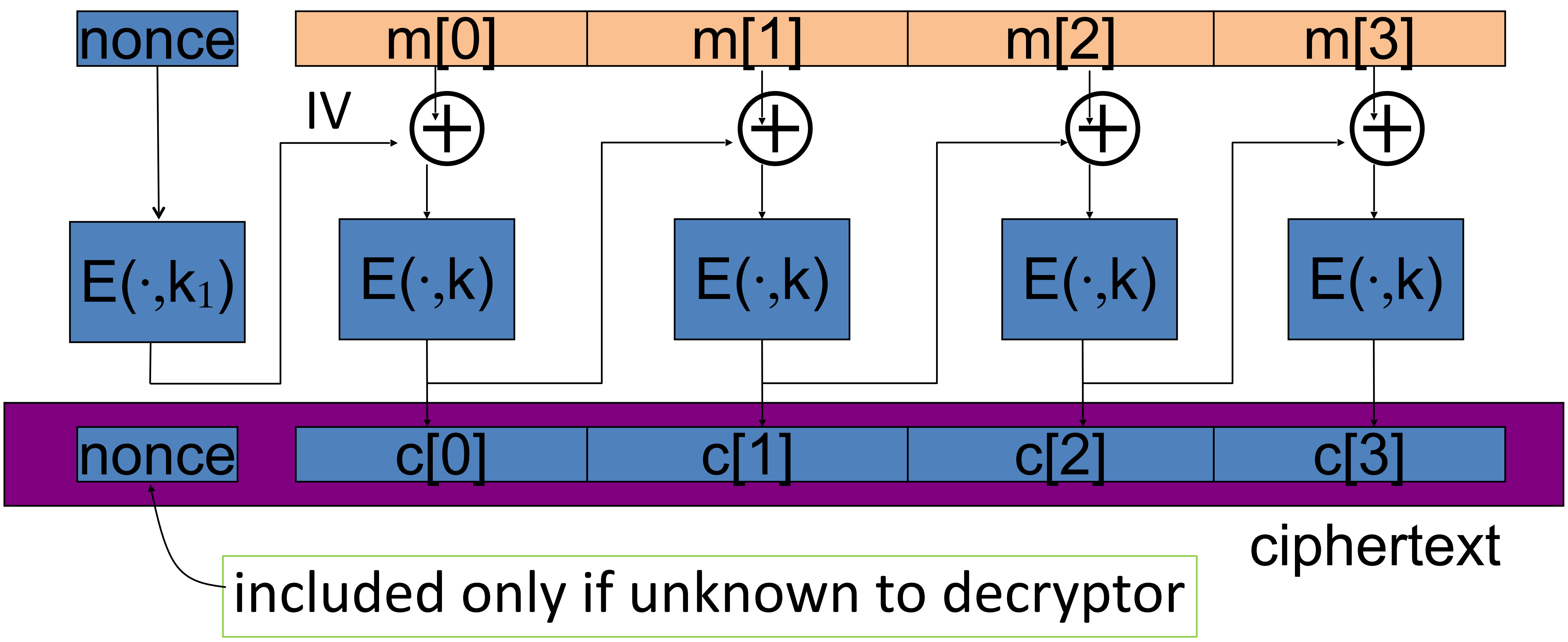


CBC where attacker can <u>predict</u> the IV is not secure.

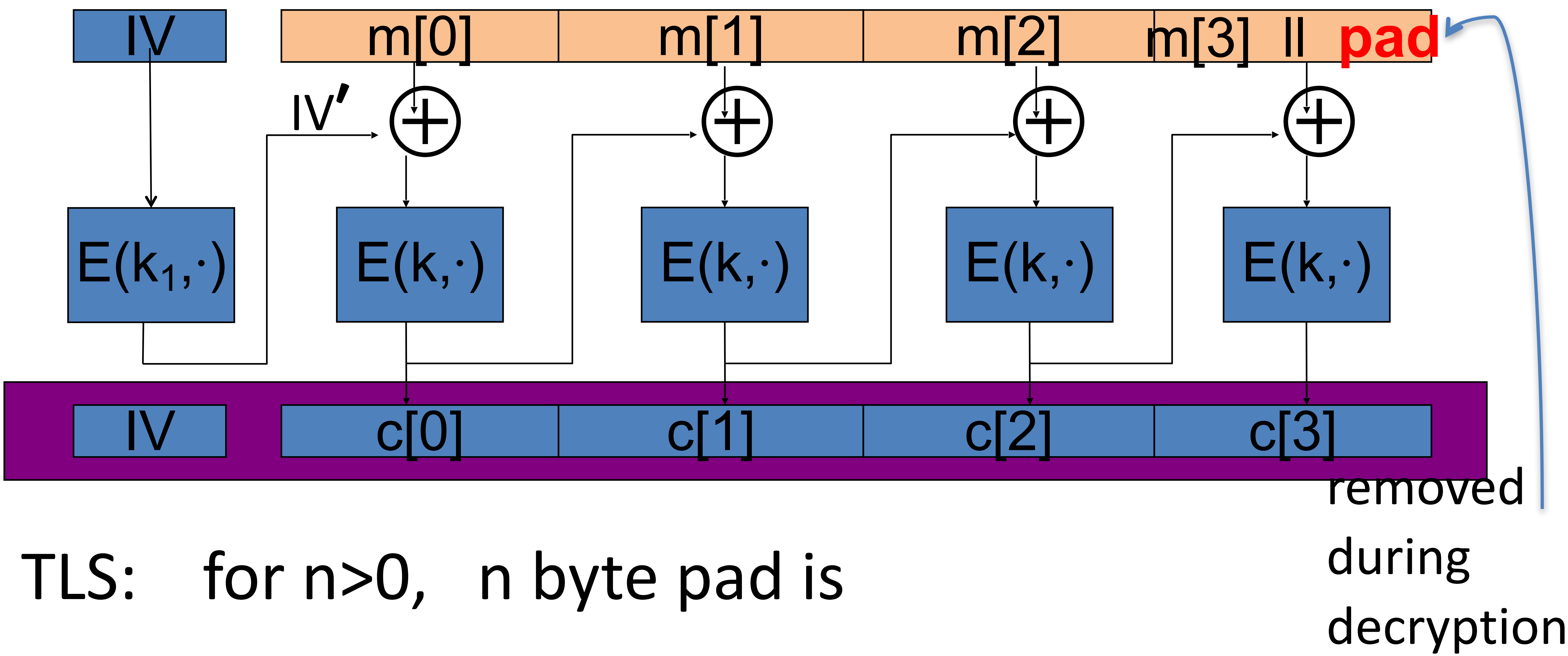
Nonce-based CBC

- Cipher block chaining with unique nonce: key = (k,k_1)

unique nonce means: (key, n) pair is used for only one message



Padding in CBC Mode



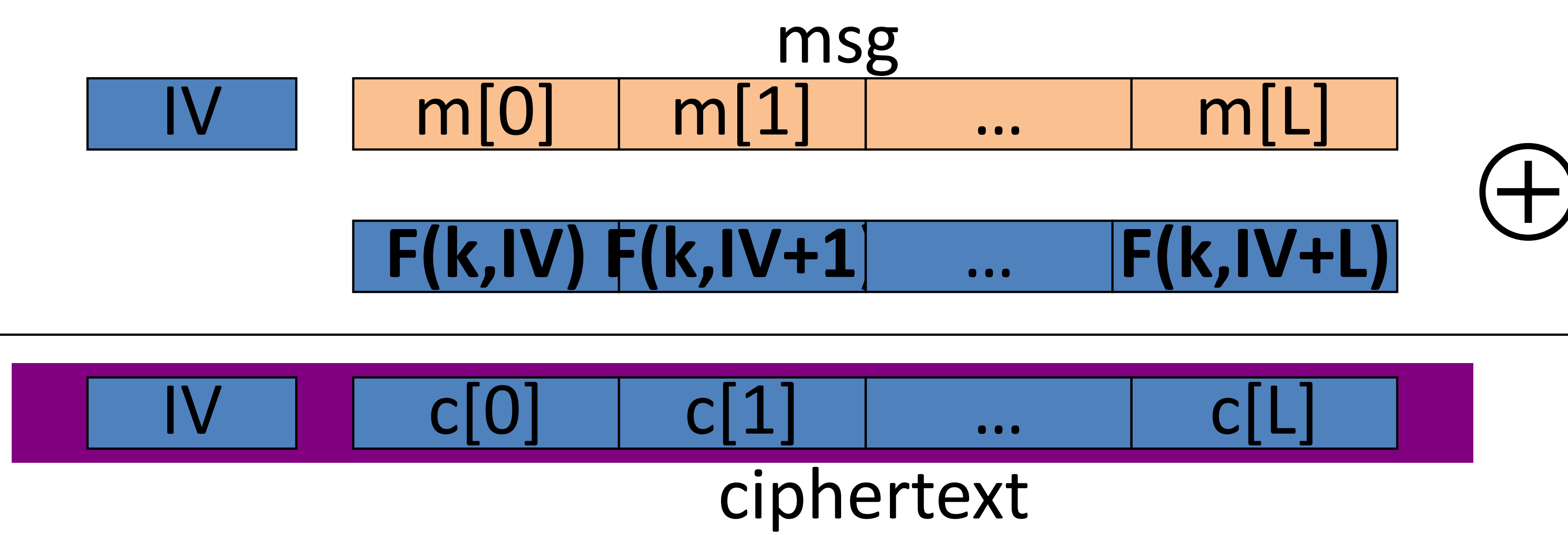
TLS: for n>0, n byte pad is

if no pad needed, add a dummy block

Random Counter-mode

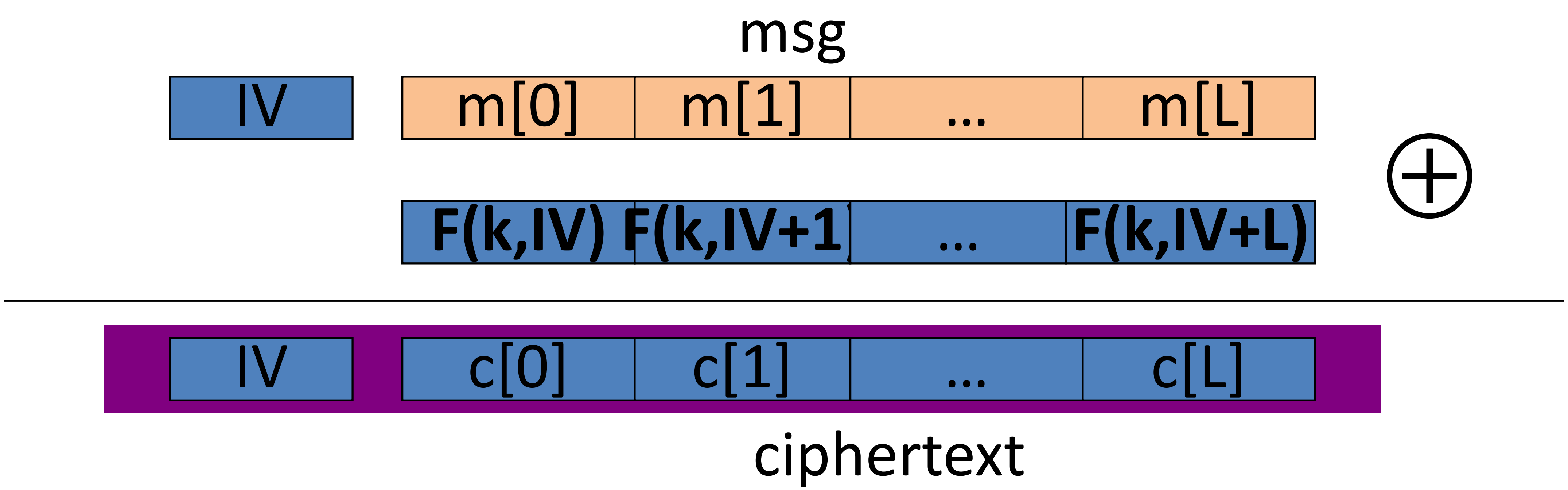
Let F: K × $\{0,1\}^n \longrightarrow \{0,1\}^n$ be a secure PRF.

E(k,m): choose a random $IV \in \{0,1\}^n$ and do:

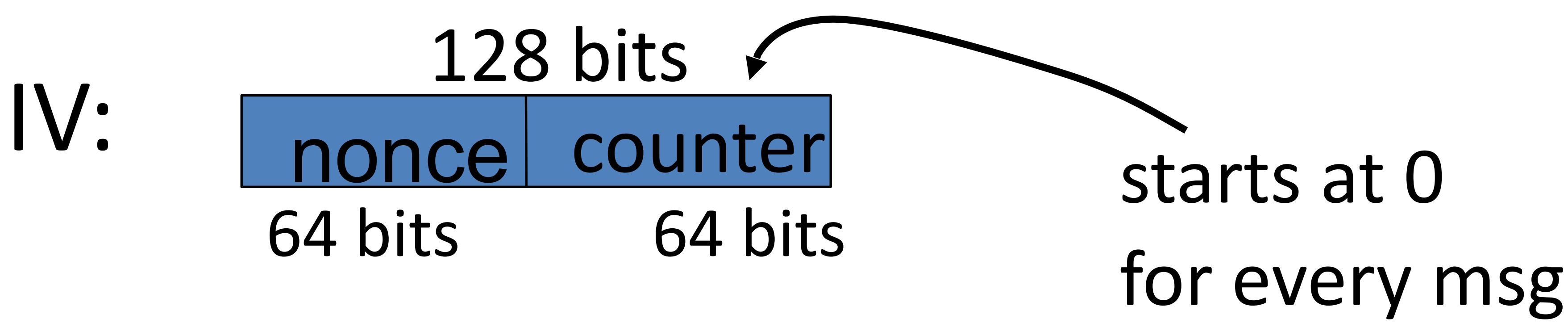


note: parallelizable (unlike CBC)

Nonce ctr-mode



To ensure F(k,x) is never used more than once, choose IV as:



Comparison: ctr vs. CBC

	CBC	ctr mode
uses	PRP	PRF
parallel processing	No	Yes
Security of rand. enc.	$q^2 L^2 \ll \lvert X \rvert$	$q^2 L \ll \lvert X \rvert$
dummy padding block	Yes	No
1 byte msgs (nonce-based)	16x expansion	no expansion

Thank you!