CS915/435 Advanced Computer Security - Elementary Cryptography

Block Cipher II

Roadmap

- Symmetric cryptography
  - Classical cryptographic
  - Stream cipher
  - **Block cipher** I, **II**
  - Hash
  - MAC
- Asymmetric cryptography
  - Key agreement
  - Public key encryption
  - Digital signature

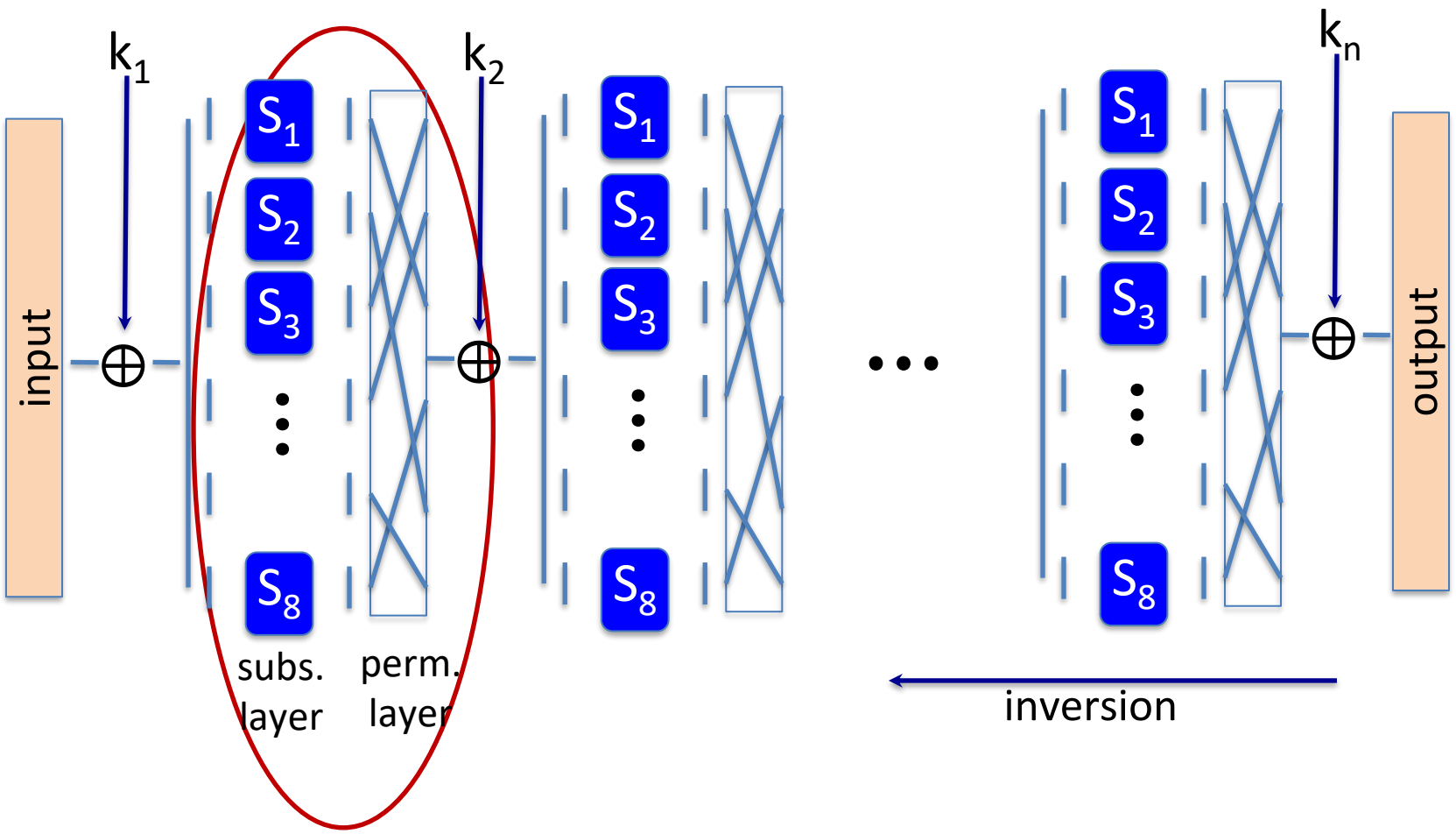
Advanced Encryption Standard

- 1997: NIST publishes request for proposal
  - Requires a block length of 128 bits and support key lengths of 128, 192 and 256
- 1998: 15 Submissions
- 1999: NIST chooses 5 finalists
  - MARS, RC6, Rijndael, Serpent and Twofish
- 2000: NIST chooses Rijndael as AES
  - Designed by Rijmen and Daemen

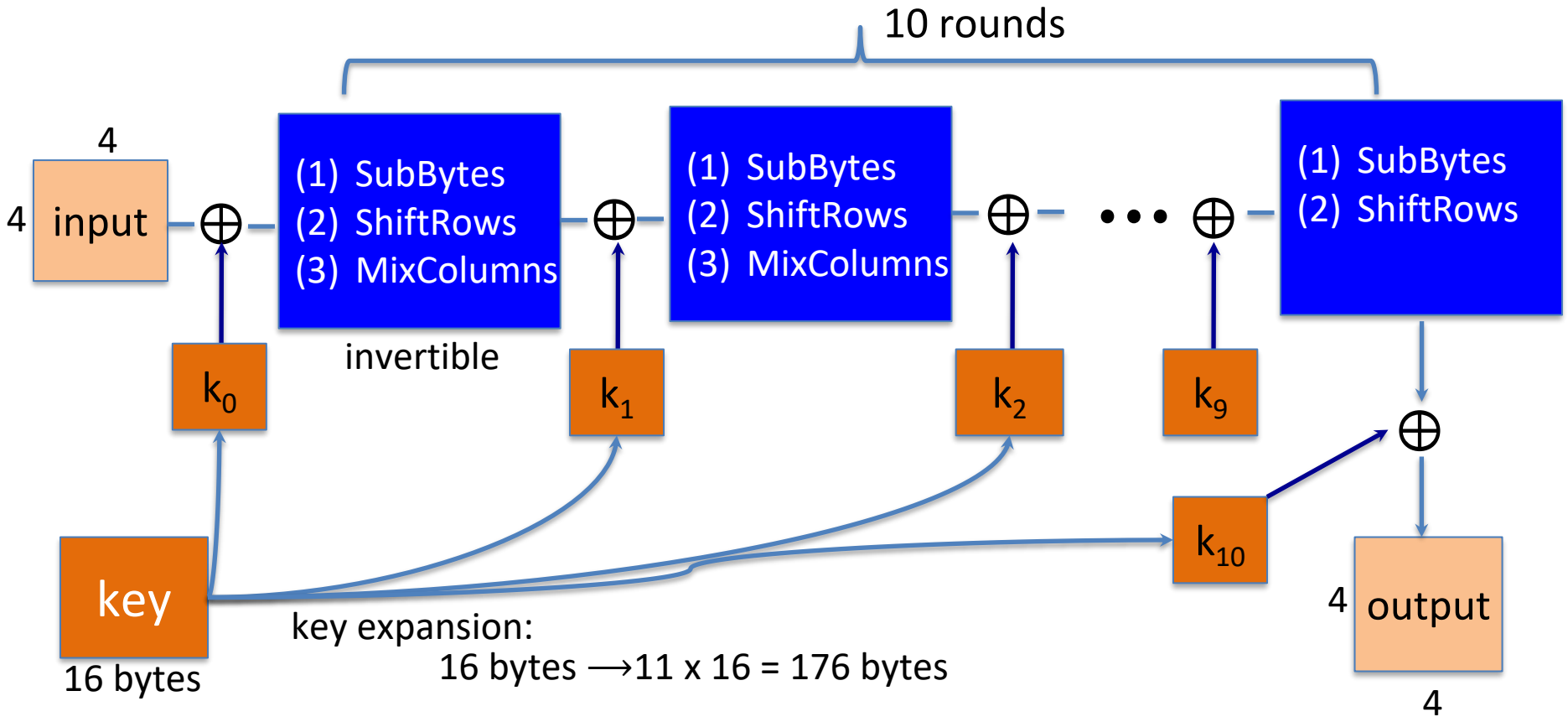
Overview of five finalists

Name	Author	Type
Mars	IBM	Extended Feistel Network
RC6	RSA	Feistel Network
Rijndael	Joan Daemen and Vincent Rjimen	Substitution Permutation Network
Serpent	Ross Anderson, Eli Biham, and Lars Knudsen	Substitution Permutation Network
Twofish	Bruce Schneier, John Kelsey, Niels Ferguson, Doug Whiting, David Wagner and Chris Hall	Feistel Network

AES is a Substitution-Permutation network (not Feistel)

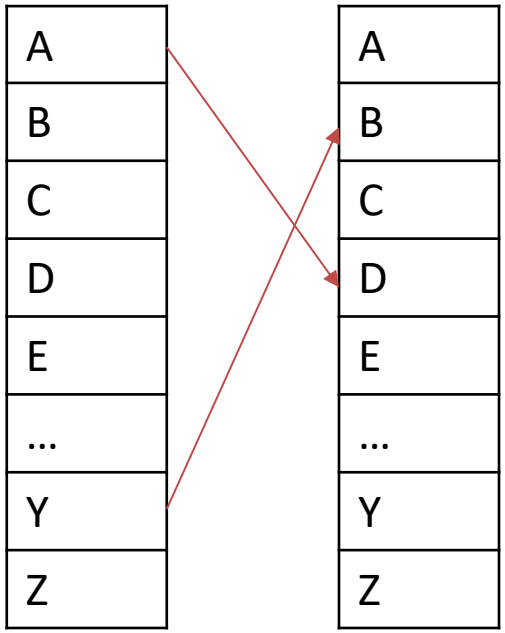


AES-128 schematic



SubBytes

Substitution cipher
(26 letters)



SubBytes in AES
(8 bits input)

00000000
00000001
00000010
00000011
00000100
…
11111110
11111111

00000000
00000001
00000010
00000011
00000100
…
11111110
11111111

Finite Field

- Suppose p is a prime. E.g., $p = 7$
- Z_P is a field. $Z_7 = \{0,1,2,3,4,5,6\}$
- Given two elements
  - $a = 3$
  - b = 5
- We have $a \times b$ mod 7 = 1
- Here b is called a multiplicative inverse of a
- Given a, we can calculate b using Extended Euclidean Algorithm

$3^{-1} * 3 = 1 \bmod 7$

$5 * 3 = 1 \bmod 7$

GCD(20, 12) = ?

au + bv = GCD(a,b)

Another example of finite field

- $GF(2^m)$: works on polynomials instead
- Suppose m = 4
  - $[1\ 0\ 0\ 1] \rightarrow x^3 + 1$
  - $[0\ 1\ 1\ 1] \rightarrow x^2 + x^1 + 1$
- First, need to define an irreducible polynomial
  - $f_1 = x^3 + x^2 + x + 1$
  - $f_2 = x^3 + x + 1$
- f_1 is not irreducible: $f_1 = (x + 1)(x^2+1)$
- f_2 is irreducible (= prime)

AES uses GF(2^8) Irreducible polynomial $x^8+x^4+x^3+x+1$

Multiplication of polynomials

- Assume two elements in $GF(2^m)$
  - $a = [0\ 1\ 0\ 1] \rightarrow x^2 + 1$
  - $b = [0\ 1\ 1\ 1] \rightarrow x^2 + x + 1$
- $a \times b = (x^2 + 1)(x^2 + x + 1) \bmod x^3 + x + 1$

  $= x^4 + x^3 + x + 1 \bmod x^3 + x + 1$

  $= (x + 1)(x^3 + x + 1) + x^2 + x \bmod x^3 + x + 1$

  $= x^2 + x \rightarrow [0\ 1\ 1\ 0]$

Computation of inverses can be done using the same extended Euclidean algorithm

SubBytes

Input: a is an element of $GF(2^8)$

Compute the inverse of a

- Let the inverse be a^{-1}
- $a^{-1} \rightarrow (a_7 a_6 a_5\ a_4 a_3\ a_2\ a_1 a_0)$

For i = 0, 1, … 7

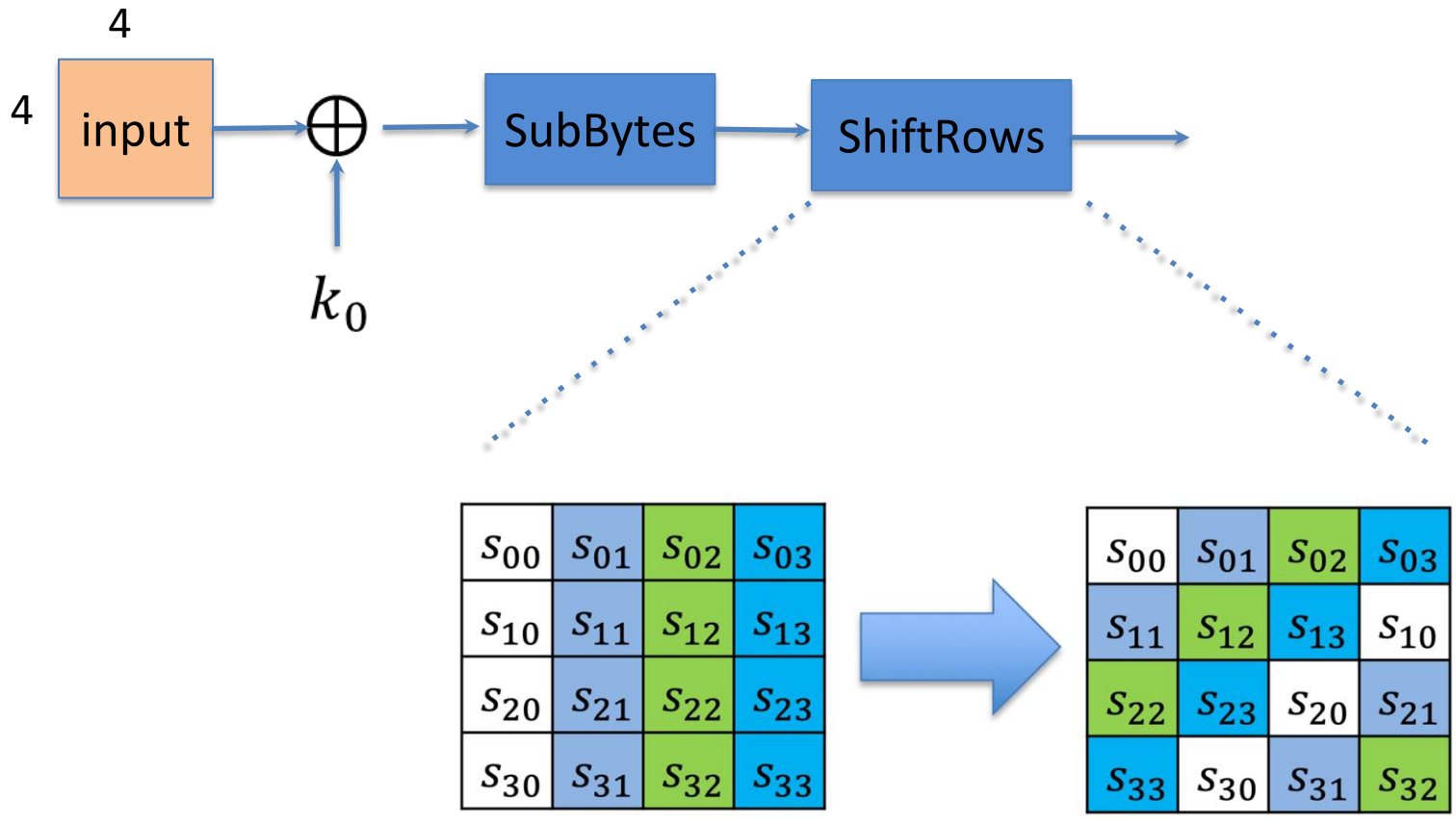
do $b_i = (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i) \bmod 2$

Output $(b_7 b_6 b_5\ b_4 b_3\ b_2\ b_1 b_0)$

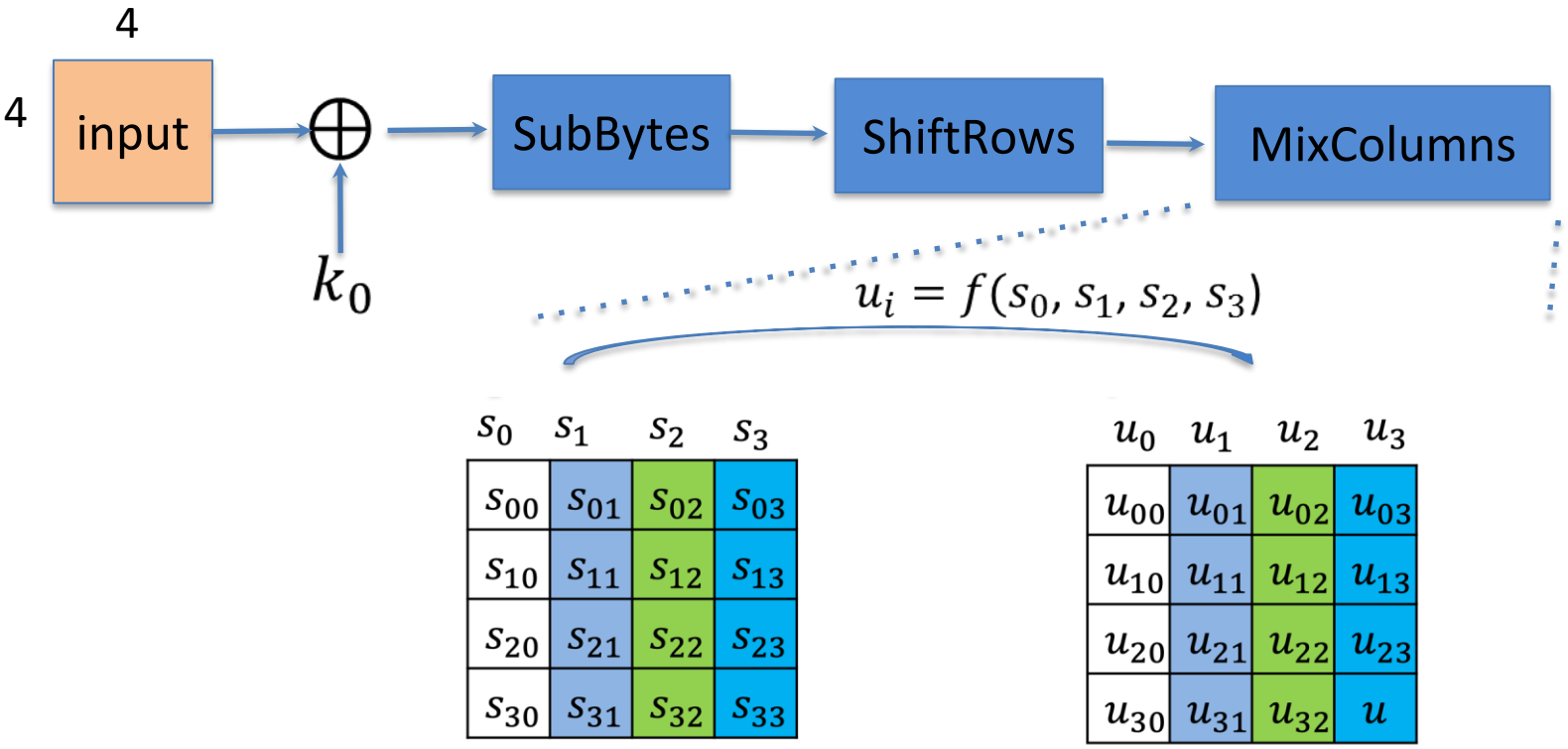
Summary in SubBytes

- Unlike DES, the S-Box is not random
  - It is defined in a finite field
- Unlike DES, the S-Box doesn't have to be hard-coded
  - It allows very compact software implementation (say on smartcards)
- Flexible trade-off between the code size and the performance

ShiftRows


4
4
input
SubBytes
ShiftRows
k_0
s_{00} s_{01} s_{02} s_{03}
s_{10} s_{11} s_{12} s_{13}
s_{20} s_{21} s_{22} s_{23}
s_{30} s_{31} s_{32} s_{33}
s_{00} s_{01} s_{02} s_{03}
s_{11} s_{12} s_{13} s_{10}
s_{22} s_{23} s_{20} s_{21}
s_{33} s_{30} s_{31} s_{32}

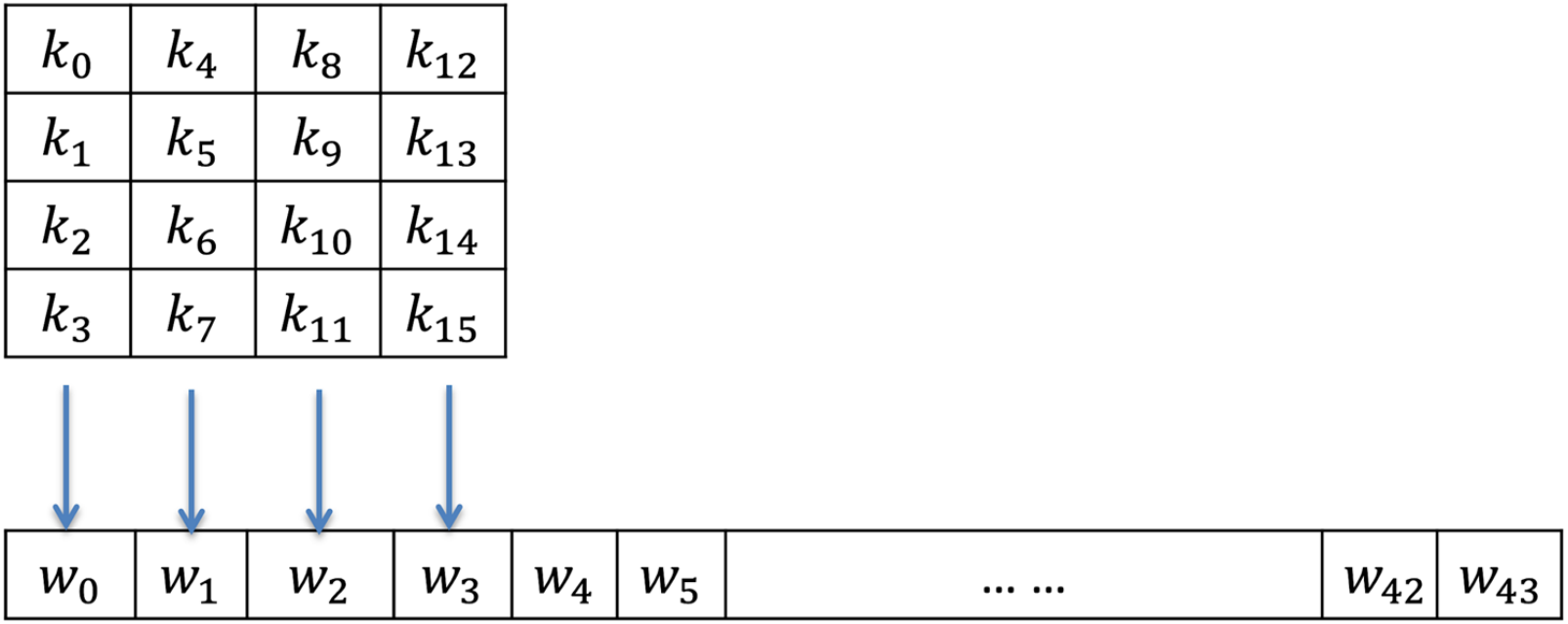
MixColumns



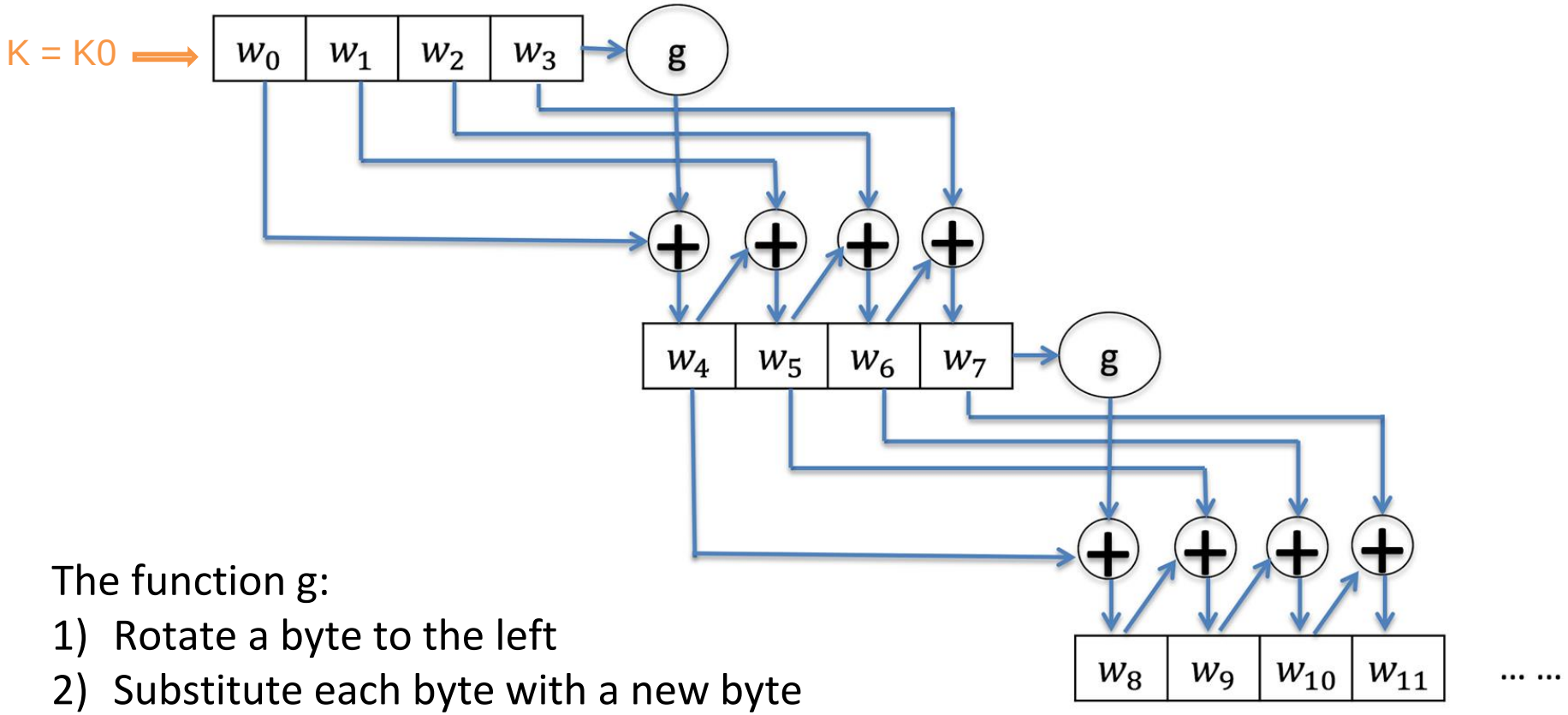
- Each byte in a column is replaced by 2 times that byte, plus three times the next byte, plus the byte that comes next, plus the byte that follows.

Key schedule (AES-128)

- Expanding 16 bytes eleven times to 176 bytes
- AES defines a word as consisting of 4 bytes



Scheduling diagram



The function g:

1) Rotate a byte to the left
2) Substitute each byte with a new byte based on s-box
3) XOR the result with some constant

Variants of AES

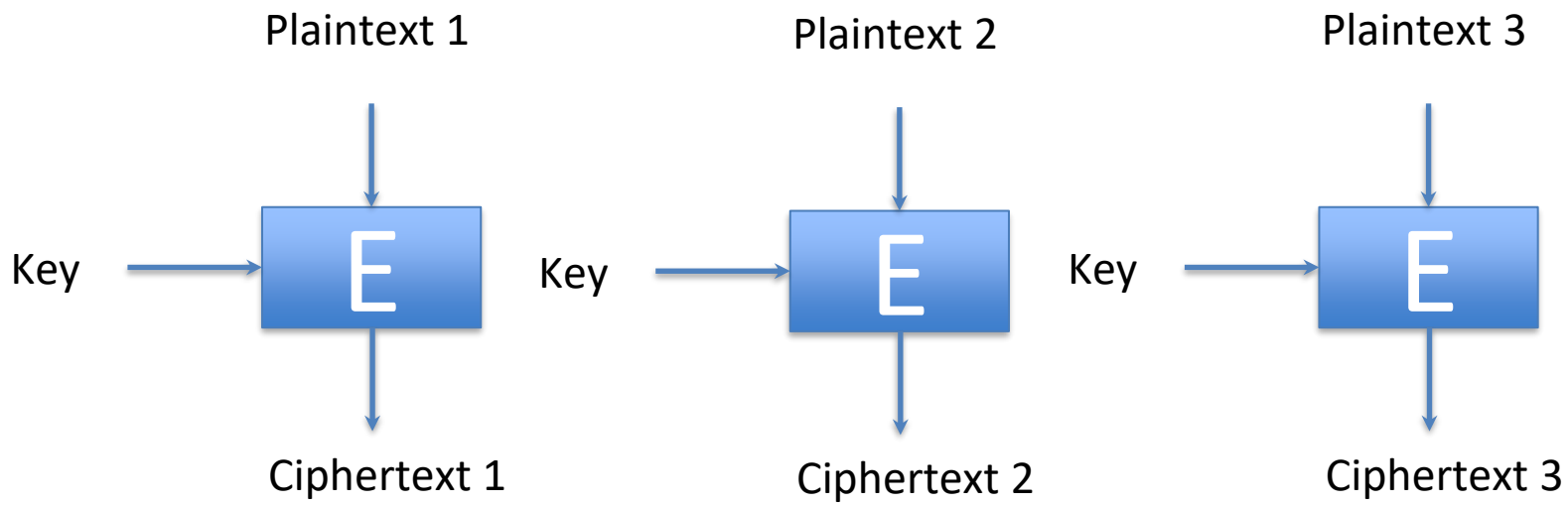
- The block sizes are all 128 bits
- The key sizes are different: 128, 192, 256
  - AES-128: 10 rounds
  - AES-192: 12 rounds
  - AES-256: 14 rounds
- The key scheduling algorithms are different

Modes of operation

- A mode of operation defines how a block cipher is applied to encrypt data
- We will cover size modes of operation
  - Electronic code book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Cipher feedback mode (CFB)
  - Output feedback mode (OFB)
  - Counter mode (CRT)

Mode 1: Electronic Codebook

- Simplest mode of operation

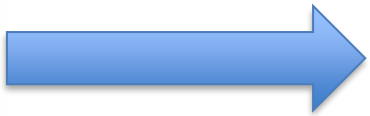


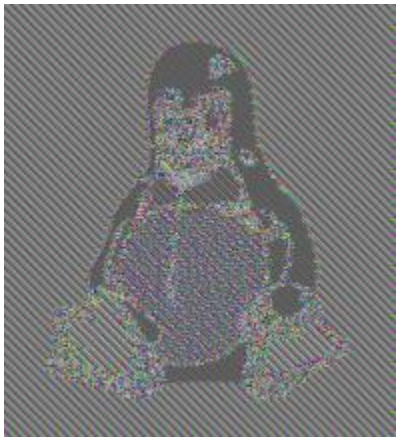
Properties of ECB

- ECB: deterministic operation
  - same plaintext-block input => same output
- Problematic in practice

Block correlations can leak info

ECB mode





Plaintext

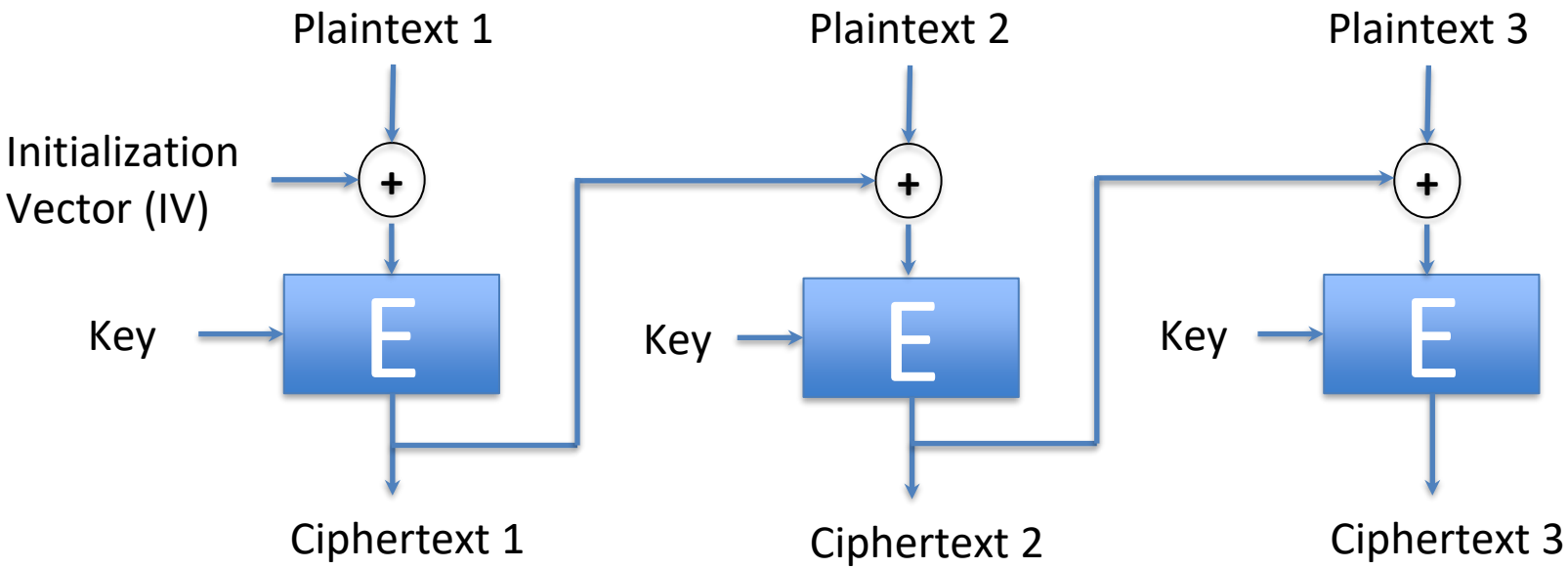
Ciphertext

(Wikipedia)

Check the SRTP standard for media encryption

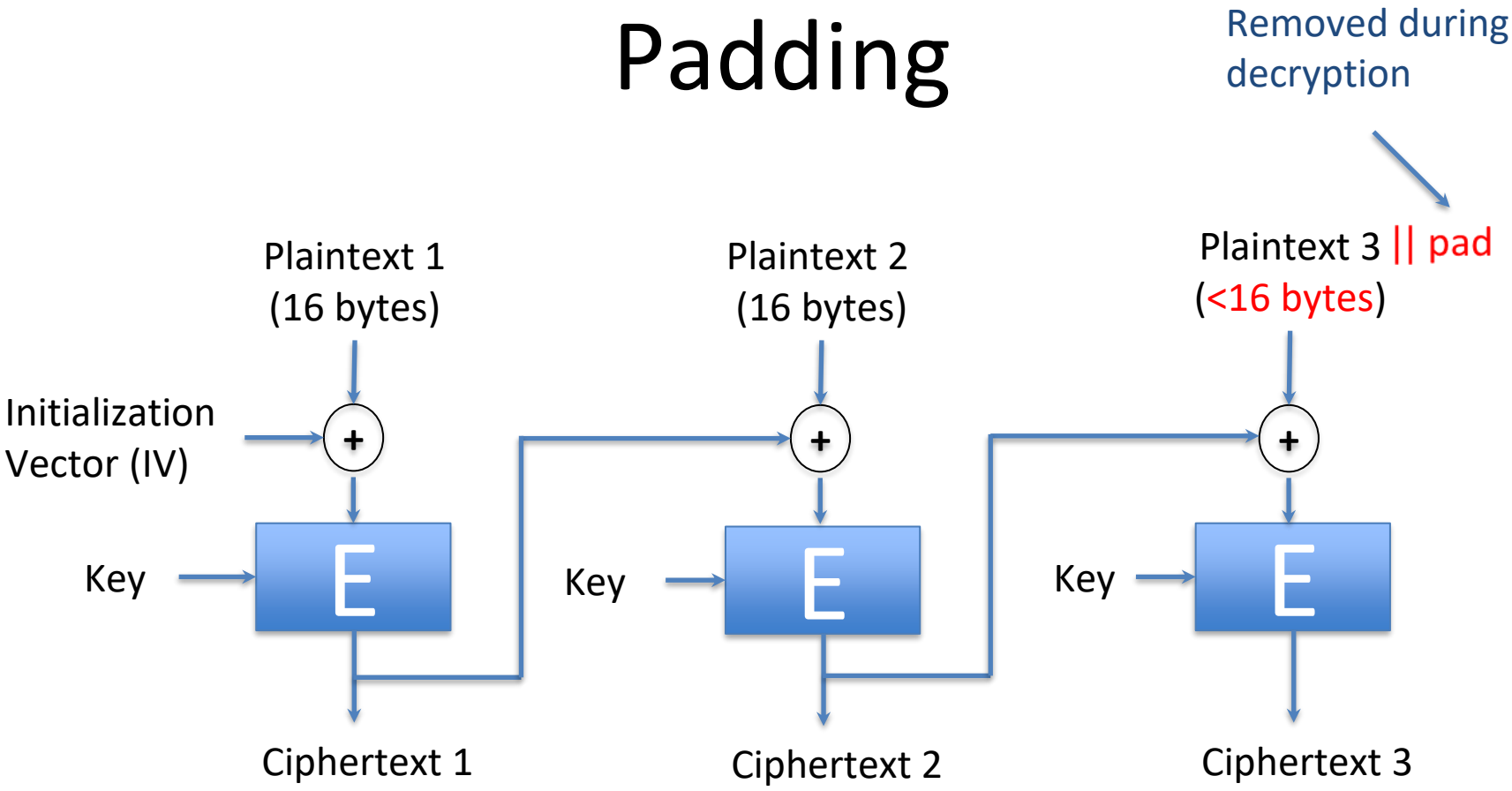
Mode 2: Cipher block chaining

- One of the most widely used modes



Ciphertext 1 = E(k, IV⊕ Plaintext 1) ⇒ Plaintex 1 = D(k, Ciphertext 1) XOR IV

Padding



PKCS7: for n>0, n byte pad is: n n n n .. n

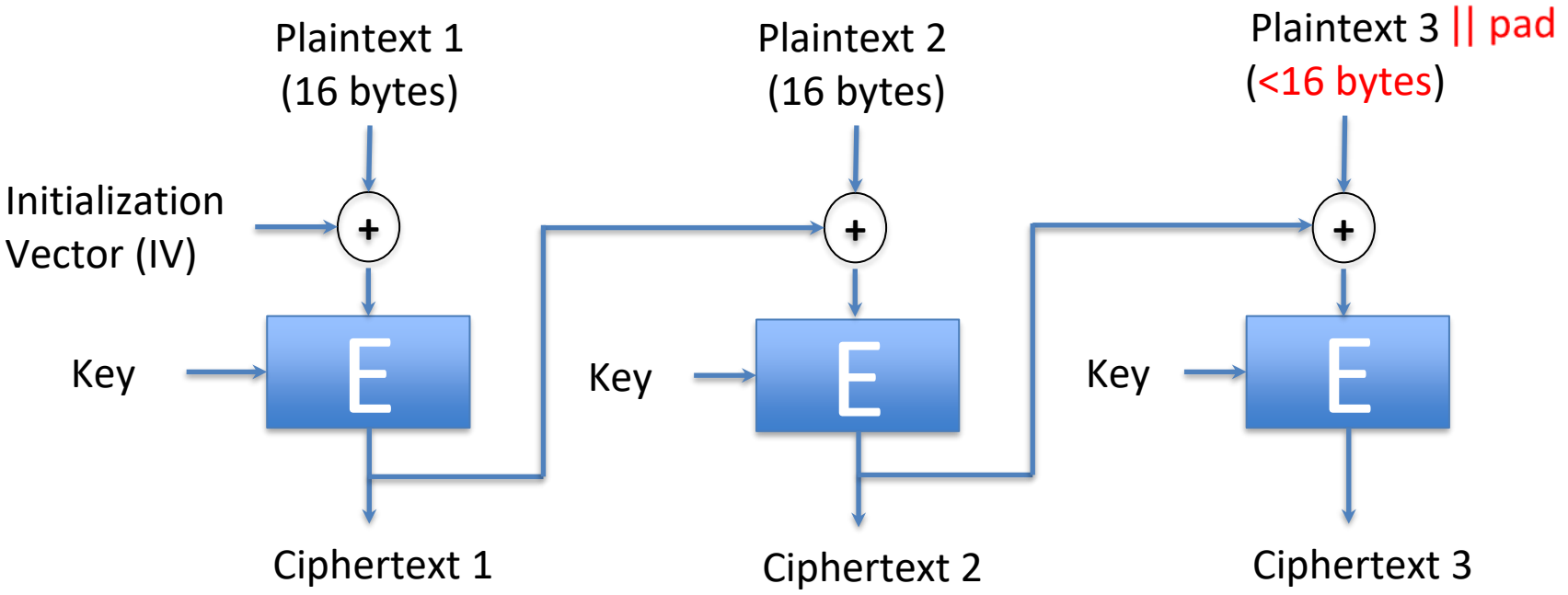
1 byte: 01
2 bytes: 02 02

Question: what if the plaintext is an **exact** multiple of block size? Do we still need padding?

Properties of CBC

- Because of a random IV, the same message inputs give different outputs (good!)
- If a plaintext block is changed, then all subsequent ciphertext blocks will be affected
  - A useful property to produce a *message authentication code (MAC)*
- If a whole block ciphertext is lost, CBC can synchronize by itself (but not if a byte is lost)
- Encryption cannot be parallelized (bad)

Padding error

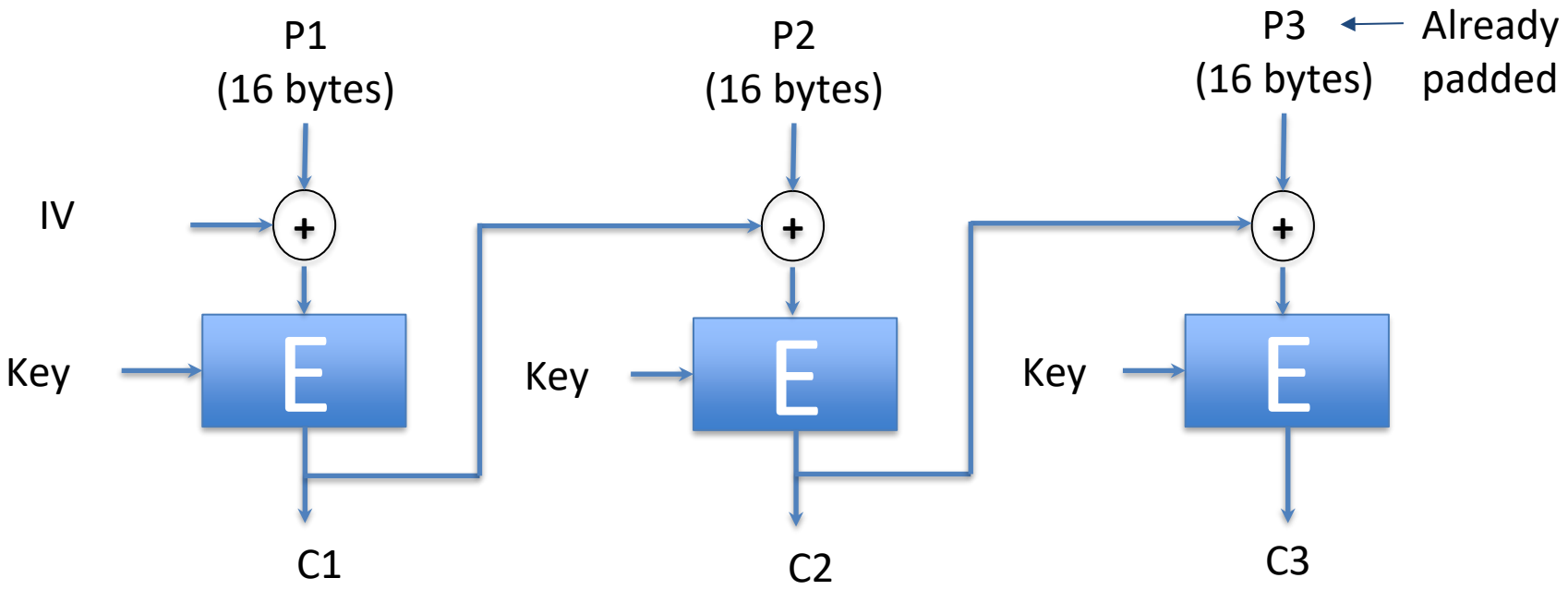


Example of PKCS7 padding: 01, 02 02, 03 03 03 …

Question: what if after decryption, the padding is found invalid?

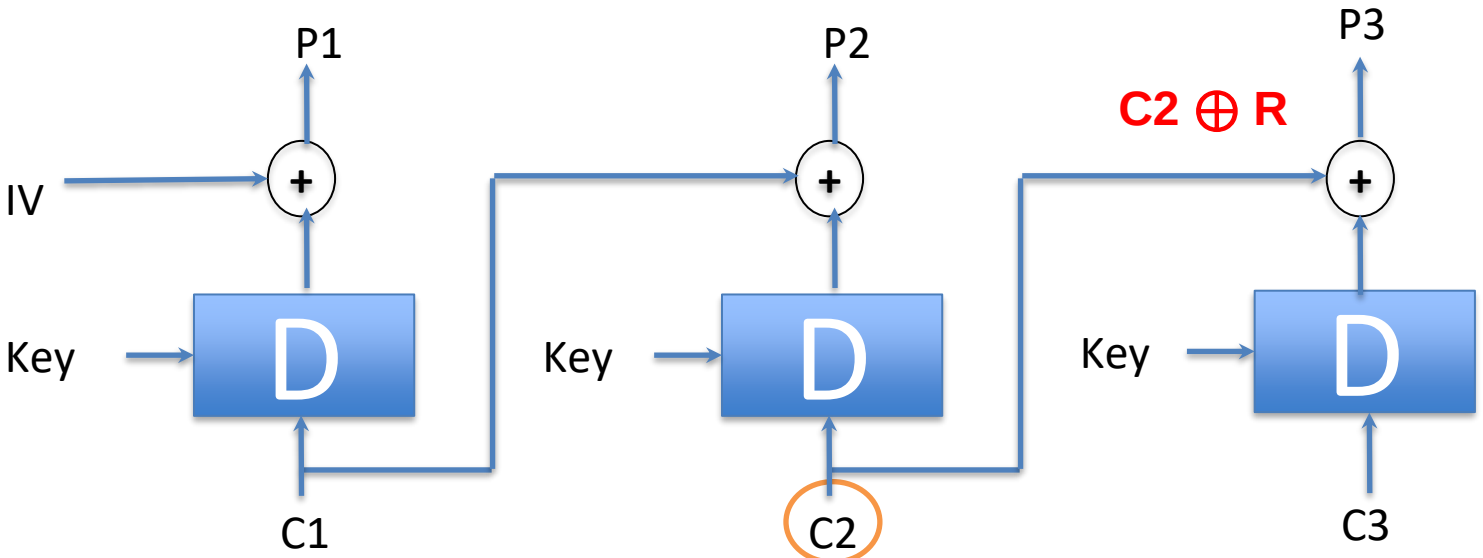
Padding oracle attack (Vandenay'02)

- Padded plain text in blocks: P1, P2, P3 …
- ciphertext : IV, C1, C2, C3 …
- If the decrypted pad is invalid, the server rejects data with an invalid-padding error



How does the attack work (1)?

- Feed modified ciphertext: IV, C1, **C2⊕R**, C3
- R={r0, r1, r2, r3, …, r15}; first 15 bytes random
- Try r15 (last byte of R) from 0 to 255
- 255 times decryption fails with invalid padding except once when (the modified output P3) ends with 01
  - Instead of P3=**C2**⊕D(K,C3), we have P3=(**C2⊕R**) ⊕ D(K,C3)

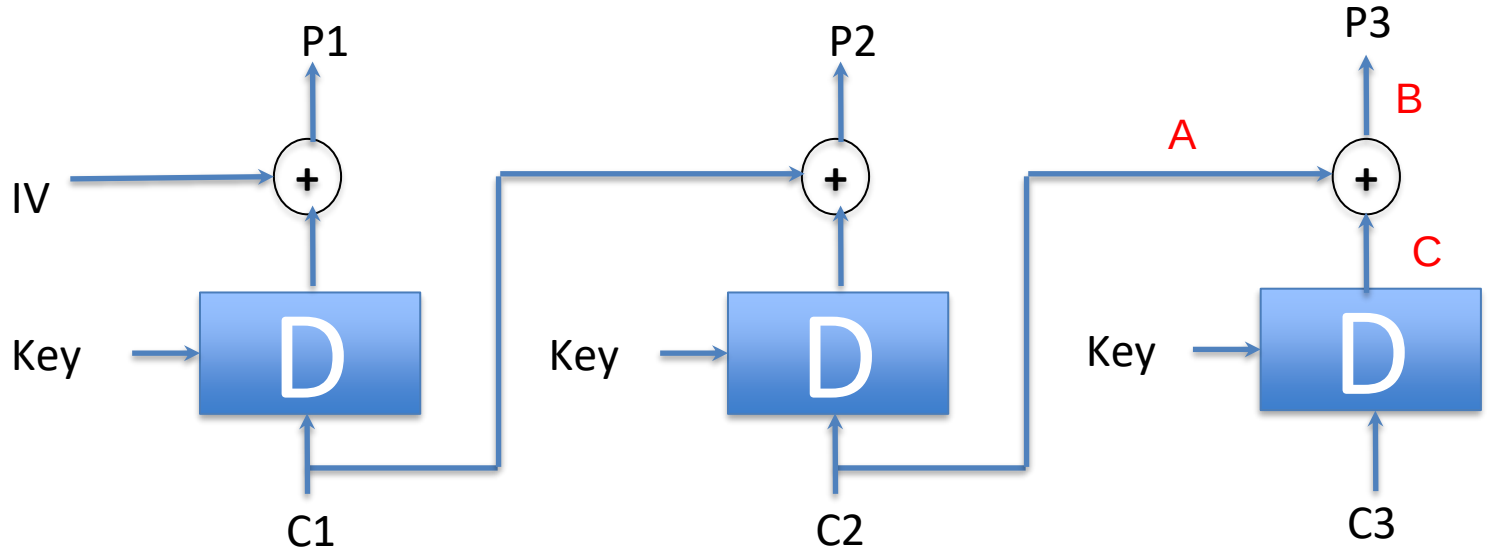


How does the attack work (2)?

- A = C2 ⊕ R
- C = D(K,C3)
- B = ?

- LB(A) = LB(C2) ⊕ r15
- LB(C) = LB(A) ⊕ LB(B)
- **LB(P3)** = LB(C2) ⊕ LB(C)
  = LB(A)⊕r15 ⊕ LB(A)⊕LB(B)

- Hence, the last byte of P3 is r15 ⊕ '01'

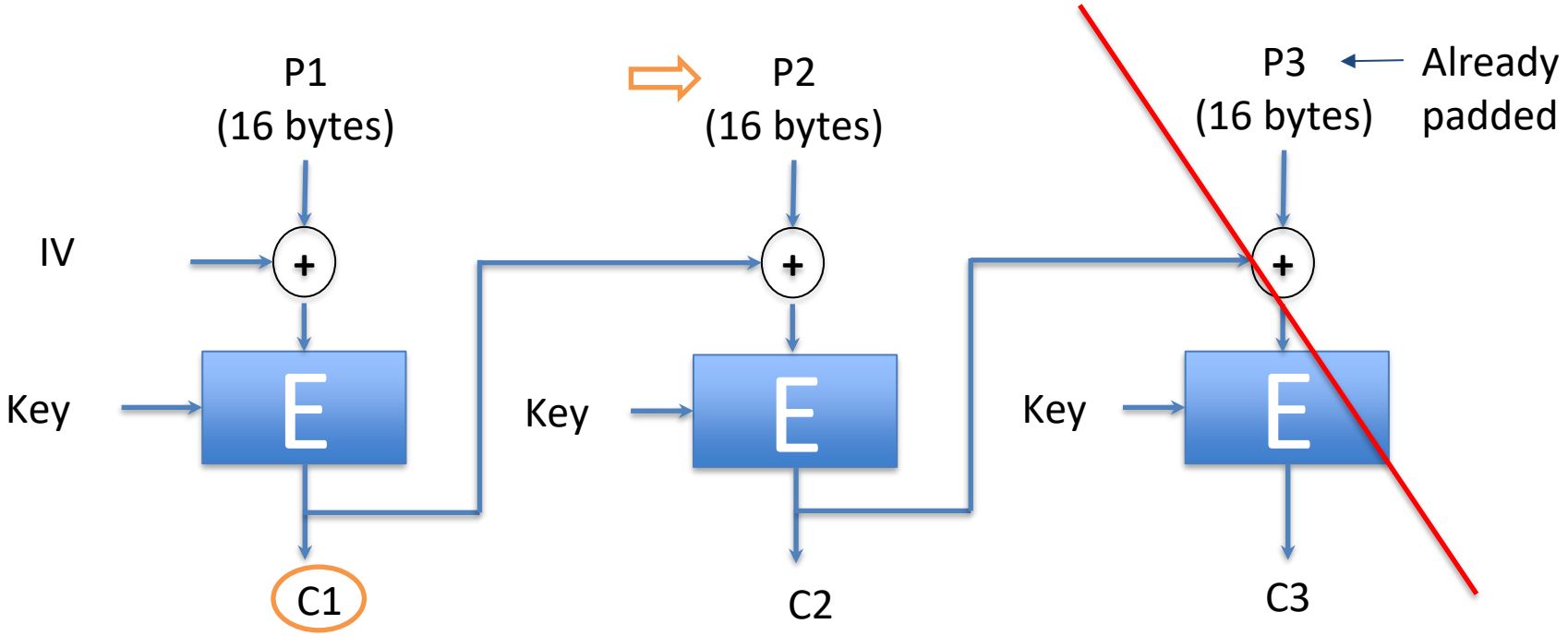


How does the attack work (3)?

- Recover the second last byte of P3
  - Fix last byte of R: r15 ⊕ '01' ⊕ '02' 02 02
  - Try r14 from 0 to 255
  - 255 failures except once when P3 ends '02 02'
  - Hence the 2nd last byte of **P3** is r14 ⊕ '02'
- Repeat the same to recover all bytes of P3
- Very efficient attack: The total calls to decryption oracle is 16 x 256 (instead of the theoretical 2^128 upper bound)

Padding oracle attack (Vandenay'02)

- How to recover the remaining blocks?

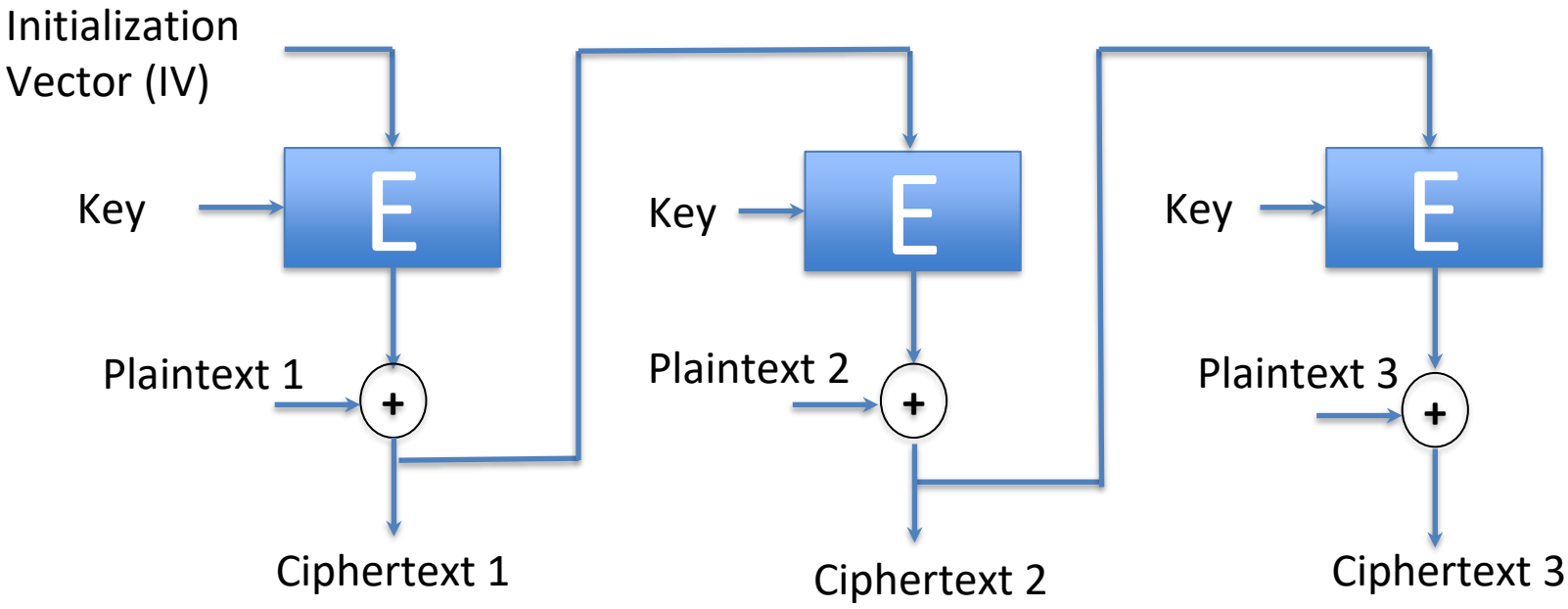


Countermeasures

- Remove the padding-error oracle
  - Show a generic error for decryption failure
  - Caveat: may still be subject to timing-attack, e.g., if invalid padding causes a quicker rejection
- Use authenticated encryption

Mode 3: Cipher feedback

- Turning a block cipher to a stream cipher



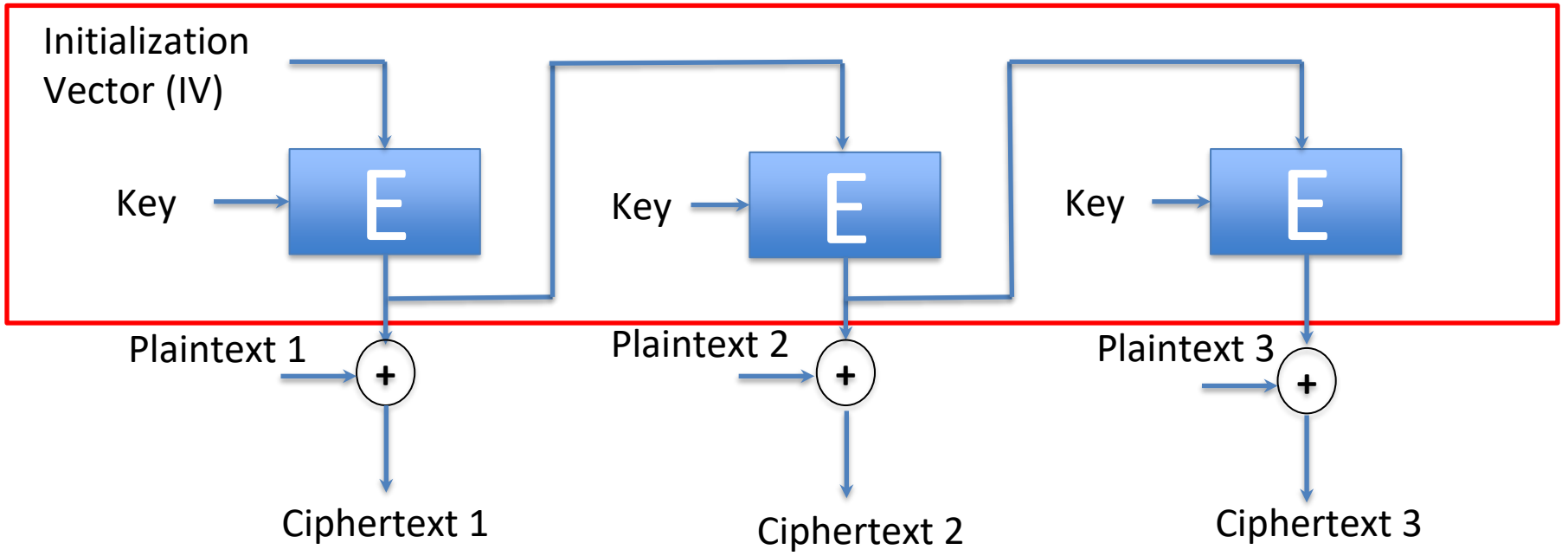
$C1 = E(IV, k) \oplus P1$, and $P1 = C1 \oplus E(IV, k)$

Properties of CFB

- If a whole blocksize of ciphertext is lost, CFB will synchronize by itself
- But if a byte or a bit is lost, CFB will lose synchronization – data can't be decrypted
- Only encryption operation is used for ENC/DEC.
- Can we parallelise encryption?
- Can we parallelise decryption?

Mode 4: Output Feedback (OFB)

- Essentially, a stream cipher

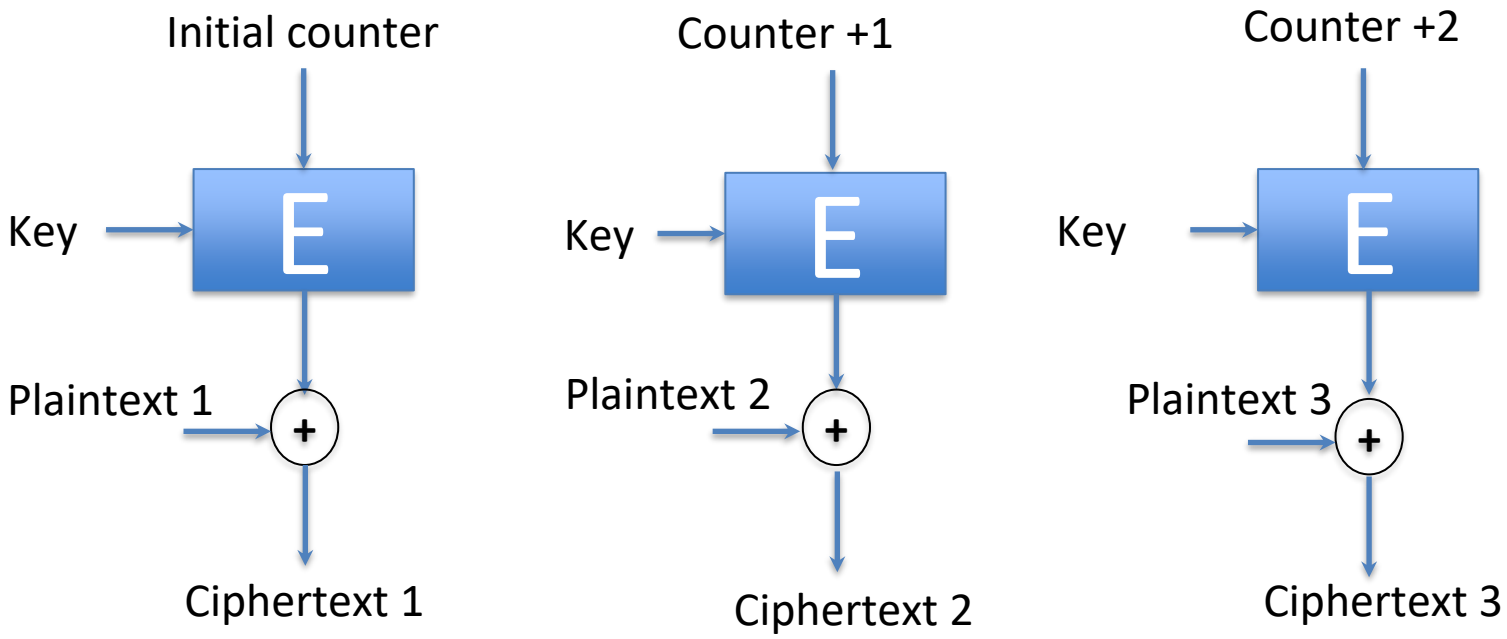


Properties of OFB

- The encryption and decryption operations are exactly the same (as one-time pad)

Mode 5: Counter (CRT)

- Getting increasingly popular (to replace CBC)



Properties of CRT

- Like OFB, CRT is essentially a stream cipher
- The encryption and decryption are the same.
- Both encryption and decryption can be parallelized (a big advantage over CBC)