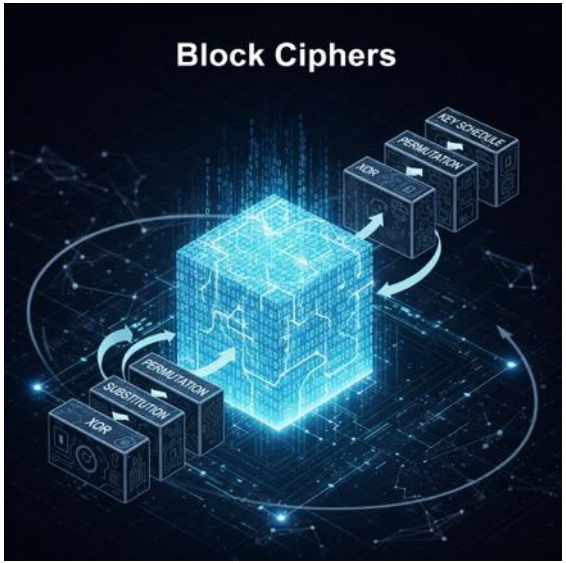
Block ciphers: an introduction

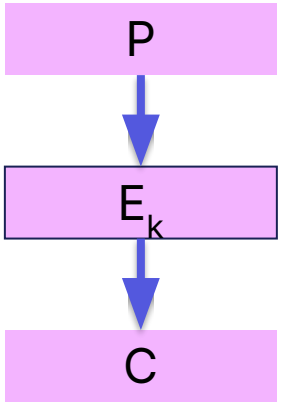


Definition of block cipher

Given
- block P of plaintext of h bits (h fixed)
- a key k of fixed # of bits

a cryptographic protocol E_k produces a block C of h bits, function of P and k (for a fixed E)

Note: lengths of both block and key (# of bits) are fixed (not necessarily equal)



Features

- Encrypts fixed-size blocks of plaintext into ciphertext (e.g., 128 bits)
- Uses a secret key to control a deterministic transformation
- Always produces the same ciphertext for the same input and key
- Decryption is the reverse transformation using the same key
- Acts as a reversible function and is a foundational building block in cryptography

Visual Analogy

- Imagine plaintext as a piece of clay
- The key is a custom mold
- Cipher shapes the clay into ciphertext
- Same mold = same result
- Emphasizes determinism and non-random transformation

Block vs. stream ciphers

- Block cipher: encrypts blocks (e.g., 128 bits) at a time
- Stream cipher: encrypts one bit or byte at a time using a keystream
- Analogy:
  - Stream = faucet
  - Block = stamping machine
- Block ciphers: better for structured data
- Stream ciphers: better for real-time communication

More detailed comparison

- **Block ciphers** work on fixed-size blocks (e.g., 128 bits). This makes them ideal for
  - Files: data is usually stored in known formats (PDF, DOCX, etc.)
  - Database records: entries have regular sizes and are accessed in blocks
  - Disk sectors: operating systems read and write data in 512-byte or 4096-byte blocks
- These are structured environments, where data is chunked in a way that fits naturally with the block cipher's operation

Desired properties

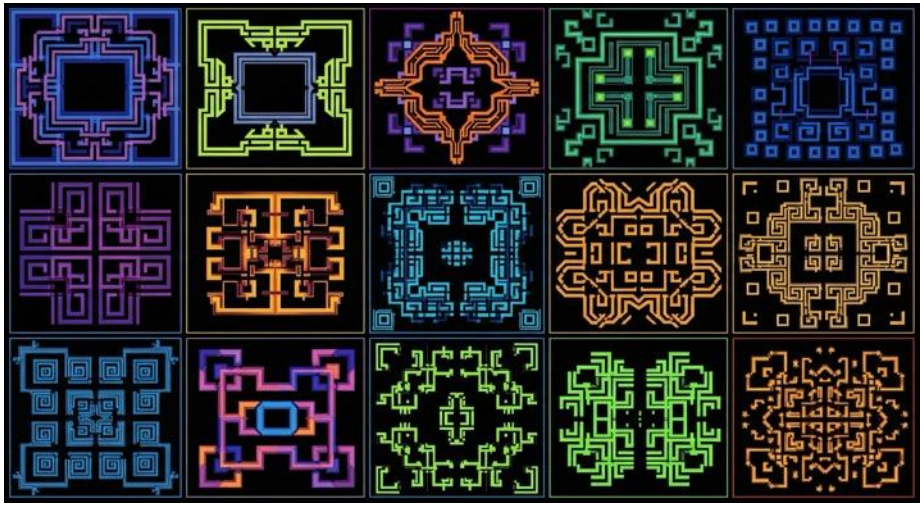
- **Invertibility**: must be able to recover original plaintext
- **Avalanche effect**: small changes = big output changes
- **Non-linearity**: avoids predictability and simple math reversal
- **Key sensitivity**: small key differences yield vastly different outputs

These features strengthen resistance to cryptanalytic attacks

Applications

- File encryption
- Disk encryption (full disk, partitions)
- Encrypted RAM/memory
- Encrypted communication packets
- Database field/row encryption
- Deployed in virtually all modern security systems

Block cipher designs from the late 20th century



Late-millennium block ciphers 1

- DES, 3-DES - (1976; 64-bit block, 56 bit key)
- RC-2 (1987)
  - designed for exporting cryptography within IBM Lotus Notes
  - 64-bit block, variable key size, vulnerable to an attack using 234 chosen plaintexts
- IDEA (1991)
  - 64-bit block, 128 bit key
  - Strong, only weakened variants have been broken, outdated
- Blowfish (1993)
  - 64-bit block size and a variable key length from 32 up to 448 bits
  - Still strong but deprecated

Late-millennium block ciphers 2

- RC5 (1994)
  - variable block size - 32, 64 or 128 bits - key size (0 to 2040 bits) and number of rounds (0 to 255). The original suggested choice of parameters were a block size of 64 bits, a 128-bit key and 12 rounds
  - Distributed.net has brute-forced RC5 messages encrypted with 56-bit and 64-bit keys and is working on cracking a 72-bit key; as of January 2025, 12% of the keyspace has been searched. At the current rate, it will take approximately 35-60 years to test every possible remaining key
  - distributed.net (or Distributed Computing Technologies, Inc. or DCTI) is a worldwide distributed computing effort that is attempting to solve large scale problems using otherwise idle CPU or GPU time. It is a non-profit organization
- AES (Rijndael, 2001)
  - 128-bit block, 128-256 bit key
  - Very strong

Historic note

- DES (data encryption standard) is a symmetric block cipher using 64-bit blocks and a 56-bit key
- Developed at IBM, approved by the US government (1976) as a standard. Size of key (56 bits) was apparently small enough to allow the NSA (US national security agency) to break it exhaustively even back in 70's.
- In the 90's it became clear that DES is too weak for contemporary hardware & algorithmics (Matsui "linear attack", requires only 2^{43} known plaintext/ciphertext pairs; in 1999 Deep Crack and distributed.net broke a DES key in 22 hours and 15 minutes)

More history

- The US government NIST (National Inst. of standards and technology) announced a call for an advanced encryption standard in 1997
- This was an international open competition. Overall, 15 proposals were made and evaluated, and 6 were finalists. Out of those, a proposal named Rijndael, by Daemen and Rijmen (two Belgians), was chosen in February 2001

Final ranking in NIST competition

Cipher	Security	Software Speed	Hardware Speed	Simplicity	Overall
Rijndael	8.5/10	9.5/10	9/10	9/10	9.0
Serpent	10/10	7/10	6/10	8/10	7.8
Twofish	9/10	8/10	8/10	7/10	8.0
RC6	8.5/10	8.5/10	8/10	6/10	7.8
MARS	9.5/10	6/10	7.5/10	5/10	7.0

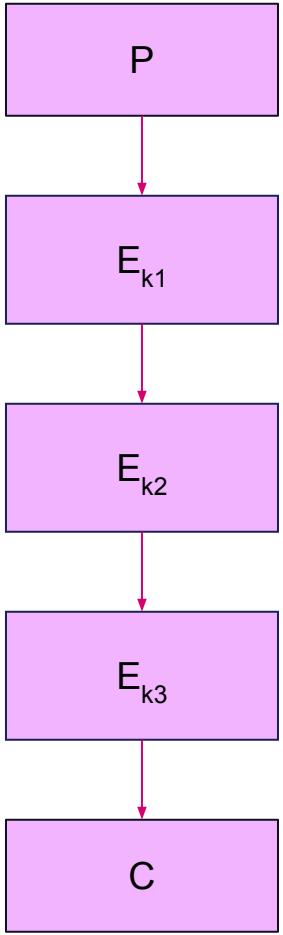
not official

Iterating DES

- Due to the weakness of DES the approach of iterating DES has been explored
- 3DES(M, k1, k2, k3) = encrypt plaintext, then encrypt ciphertext, then encrypt new ciphertext
  - 3DES(M, k1, k2, k3) = DES(DES(DES(M, k1), k2), k3), that gives the illusion of a security against brute-force of 56×3 bits = 168 bits
  - For the three levels 3DES often uses an encryptor, a decryptor, an encryptor (called in this case 3DES-EDE)
  - But attack Meet-in-the-Middle (next slide) reduces such length to 112 bits
- Iterating can be used (theoretically) with other ciphers

More on iterating

- Plaintext undergoes encryption repeatedly by underlying cipher
- Ideally, each stage uses a different key
- In practice triple cipher is usually
  $C = E_{k1}(E_{k2}(E_{k1}(P)))$ [EEE mode] or
  $C = E_{k1}(D_{k2}(E_{k1}(P)))$ [EDE mode]
- EDE is more common in practice



Ways of iterating DES

- Sometimes only two keys are used in 3DES
- Identical key must be at the beginning and the end
- Legal advantage (export license) due to smaller overall key size
- Why not using 2DES?

MITM – Meet-in-the-Middle

- Not to be confused with Man-in-the-Middle
- Requirements (assume EE)
  - Known plaintext/ciphertext pairs
  - 2^n encryptions + 2^n decryptions (2 keys of n bit), instead of 2^{2n} brute-force
  - 2^n memory space
- *Idea: try all possible 2^n encryptions of the plaintext and all possible 2^n decryptions of the ciphertext. Encryptions stored into a lookup table*
- *Check for a pair of keys that transform the plaintext in the ciphertext. Test pair on other pairs plaintext/ciphertext*
- Note: the method can be applied to all block ciphers

MITM – Meet-in-the-Middle

- MITM can be (theoretically) applied on *s* iterations
- If key-length is *sn*, brute-forcing the iteration doesn't require 2^{sn} attempts, but only (about) $2^{sn/2}$
- MITM needs huge memory
- For 3 or more iterations, memory becomes impractical
- Triple encryption is the realistic upper bound

Advanced Encryption Standard

AES

- Symmetric block cipher (block size: 128 bits = 16 bytes)
- Key lengths: 128, 192, or 256 bits
- Approved US standard (2001)
- Finite fields algebra

Noteworthy math results

Euler theorem

If a ed n are coprime positive integers (i.e. GCD(a, n) = 1) and $\varphi(n)$ is the is Euler's *totient function* (how many positive integers not greater than n are coprime with n)

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Bézout's identity

Let a and b be nonzero integers with greatest common divisor d. Then there exist signed integers x and y such that $ax + by = d$

x and y can be computed by the extended Euclidean algorithm

Challenge

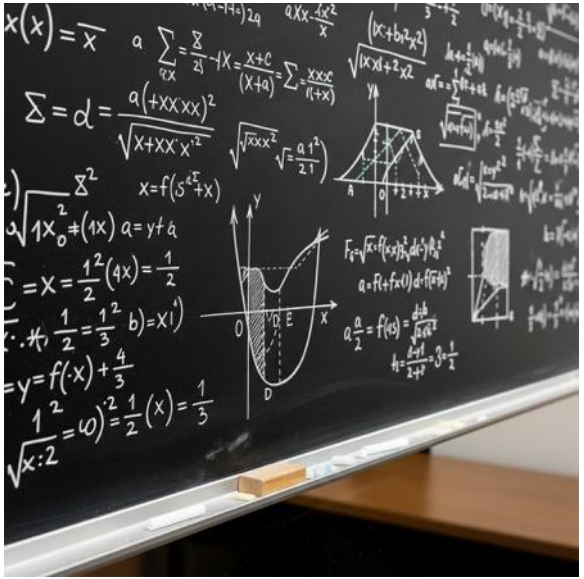
compute

$$2^{200} \bmod 127$$

using only pen, paper and Euler theorem...

Notation

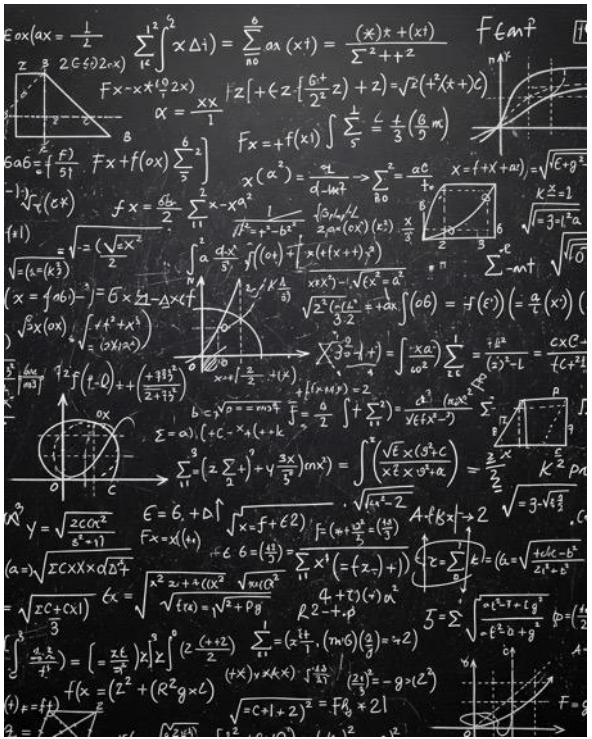
- Z_n = {[0], [1], [2], …, [n-1]}
  - here [i] is the equivalence class of integers congruent to i (mod n)
  - for brevity people often write Z_n = {0, 1, 2, …, n-1}
  - quotient set commonly called ring of integers modulo n
- Z_m^* = multiplicative group modulo m
  - natural numbers mod m that are relatively prime (co-prime) to m
  - $Z_m^* \subseteq Z_m$



Totient function

previous definitions

$\varphi(m)$ = Euler's totient function = $|Z_m^*|$ = size of the multiplicative group of Z_m



Galois field GF(p^k)

- **Theorem**: for every prime power p^k ($k \in \mathbb{N}^+$) there is a unique finite field containing p^k elements. These fields are denoted by GF(p^k)
- There are no finite fields with other cardinalities

Évariste Galois (1811-1832)

Implementing $GF(p^k)$ arithmetic

Theorem: Let f(x) be an irreducible polynomial of degree k over Z_p.

The finite field $GF(p^k)$ can be realized as the set of degree k-1 polynomials over Z_p, with addition and multiplication done modulo f(x)

Implementing GF(2^5)

Addition: bit-wise XOR (since 1+1=0)

	x^3+x+1	(0,1,0,1,1)
	+	
x^4+	x^3+x	(1,1,0,1,0)
x^4	+1	(1,0,0,0,1)

Implementing GF(2^5)

Multiplication: polynomial multiplication, and then remainder modulo the defining polynomial f(x)

Maple

```
> g(x):=(x^4+x^3+x+1)*(x^3+x+1);
```

$$g(x) := (x^4 + x^3 + x + 1)(x^3 + x + 1)$$

```
> f(x):=x^5+x^4+x^3+x+1;
```

$$f(x) := x^5 + x^4 + x^3 + x + 1$$

```
> rem(g(x),f(x),x);
```

$$1 + 3x^4 + x^3 + 2x$$

```
> % mod 2;
```

$$1 + x^4 + x^3$$

(1,1,0,1,1)*(0,1,0,1,1)

= (1,1,0,0,1)

For small size finite field, a lookup table is the most efficient method for implementing multiplication.

AES - Advanced Encryption Standard

- Symmetric block cipher
- Key lengths: 128, 192, or 256 bits
  - original Rijndael supports more lengths

Rationale

- Resistance to all known attacks
- Speed and code compactness
  - good for devices with limited computing power, e.g. smart cards
- Simplicity

AES Specifications

- Input & output block length: 128 bits
- State: 128 bits, arranged in a 4-by-4 matrix of bytes

$A_{0,3}$	$A_{0,2}$	$A_{0,1}$	$A_{0,0}$
$A_{1,3}$	$A_{1,2}$	$A_{1,1}$	$A_{1,0}$
$A_{2,3}$	$A_{2,2}$	$A_{2,1}$	$A_{2,0}$
$A_{3,3}$	$A_{3,2}$	$A_{3,1}$	$A_{3,0}$

each byte is viewed as an element in $GF(2^8)$

Input/Output: $A_{0,0}$, $A_{1,0}$, $A_{2,0}$, $A_{3,0}$, $A_{0,1}$,...

AES Specifications

- Key length: 128, 196, 256 bits

Cipher Key Layout: n = 128, 196, 256 bits, arranged in a 4-by-n/32 matrix of bytes

$K_{0,5}$	$K_{0,4}$	$K_{0,3}$	$K_{0,2}$	$K_{0,1}$	$K_{0,0}$
$K_{1,5}$	$K_{1,4}$	$K_{1,3}$	$K_{1,2}$	$K_{1,1}$	$K_{1,0}$
$K_{2,5}$	$K_{2,4}$	$K_{2,3}$	$K_{2,2}$	$K_{2,1}$	$K_{2,0}$
$K_{3,5}$	$K_{3,4}$	$K_{3,3}$	$K_{3,2}$	$K_{3,1}$	$K_{3,0}$

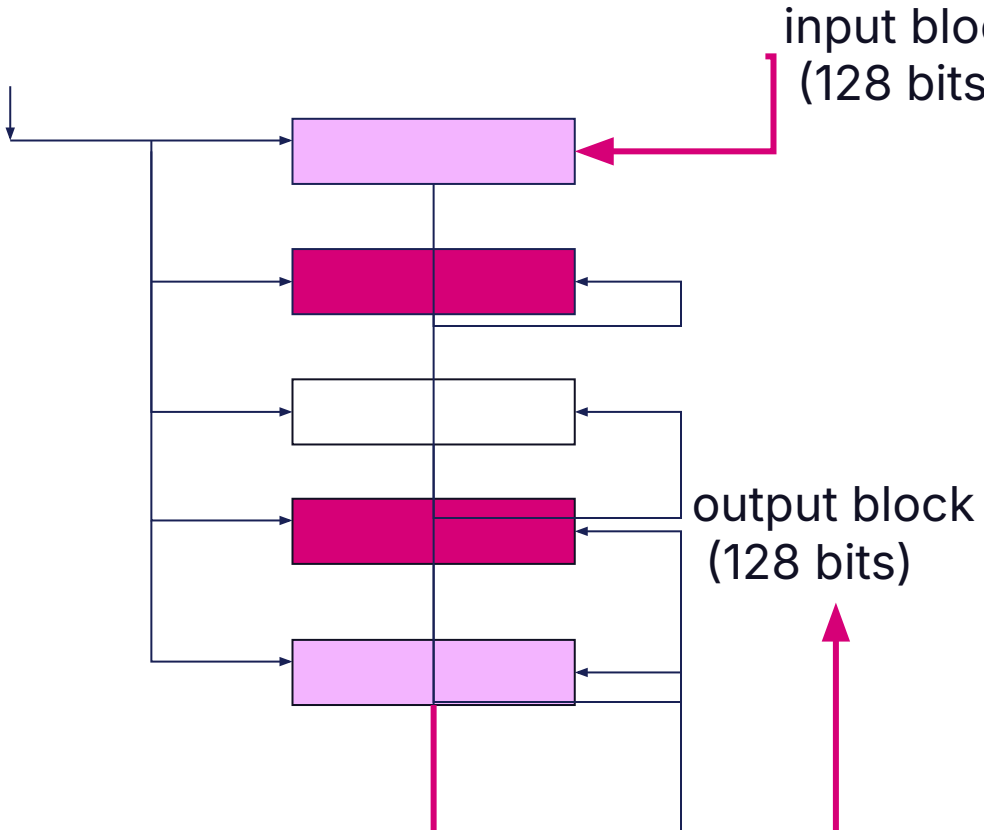
Initial layout: $K_{0,0}$, $K_{1,0}$, $K_{2,0}$, $K_{3,0}$, $K_{0,1}$,...

AES Specifications

High level code

```
AES(State, Key)
    KeyExpansion(Key, ExpandKey)
    AddRoundKey(State, ExpandKey[0])
    for (i = 1; i < R; i++) do
            Round(State, ExpandKey[i]);
    FinalRound(State, ExpandKey[R]);
```

Encryption: Carried out in rounds



Rounds in AES

- 128 bits AES uses 10 rounds, no shortcuts known for 6 rounds
  - The secret key is expanded from 128 bits to 10 round keys, 128 bits each
  - Each round changes the state, then XORs the round key (for longer keys, add one round for every extra 32 bits)

- Each rounds complicates things a little
- Overall, it seems infeasible to invert without the secret key (but easy given the key)

AES Specifications: One Round

Transform the state by applying:

1. Substitution
2. Shift rows
3. Mix columns
4. XOR round key

$A_{0,3}$	$A_{0,2}$	$A_{0,1}$	$A_{0,0}$
$A_{1,3}$	$A_{1,2}$	$A_{1,1}$	$A_{1,0}$
$A_{2,3}$	$A_{2,2}$	$A_{2,1}$	$A_{2,0}$
$A_{3,3}$	$A_{3,2}$	$A_{3,1}$	$A_{3,0}$

Substitution (S-Box)

- Substitution operates on every byte separately: $A_{i,j} \leftarrow A_{i,j}^{-1}$ (multiplicative inverse in GF(2^8) which is highly nonlinear)
- If $A_{i,j} = 0$, don't change $A_{i,j}$
- Clearly, the substitution is invertible

Cyclic shift of rows

$A_{0,3}$	$A_{0,2}$	$A_{0,1}$	$A_{0,0}$
$A_{1,2}$	$A_{1,1}$	$A_{1,0}$	$A_{1,3}$
$A_{2,1}$	$A_{2,0}$	$A_{2,3}$	$A_{2,2}$
$A_{3,0}$	$A_{3,3}$	$A_{3,2}$	$A_{3,1}$

no shift

shift 1 position

shift 2 positions

shift 3 positions

Clearly, the shift is invertible

Mixing Columns

- Every state column is considered as a polynomial over GF(2^8)
- Multiply with an invertible polynomial
- 03 x^3 + 01x^2 + 01x + 02 (mod x^4 + 1)
- Inv = 0B x^3 + 0D x^2 +09 x + 0E

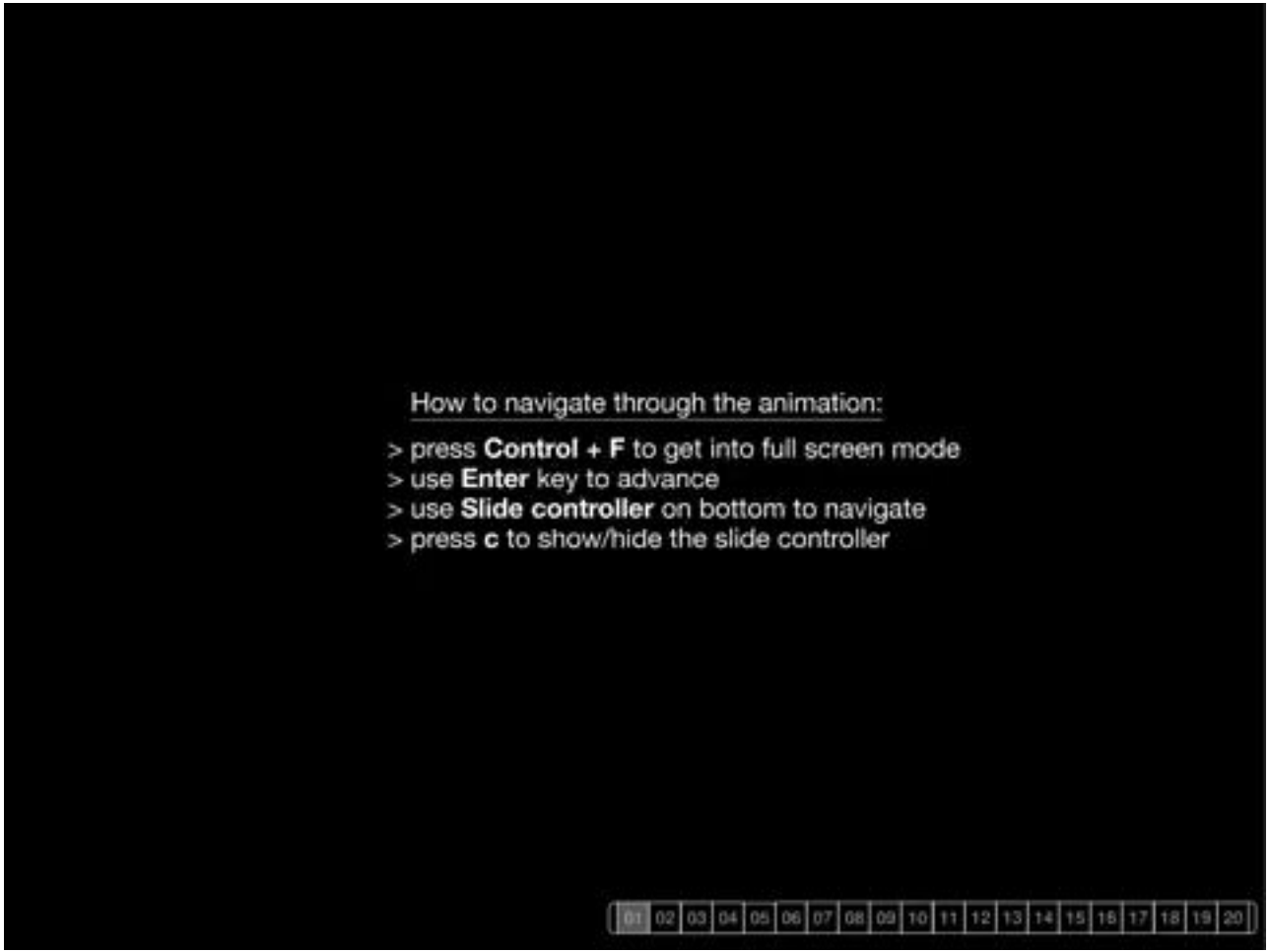
Round: SubBytes(State)
ShiftRows(State)
MixColumns(State)
AddRoundKey(State,ExpandedKey[i])

Key Expansion

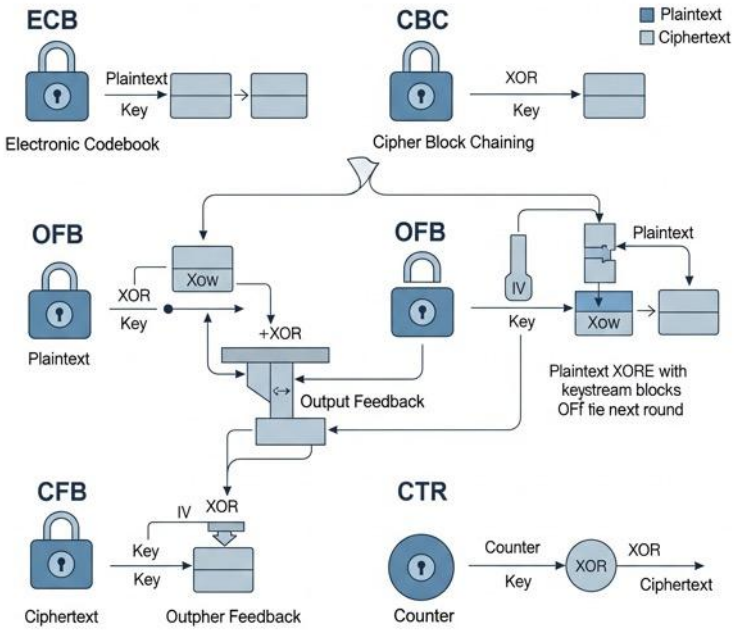
- Generate a “different key” per round
- Need a 4 × 4 matrix of values (over GF(2^8)) per round
- Based upon a non-linear transformation of the original key
- Details available: *The Design of Rijndael*, Joan Daemen and Vincent Rijmen, Springer

Animation

time 4' 25''



Introduction to block cipher modes of operation



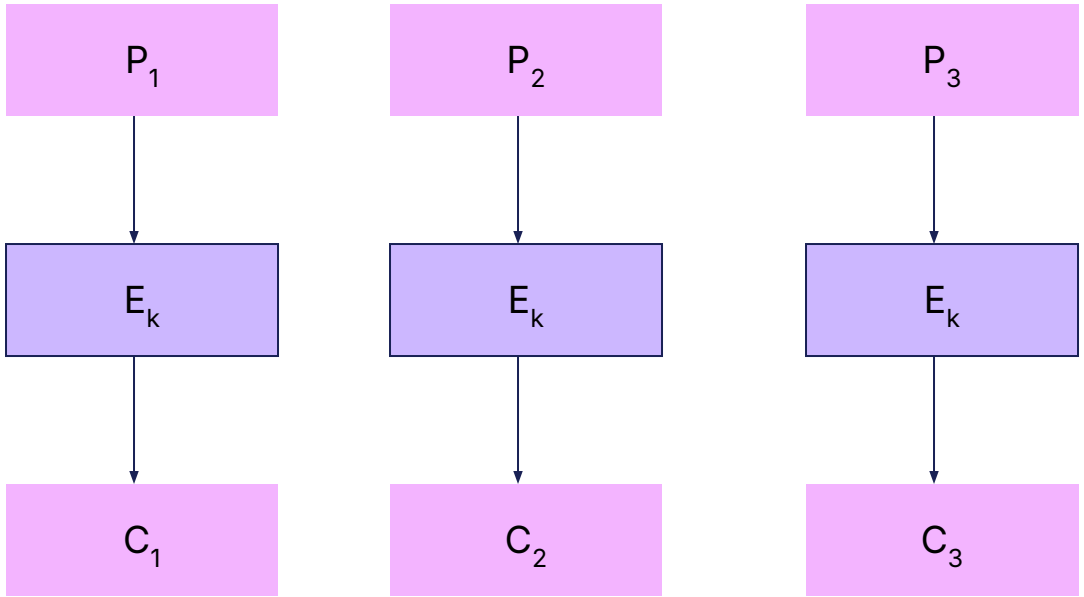
Block cipher modes of operation

- block ciphers operate on blocks of fixed length, often 64 or 128 bits
- because messages may be of any length, and because encrypting the same plaintext under the same key always produces the same output,

several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length

ECB Mode Encryption (Electronic Code Book)

encrypt each plaintext block separately



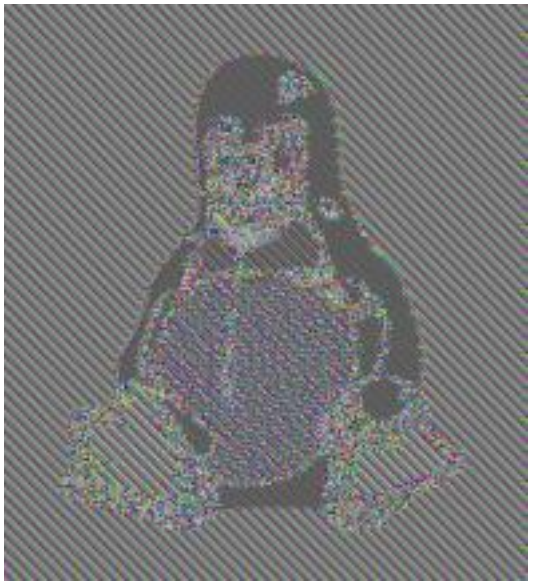
Properties of ECB

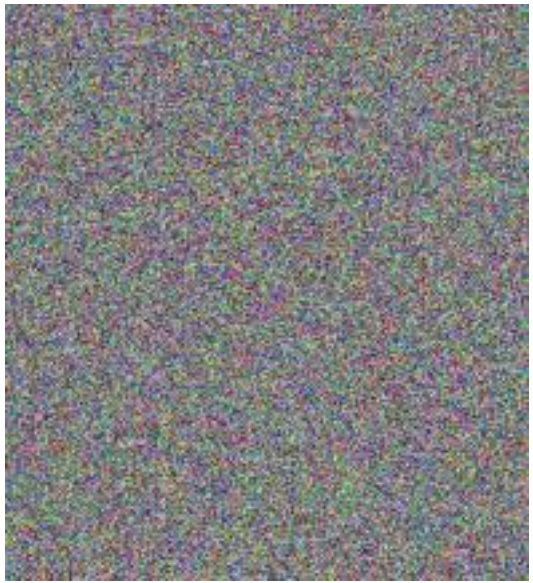
- Simple and efficient
- Parallel implementation possible
- Does **not** conceal plaintext patterns
- Active attacks are possible (plaintext can be easily manipulated by removing, repeating, or interchanging blocks)

Only teaching purposes

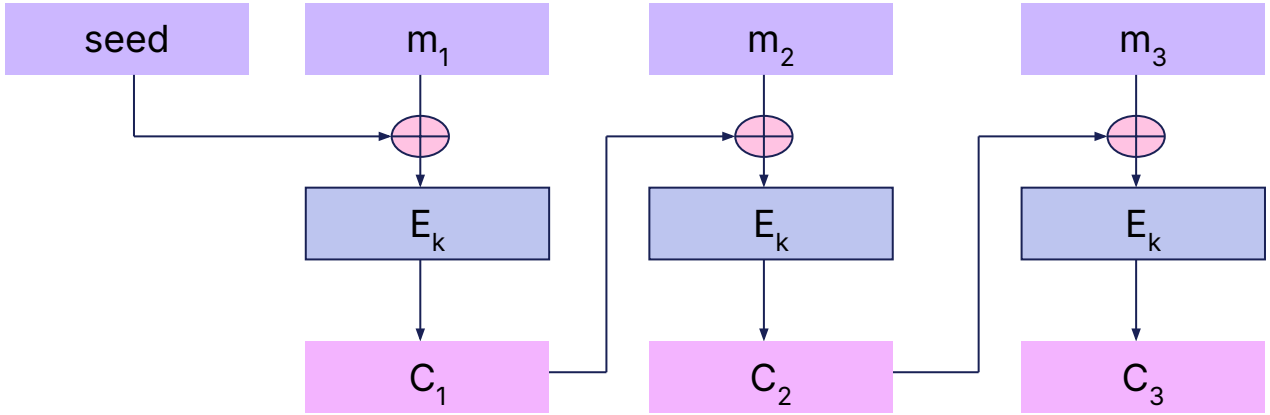
ECB: plaintext repetitions

plaintext ciphertext ECB good ciphertext



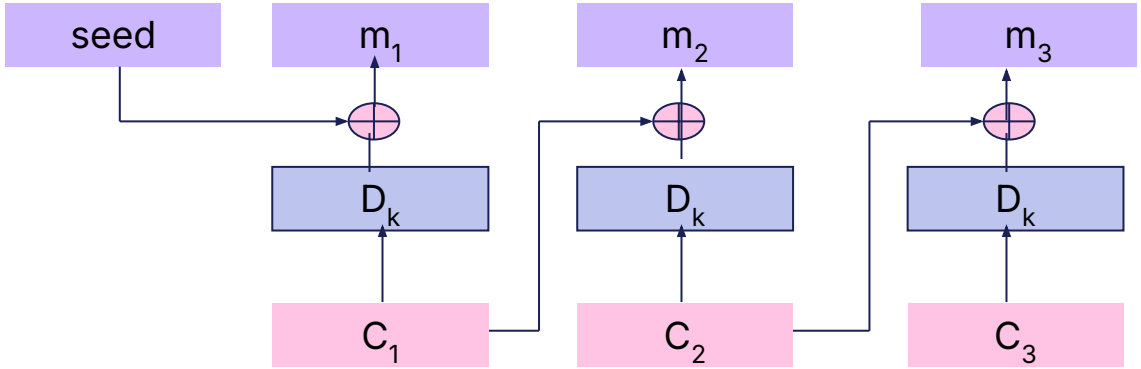


CBC (Cipher Block Chaining) mode



- Previous ciphertext is XORed with current plaintext before encrypting current block
- Seed (called initialization vector, or IV) is used to start the process; it can be sent without encryption
- Seed = 0 safe in most but NOT all cases (e.g., assume the file with salaries is sent once a month, with the same seed we can detect changes in the salaries) therefore a random seed is better

CBC (Cipher Block Chaining): decryption



- IF a transmission error changes one bit of $C_{(i-1)}$
- THEN block m_i changes in a predictable way (this can be exploited by adversary)
- BUT there are unpredictable changes in $m_{(i-1)}$;
- Solution: always use error detecting codes (for example CRC) to check quality of transmission

Properties of CBC

- Asynchronous stream cipher
- Errors in one ciphertext block don't propagate much
  - a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext
- Conceals plaintext patterns
- No parallel encryption
- Yes parallel decryption
- Plaintext cannot be easily manipulated
- Standard in many systems: SSL, IPSec etc.

More on CBC

- message must be padded to a multiple of the cipher block size
  - one way to handle this issue is ciphertext stealing (next slide)
- a plaintext can be recovered from just two adjacent blocks of ciphertext
  - therefore, decryption can be parallelized
  - usually a message is encrypted once, but decrypted many times

Ciphertext stealing

- general method that allows for processing of messages that are not evenly divisible into blocks
  - without resulting in any expansion of the ciphertext
  - at the cost of slightly increased complexity
- consists of altering processing of the last two blocks of plaintext, resulting in a reordered transmission of the last two blocks of ciphertext (and no ciphertext expansion)
- suitable for ECB and CBC
- from NIST website http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/ciphertext%20stealing%20proposal.pdf

Stealing procedures

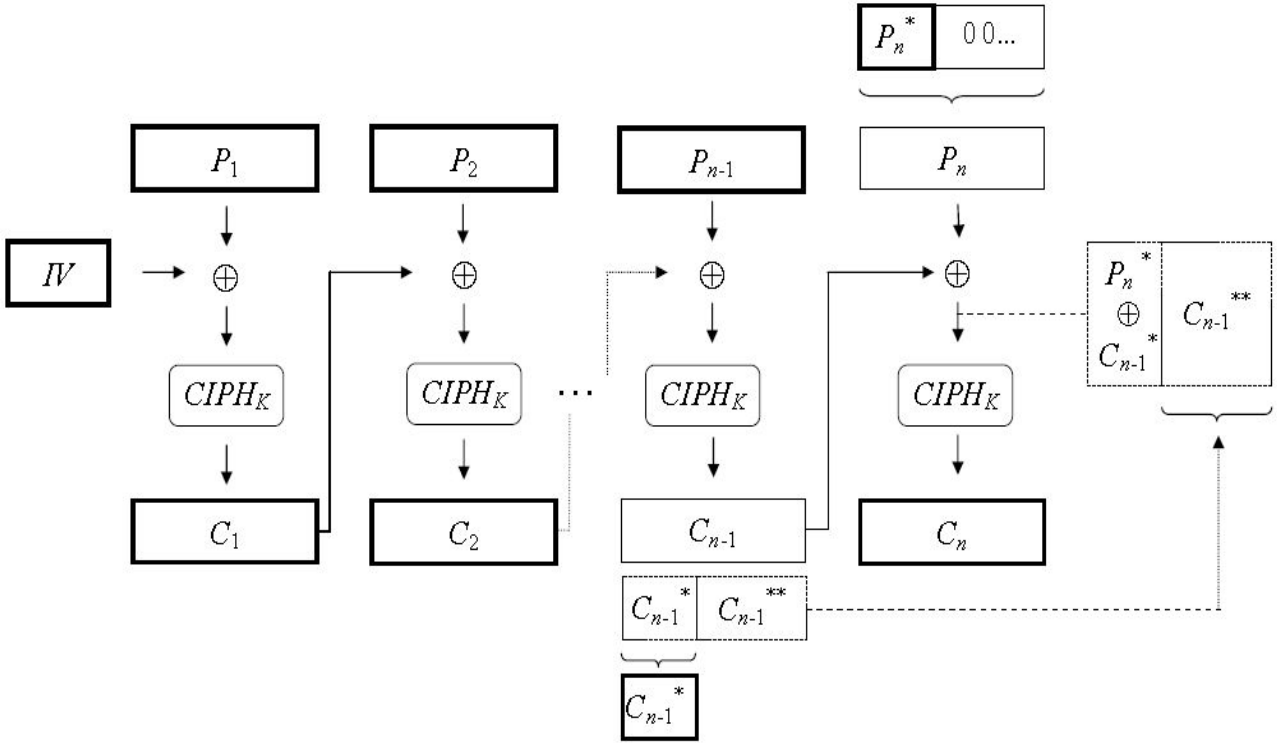
Encryption procedure

- Apply standard CBC encryption to all complete blocks
- If the last block is **partial**, process the previous full block normally
- Use the ciphertext of the previous block to fill the partial block (ciphertext stealing)
- Swap the last two ciphertext blocks
- Transmit the ciphertext as-is (possibly non-multiple of block size)

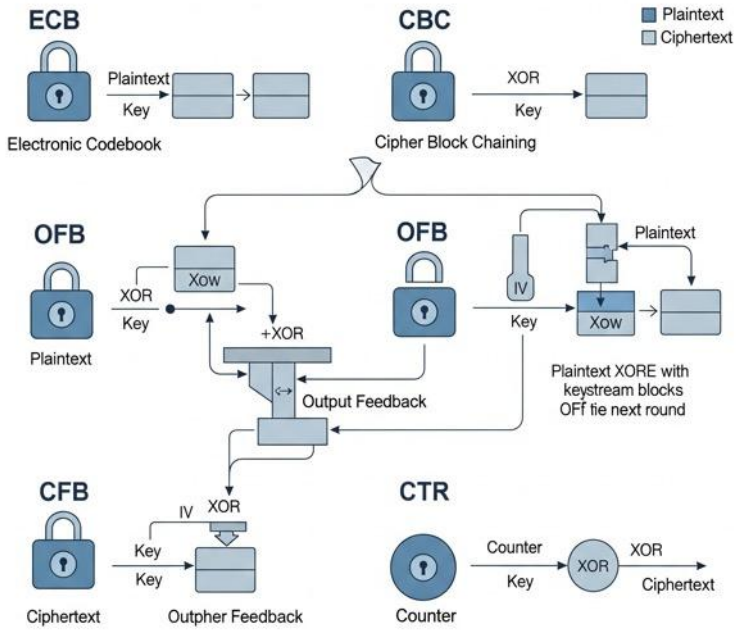
Decryption procedure

- Requires knowledge of original plaintext length
- Swap the last two ciphertext blocks back to original order
- Decrypt with CBC as usual
- Discard the extra bytes to recover the correct final block

Stealing diagram



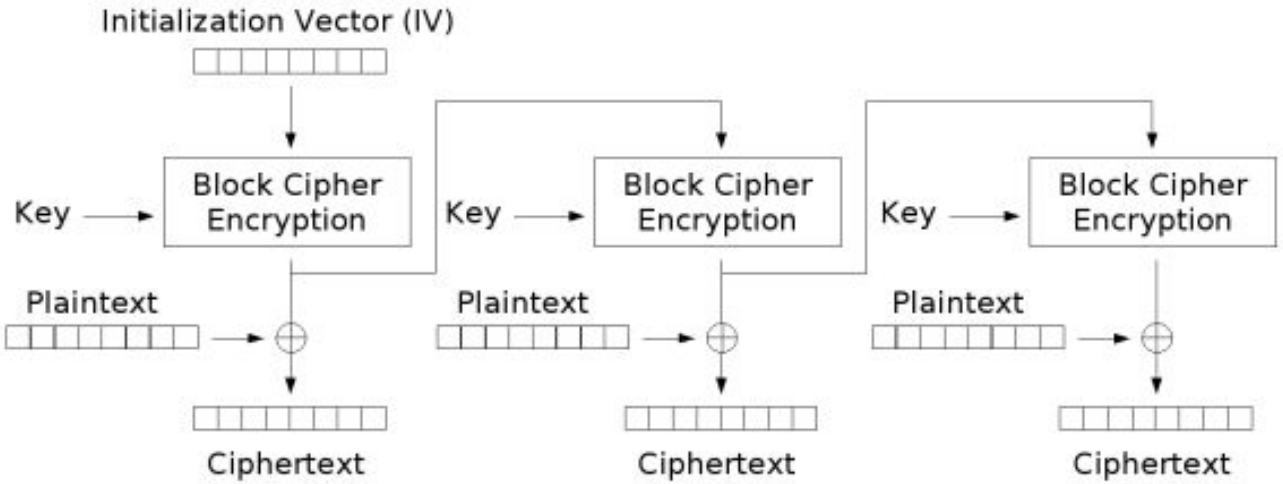
OFB and CTR


ECB
Plaintext
Key
Electronic Codebook
CBC
XOR
Key
Cipher Block Chaining
Plaintext
Ciphertext
OFB
XOR
Key
Plaintext
Xow
+XOR
Output Feedback
OFB
IV
Key
Xow
Plaintext
Plaintext XORE with
keystream blocks
OFf fie next round
CFB
IV
XOR
Key
Key
Ciphertext
Outpher Feedback
CTR
Counter
Key
XOR
XOR
Ciphertext
Counter

OFB Mode (Output FeedBack)

- Makes a block cipher into a synchronous stream cipher: it generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext
- Flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error correcting codes to function normally even when applied before encryption

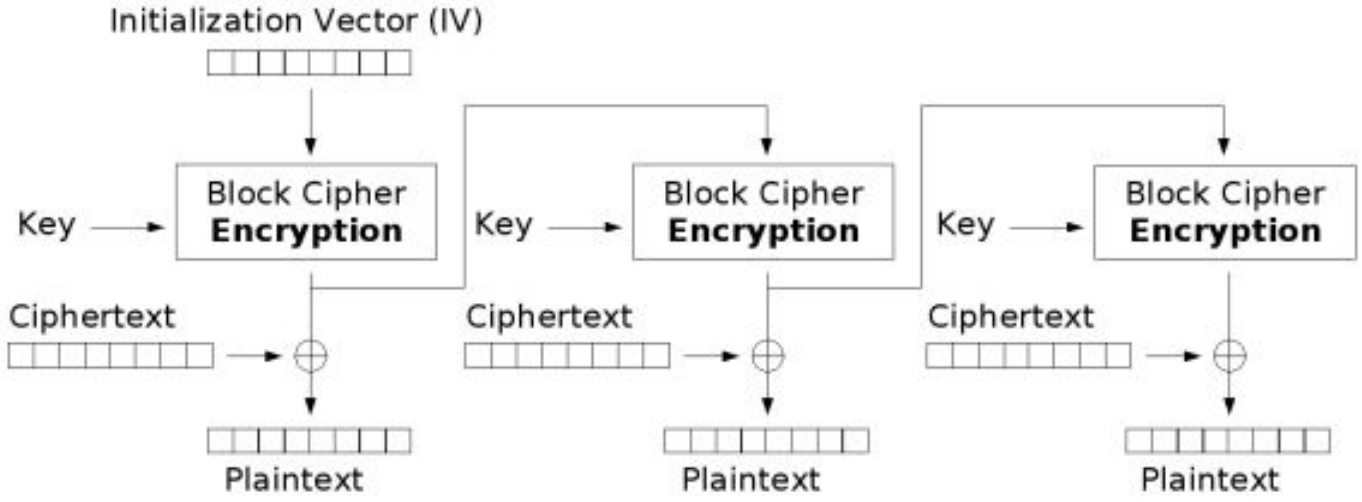
OFB scheme: encryption



Output Feedback (OFB) mode encryption

An initialization vector IV is used as a "seed" for a sequence of data blocks

OFB scheme: decryption



Output Feedback (OFB) mode decryption

Still **encryptors** because same keystream must be produced

OFB math

Because of the symmetry of the XOR operation, encryption and decryption are exactly the same

$$C_i = P_i \oplus O_i$$
$$P_i = C_i \oplus O_i$$
$$O_i = E_k(O_{i-1})$$
$$O_0 = \text{IV}$$

OFB mode

Discussion

- If E_k is public (known to the adversary) then initial seed must be encrypted
- If E is a cryptographic function that depends on a secret key, then initial seed can be sent in clear
- Initial seed must be modified for EVERY new encryption
- Extension: it can be modified in such a way that only k bits of each keystream block are used to compute the ciphertext (k-OFB)

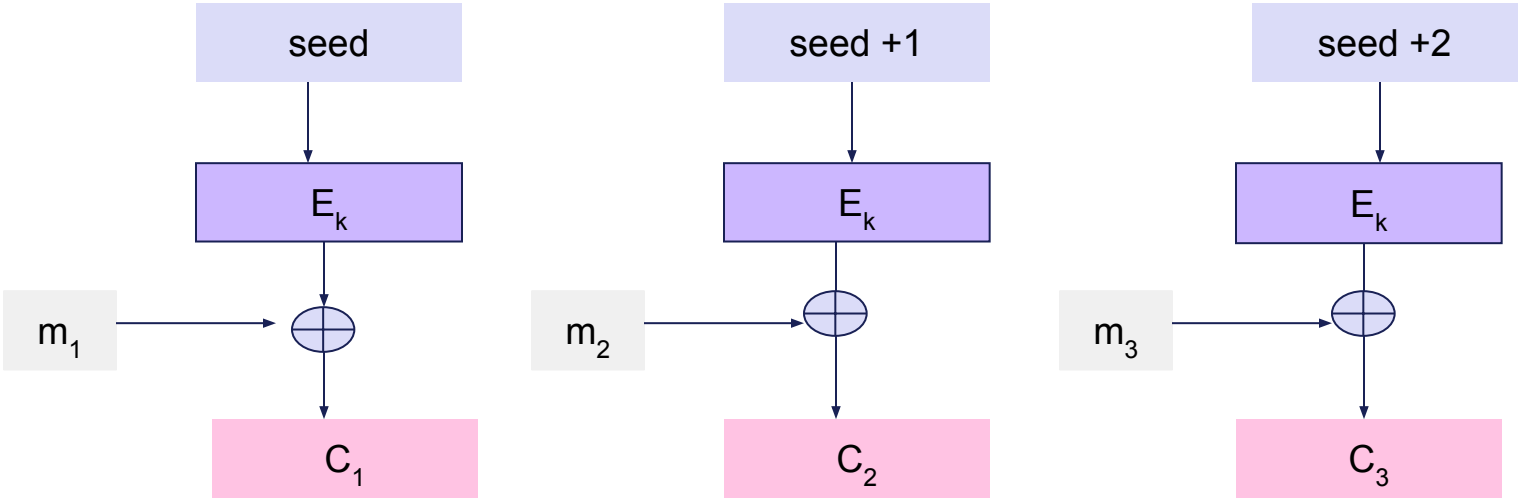
Properties of OFB

- Synchronous stream cipher
- Errors in ciphertext do not propagate
- Pre-processing is possible
- Conceals plaintext patterns
- No parallel implementation known
- Active attacks by manipulating plaintext are possible

CTR (Counter Mode)

- Aka Integer Counter Mode (ICM) and Segmented Integer Counter (SIC) mode
- turns a block cipher into a stream cipher: it generates the next keystream block by encrypting successive values of a "counter"
    - counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular
- the usage of a simple deterministic input function raised controversial discussions
- has similar characteristics to OFB, but also allows a random-access property during decryption
- well suited to operation on a multi-processor machine where blocks can be encrypted in parallel

CTR (Counter Mode)



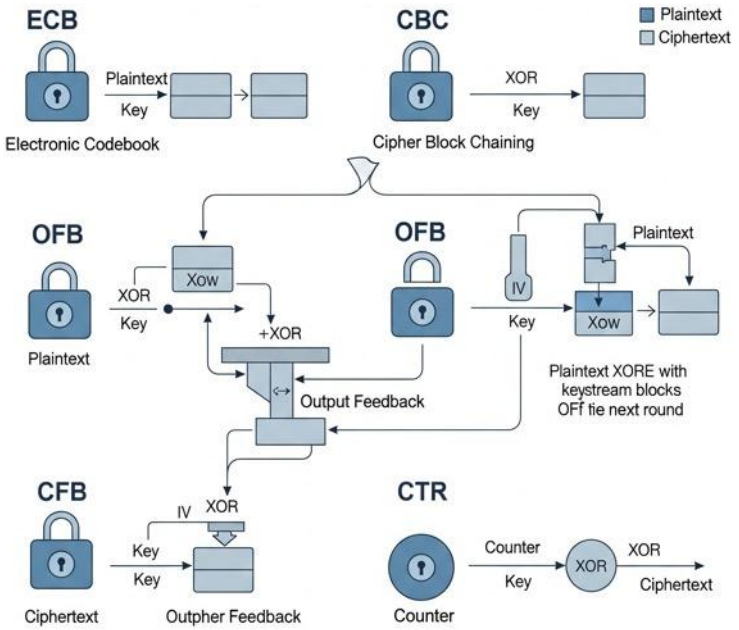
Similar to OFB

- There are problems in repeated use of same seed (like OFB)
- CTR vs OFB: using CTR you can decrypt the message starting from block *i* for any *i*

On the IV

- Most modes (except ECB) require an initialization vector, or IV
  - sort of "dummy block" to kick off the process for the first real block, and to provide some randomization for the process
  - no need for the IV to be secret, in most cases, but it is important that it is **never reused** with the same key
- For CBC reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages
- In CBC mode, the IV must, in addition, be unpredictable at encryption time
  - there is a TLS CBC IV attack
- For OFB and CTR, reusing an IV destroys security

Analysis and comparison of block modes of operations



Recap on block modes of operations

- There are many block operation modes
  - more than 4
- Used to overcome the limitation that the same algorithm with the same key always encrypts the same plaintext block to the same ciphertext
- Since they are many, how to make a choice?

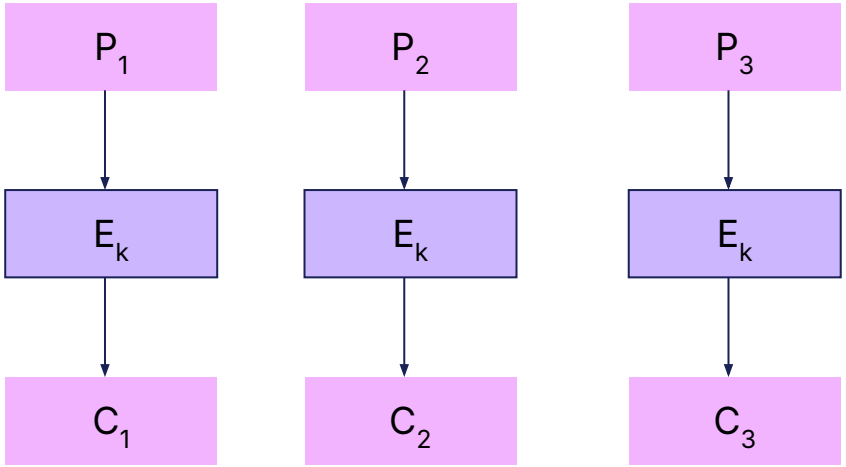
Methodology of analysis

Some questions are relevant for most modes

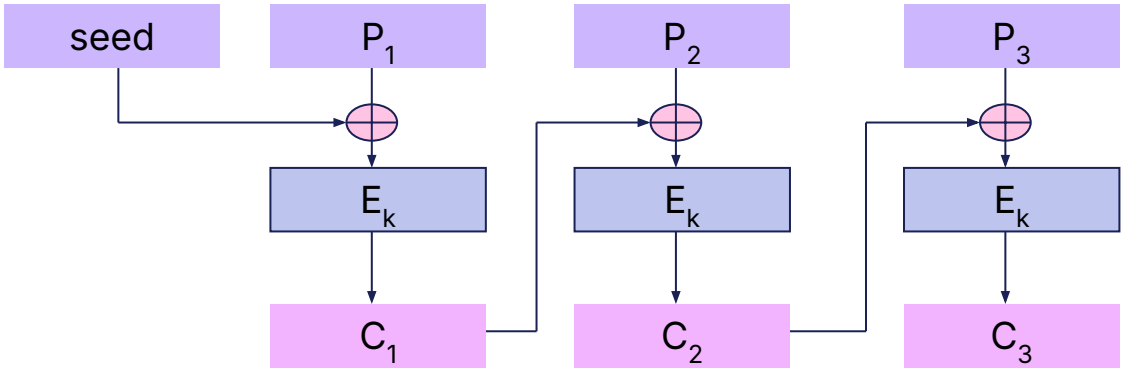
1. Can it encrypt in parallel?
2. Can it decrypt in parallel?
3. Is preprocessing useful?
4. Does it support ciphertext direct addressing?
5. What consequences for a ciphertext bit flipping?
6. Does it turn block encryption to stream encryption?

Recap schemes 1

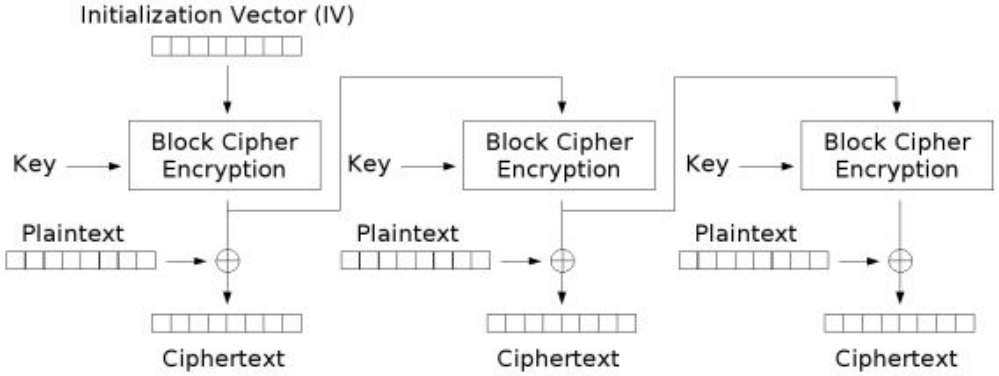
ECB



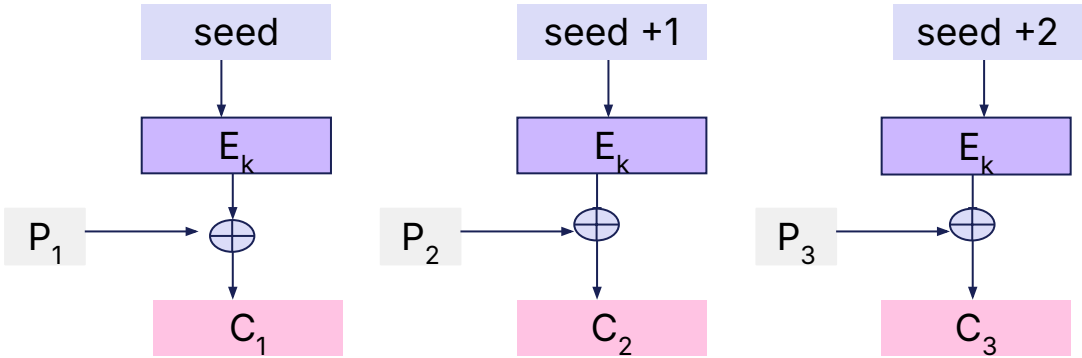
CBC



Recap schemes 2



Output Feedback (OFB) mode encryption



CTR

Comparison

1. Can it encrypt in parallel?
2. Can it decrypt in parallel?
3. Is preprocessing useful?
4. Does it support ciphertext direct addressing?
5. What consequences for a ciphertext bit flipping?
6. Does it turn block encryption to stream encryption?

	1	2	3	4	5	6	secure
ECB	Y	Y	N	Y	block ruined	N	N
CBC	N	Y	N	N	block ruined, subsequent with bit flipped	asynchronous	Y
OFB	N	N	Y	N	bit flip	synchronous	Y
CTR	Y	Y	Y	Y	bit flip	synchronous	Y