Block Ciphers and Modes of Operation

ITIS 6200/8200 Fall 2023 - Lecture 3

Announcements

ITIS 6200 / 8200

- Assignment 1 release later today
    - Due Sep.12th 11:59 PM EST
    - Start early!

- Office hours start this week
    - Instructor: Tu / Th 1:45-3 PM
        - Woodward Hall, 330D
    - TA (Vineeth): Mon 7-8 PM
        - COED-065
        - https://meet.google.com/xde-dmob-ipa
    - TA (Tarun): Wed 6-7 PM
        - Woodward Hall, 309
        - https://meet.google.com/irn-doeo-uhr?hs=122&authuser=0

Recap

- What's cryptography?
  - Communicating securely over insecure channels
  - Provide rigorous guarantees of security
- Definitions
  - Alice and Bob want to send messages over an insecure channel. Eve can read anything sent over the insecure channel. Mallory can read or modify anything sent over the insecure channel.
  - We want to ensure confidentiality (adversary can't read message), integrity (adversary can't modify message), and authenticity (prove message came from sender)
  - Crypto uses secret keys. Kerckhoff's principle says to assume the attacker knows the entire system, except the secret keys.
  - There are several different threat models. We'll focus on the chosen plaintext attack (CPA), where Eve tricks Alice into encrypting some messages.

Recap

- IND-CPA security
  - Even if Eve can trick Alice into encrypting some messages of Eve’s choosing, given the encryption of either M_0 or M_1, Eve cannot distinguish which message was sent with probability greater than 1/2.
  - We can use the IND-CPA game to test for IND-CPA security
- One-time pads
  - Symmetric encryption scheme: Alice and Bob share a secret key.
  - Encryption and decryption: Bitwise XOR with the key.
  - No information leakage if the key is never reused.
  - Information leaks if the key is reused.

Block Ciphers

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	● One-time pads ● Block ciphers with chaining modes (e.g. AES-CBC)	● RSA encryption ● ElGamal encryption
Integrity, Authentication	● MACs (e.g. HMAC)	● Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)

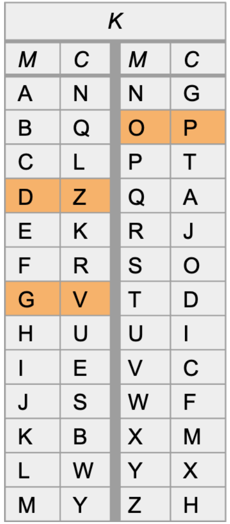
- Key management (certificates)
- Password management

Intro to Block Ciphers

- Types of symmetric key crypto
- Use a fixed length key to encrypt a fixed length block of data
- For example, a 64-bit block of data and a 128-bit key

Intro to Block Ciphers

- Similar to a substitution cipher
  - Much larger alphabet!



K			
M	*C*	*M*	*C*
A	N	N	G
B	Q	O	P
C	L	P	T
D	Z	Q	A
E	K	R	J
F	R	S	O
G	V	T	D
H	U	U	I
I	E	V	C
J	S	W	F
K	B	X	M
L	W	Y	X
M	Y	Z	H

- Example: if we have a 64-bit block cipher, then our substitution table has 2^{64} entries ($1.8 * 10^{19}$)
  - That's a big substitution table!

- Goal of a block cipher: Do this with an algorithm and a small key

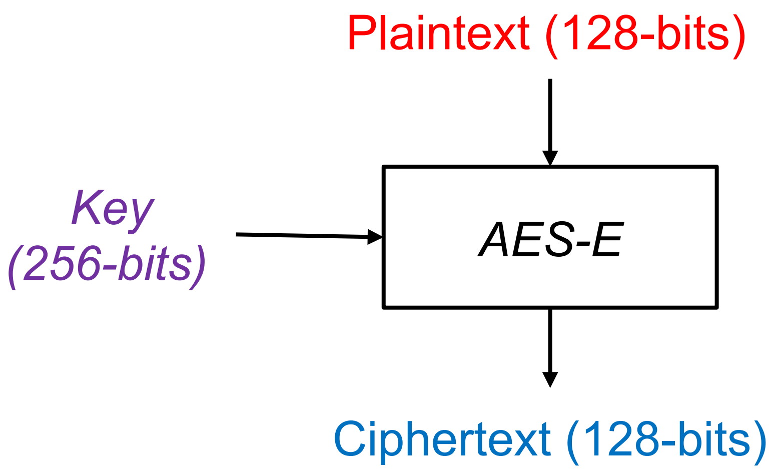
Real-World Block Ciphers

- Data Encryption Standard (DES)
  - 64-bit block size
  - 56-bit key size
  - Released in 1976
  - US government standard until 2001

- Advanced Encryption Standard (AES)
  - 128-bit block size
  - 128, 192, or 256 bit key size
  - Current US government standard
  - Most widely used

Simplified AES Example

Encryption



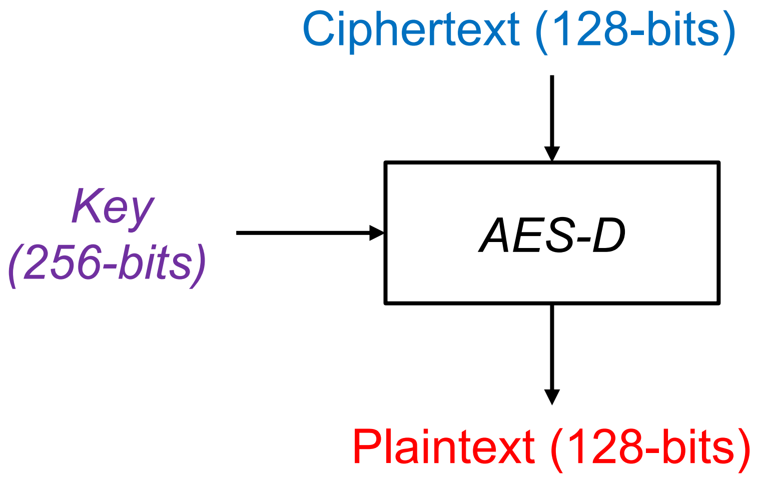
Plaintext (128-bits)

Key (256-bits)

AES-E

Ciphertext (128-bits)

Decryption



Ciphertext (128-bits)

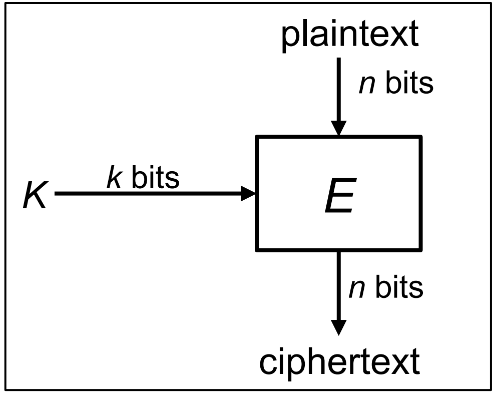
Key (256-bits)

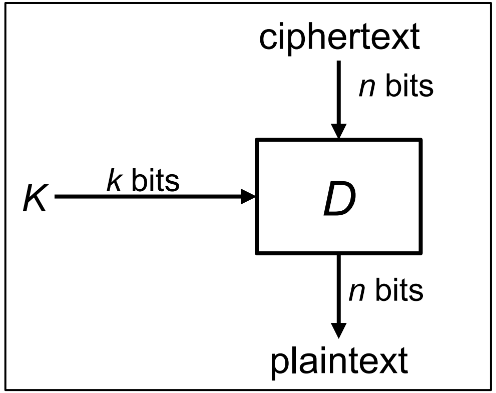
AES-D

Plaintext (128-bits)

Block Ciphers: Definition

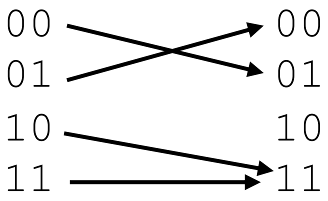
- **Block cipher**: An encryption/decryption algorithm that encrypts a fixed-sized block of bits
- $E_K(M) \rightarrow C$: Encryption
  - Inputs: k-bit key K and an n-bit plaintext M
  - Output: An n-bit ciphertext C
- $D_K(C) \rightarrow M$: Decryption
  - Inputs: a k-bit key, and an n-bit ciphertext C
  - Output: An n-bit plaintext
  - The inverse of the encryption function
- Desired Properties
  - **Correctness**: E_K is a permutation, D_K is its inverse
  - **Efficiency**: Encryption/decryption should be fast
  - **Security**: E behaves like a random permutation



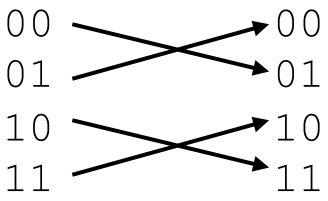


Block Ciphers: Correctness

- When the key *k* is fixed, *EK*(*M*) must be a **permutation** (**bijective function**) on *n*-bit strings
  - Each input must correspond to exactly one unique output
- Intuition
  - Suppose *EK*(*M*) is not bijective
  - Then two inputs might correspond to the same output: *E*(*K*, *x*1) = *E*(*K*, *x*2) = *y*
  - Given ciphertext *y*, you can't uniquely decrypt. *D*(*K*, *y*) = *x*1? *D*(*K*, *y*) = *x*2?



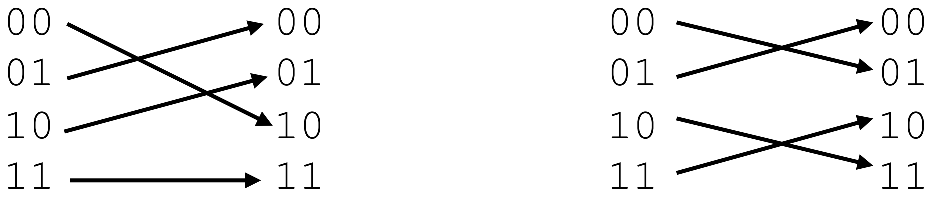
Not bijective: Two inputs encrypt to the same output



Bijective: Each input maps to exactly one unique output

Block Ciphers: Security

- A secure block cipher behaves like a randomly chosen permutation from the set of all permutations on *n*-bit strings
    - A random permutation: Each *n*-bit input is mapped to one randomly-chosen *n*-bit output
- Consider a distinguishing game
    - Eve gets two boxes: One is a randomly chosen permutation, and one is *EK* with a randomly chosen key *K*
    - Eve should not be able to tell which is which with probability > 1/2



One of these is E_K with a randomly chosen *K*, and the other one is a randomly chosen permutation. Eve can't distinguish them.

Block Ciphers: Brute-force attacks?

- How hard is it to run a brute-force attack on a 128-bit key?
  - We have to try 2^{128} possibilities. How big is 2^{128}?
- Handy approximation: $2^{10} \approx 10^3$
  - $2^{128} = 2^{10*12.8} \approx (10^3)^{12.8} \approx (10^3)^{13} = 10^{39}$
- Suppose we have massive hardware that can try 10^9 (1 billion) keys in 1 nanosecond (a billionth of a second). That's 10^{18} keys per second
  - We'll need $10^{39} / 10^{18} = 10^{21}$ seconds. How long is that?
  - One year $\approx 3 \times 10^7$ seconds
  - 10^{21} seconds / $3 \times 10^7 \approx 3 \times 10^{13}$ years ≈ 30 trillion years
- **Takeaway**: Brute-forcing a 128-bit key takes astronomically long. Don't even try.

Block Ciphers: Efficiency

- Encryption and decryption should be computable in microseconds
  - Formally: KeyGen(), Enc(), and Dec(), should not take exponential time
- Block cipher algorithms typically use operations like XOR, bit-shifting, and small table lookups
  - Very fast on modern processors
- Modern CPUs provide dedicated hardware support for block ciphers

AES (Advanced Encryption Standard)

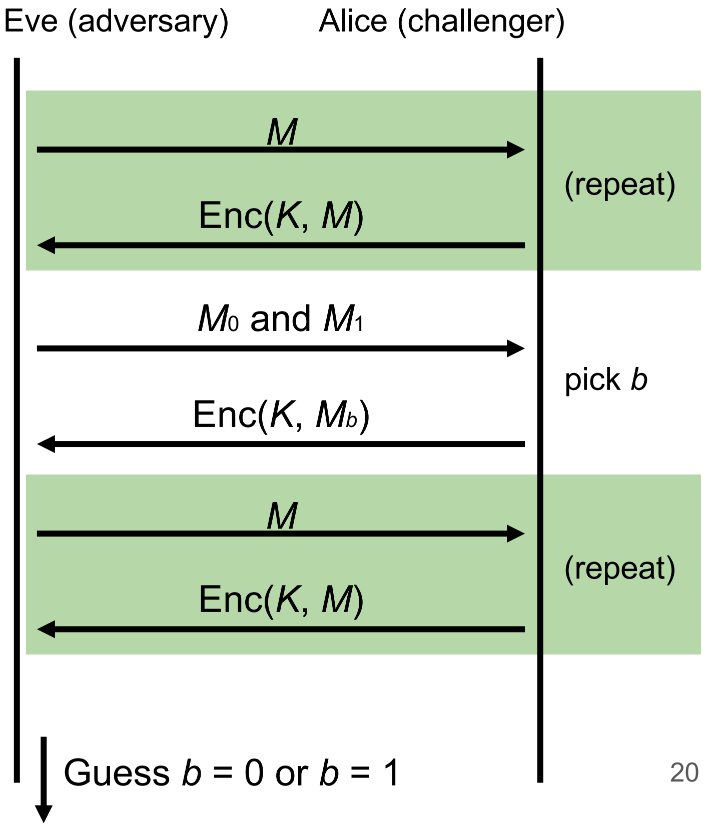
- Key size 128, 192, or 256 bits (*k* = 128, 192, or 256)
    - Actual cipher names are AES-128, AES-192, and AES-256
- Block size 128 bits (*n* = 128)
    - Note: The block size is still always 128 bits, regardless of key size
- You don’t need to know how AES works, but you do need to know its parameters

AES (Advanced Encryption Standard)

- There is no formal proof that AES is secure (indistinguishable from a random permutation)
- However, in 20 years, nobody has been able to break it, so it is *assumed* to be secure
  - The NSA uses AES-256 for secrets they want to keep secure for the 40 years (even in the face of unknown breakthroughs in research)
- **Takeaway**: AES is the modern standard block cipher algorithm
  - The standard key size (128 bits) is large enough to prevent brute-force attacks

Are Block Ciphers IND-CPA Secure?

- Consider the following adversary:
  - Eve sends two different messages M_0 and M_1
  - Eve receives either $E_K(M_0)$ or $E_K(M_1)$
  - Eve requests the encryption of M_0 again
  - Strategy: If the encryption of M_0 matches what she received, guess b = 0. Else, guess b = 1.
- Eve can win the IND-CPA game with probability 1!
  - Block ciphers are not IND-CPA secure



Issues with Block Ciphers

- Block ciphers are not IND-CPA secure, because they're deterministic
  - A scheme is **deterministic** if the same input always produces the same output
  - No deterministic scheme can be IND-CPA secure because the adversary can always tell if the same message was encrypted twice
- Block ciphers can only encrypt messages of a fixed size
  - For example, AES can only encrypt-decrypt 128-bit messages
  - What if we want to encrypt something longer than 128 bits?
- To address these problems, we'll add **modes of operation** that use block ciphers as a building block!

Summary: Block Ciphers

- Encryption: input a *k*-bit key and *n*-bit plaintext, receive *n*-bit ciphertext
- Decryption: input a *k*-bit key and *n*-bit ciphertext, receive *n*-bit plaintext
- Correctness: when the key is fixed, $E_K(M)$ should be bijective
- Security
  - Without the key, E_K(m) is computationally indistinguishable from a random permutation
  - Brute-force attacks take astronomically long and are not possible
- Efficiency: algorithms use XORs and bit-shifting (very fast)
- Implementation: AES is the modern standard
- Issues
  - Not IND-CPA secure because they're deterministic
  - Can only encrypt *n*-bit messages

Block Cipher Modes of Operation

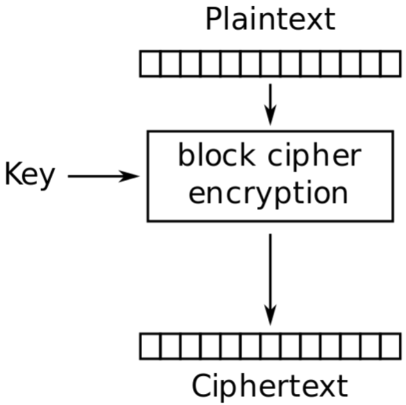
Block Ciphers Operating Mode

- What to do with large messages?
  - We need to break up the data into blocks and encrypt those
  - The way we encrypt the blocks impacts security
  - The ways of encrypting blocks are called *operating modes*

Scratchpad: Let’s design it together

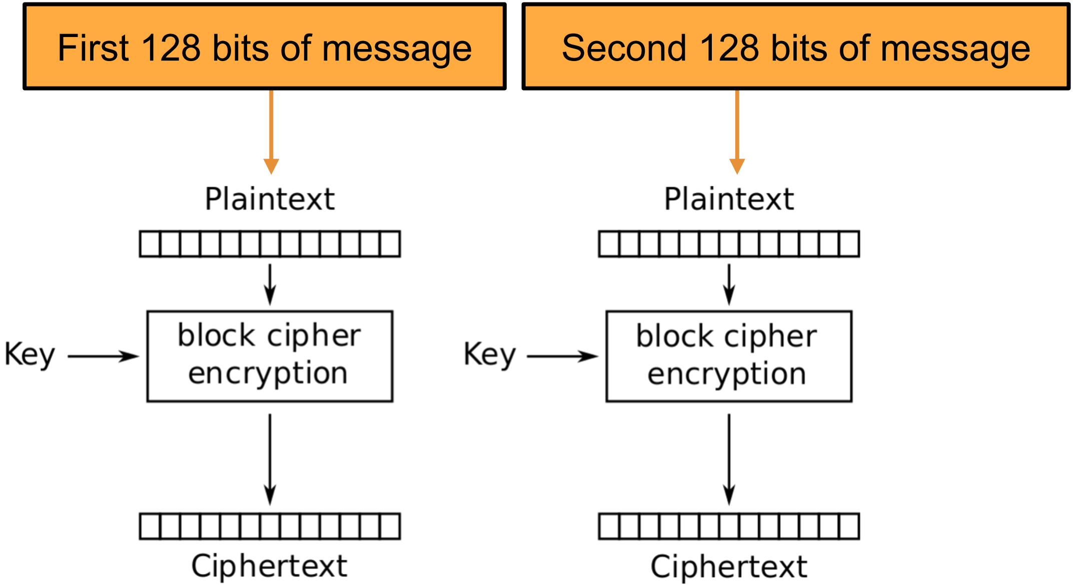
Here’s an AES block. Remember that it can only encrypt 128-bit messages.

How can we use AES to encrypt a longer message (say, 256 bits?)



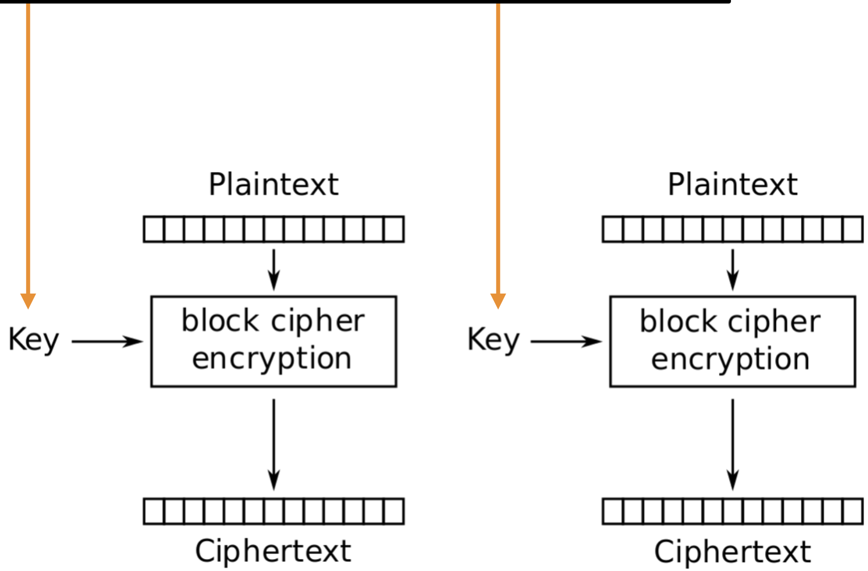
Scratchpad: Let’s design it together

Idea: Let’s use AES twice!



Scratchpad: Let’s design it together

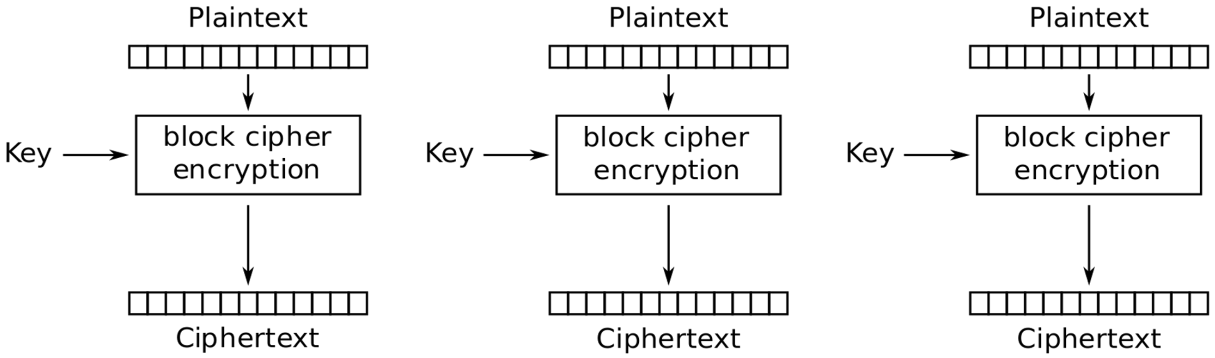
Note that we are using the same key twice. We want to avoid a situation like one-time pads where we need very long keys.



ECB Mode

ITIS 6200 / 8200

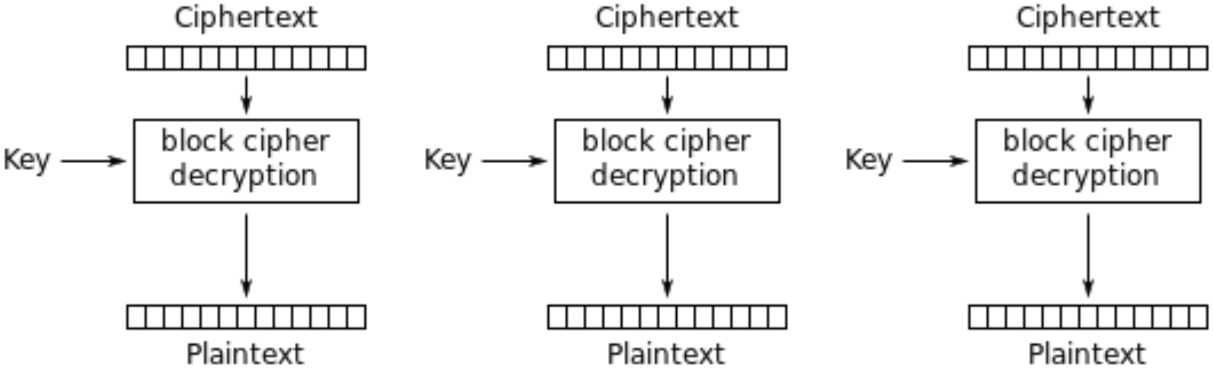
- **Electronic code book (ECB) mode**
  - Enc(K, M) = C_1 || C_2 || … || C_j
  - Assume j is the number of blocks of plaintext in M, each of size n



Electronic Codebook (ECB) mode encryption

ECB Mode

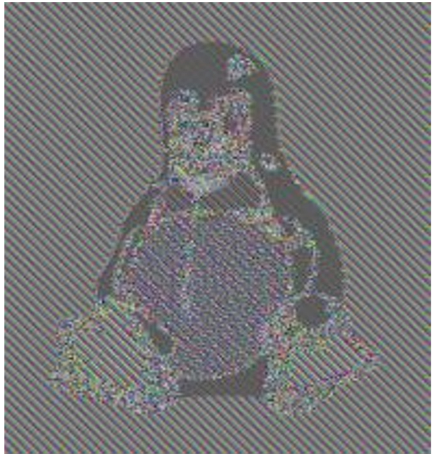
- AES-ECB is not IND-CPA secure. Why?
  - Because ECB is deterministic



Electronic Codebook (ECB) mode decryption

ECB Mode: Problems

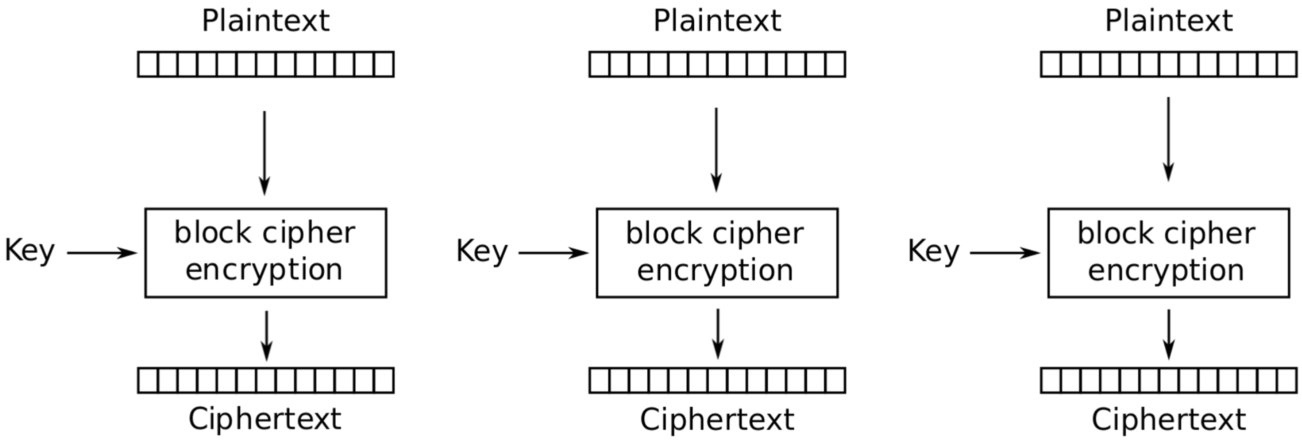
- Deterministic: the same plaintext blocks produce the same ciphertext blocks
  - Many computer files have duplicate blocks, and we don't want an attacker to be able to tell this



Scratchpad: Let’s design it together

Here’s ECB mode. It’s not IND-CPA secure because it’s deterministic.

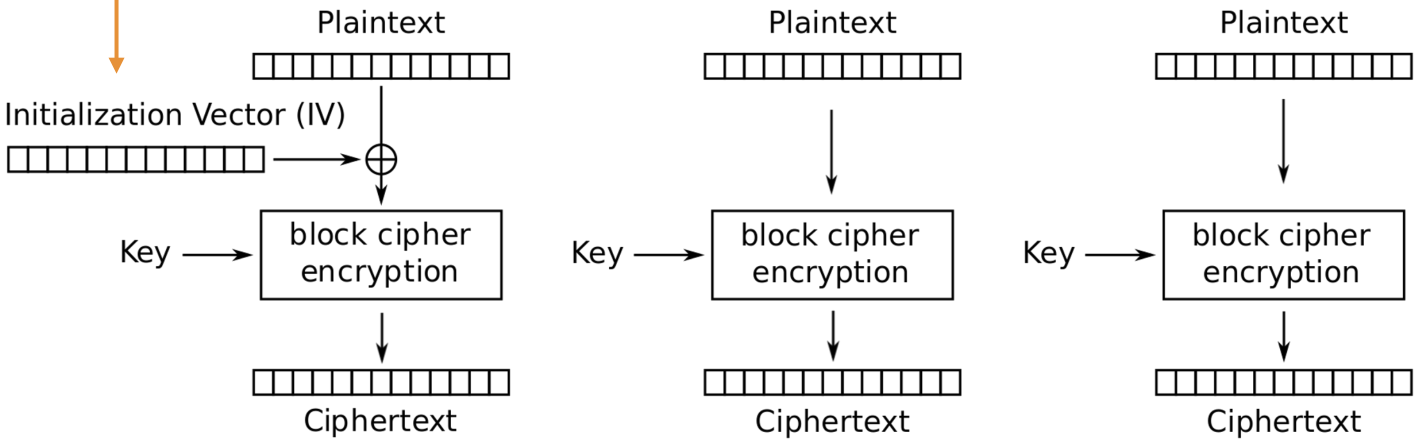
Let’s fix that by adding some randomness.



Scratchpad: Let’s design it together

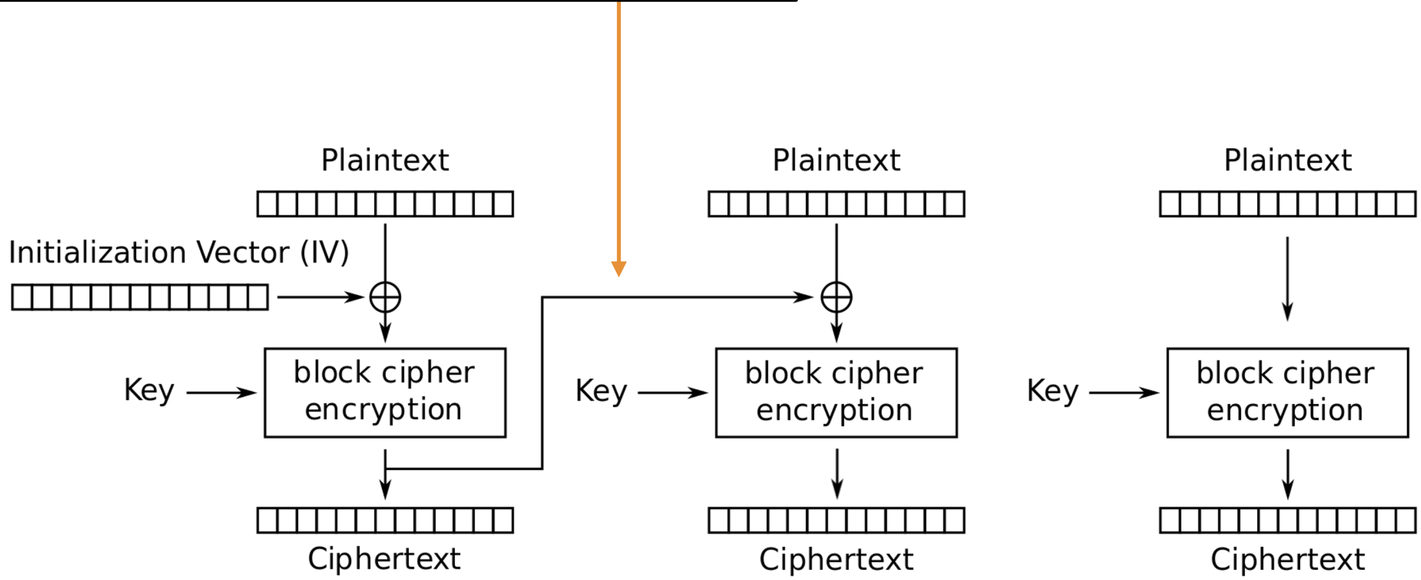
The **Initialization Vector** (**IV**) is different for every encryption. Now the first ciphertext block will be different for every encryption!

Okay, but the other blocks are still deterministic...



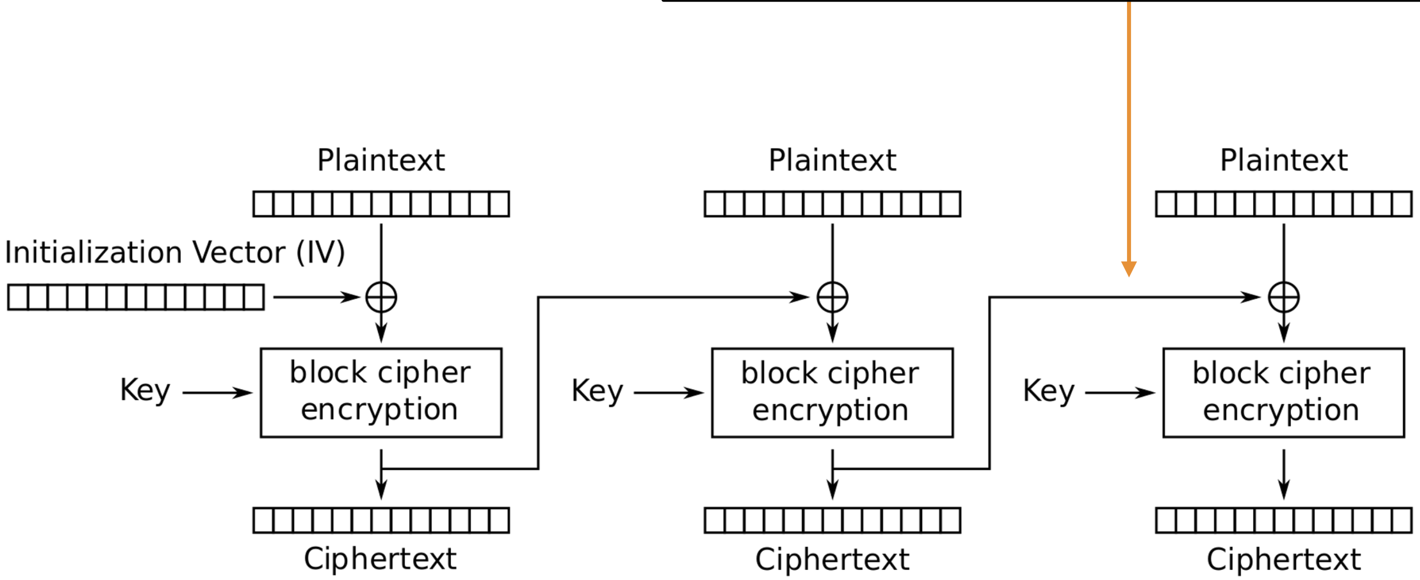
Scratchpad: Let’s design it together

Idea: The first ciphertext block was computed with some randomness. Let’s use it to add randomness to the second block.



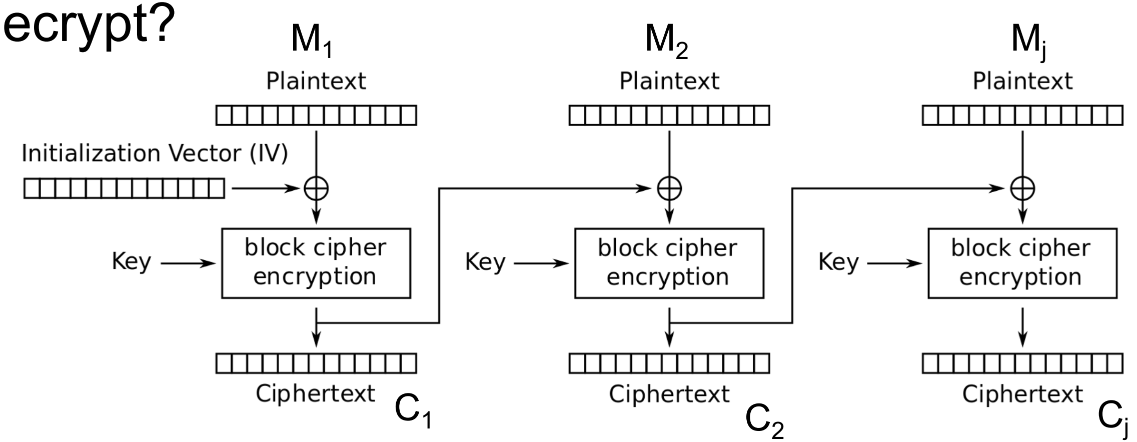
Scratchpad: Let’s design it together

Now the second ciphertext block has some randomness in it. Let’s use it to add randomness to the third block.



CBC Mode

- This is called **cipher block chaining (CBC) mode**
- $C_i = E_K(\mathbf{M_i} \oplus \mathbf{C_{i-1}})$; $C_0 = IV$
- Enc(K, M):
  - Split M into j plaintext blocks $M_1 \ldots M_j$ each of size n
  - Choose a random IV
  - Compute and output $(IV, C_1, \ldots, C_j)$ as the overall ciphertext
- How do we decrypt?

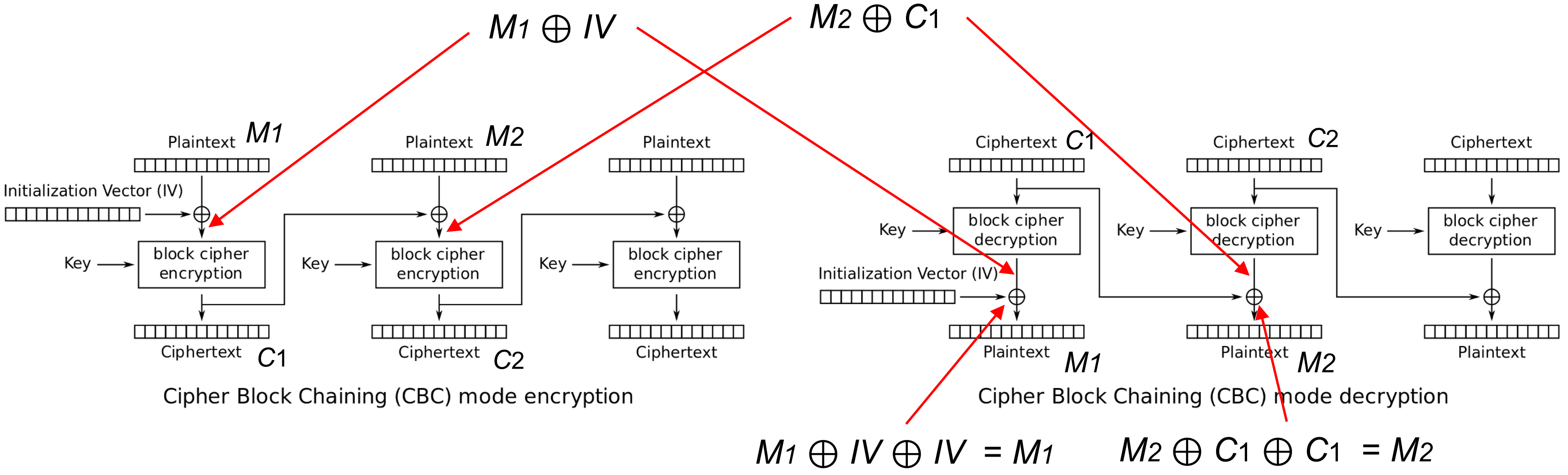


Cipher Block Chaining (CBC) mode encryption

CBC Mode: Decryption

ITIS 6200 / 8200

- How do we decrypt CBC mode?
  - Parse ciphertext as (IV, C_1, …, C_j)
  - Decrypt each ciphertext and then XOR with IV or previous ciphertext

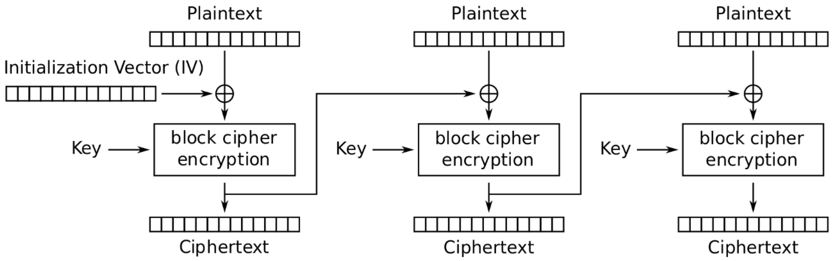


CBC Mode: Decryption

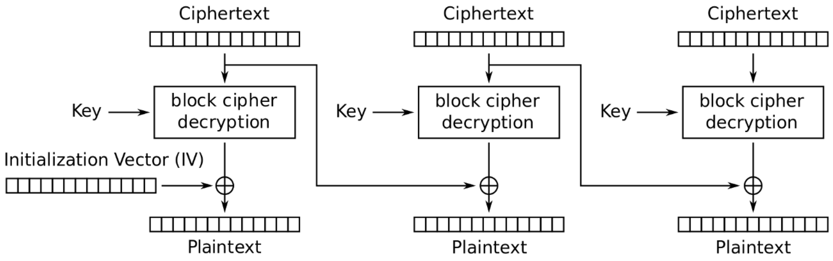
$C_i = E_K(M_i \oplus C_{i-1})$	Definition of encryption
$D_K(C_i) = D_K(E_K(M_i \oplus C_{i-1}))$	Decrypting both sides
$D_K(C_i) = M_i \oplus C_{i-1}$	Decryption and encryption cancel
$D_K(C_i) \oplus C_{i-1} = M_i \oplus C_{i-1} \oplus C_{i-1}$	XOR both sides with C_{i-1}
$D_K(C_i) \oplus C_{i-1} = M_i$	XOR property

CBC Mode: Efficiency & Parallelism

- Can encryption be parallelized?
  - No, we have to wait for block i to finish before encrypting block i+1
- Can decryption be parallelized?
  - Yes, decryption only requires ciphertext as input



Cipher Block Chaining (CBC) mode encryption

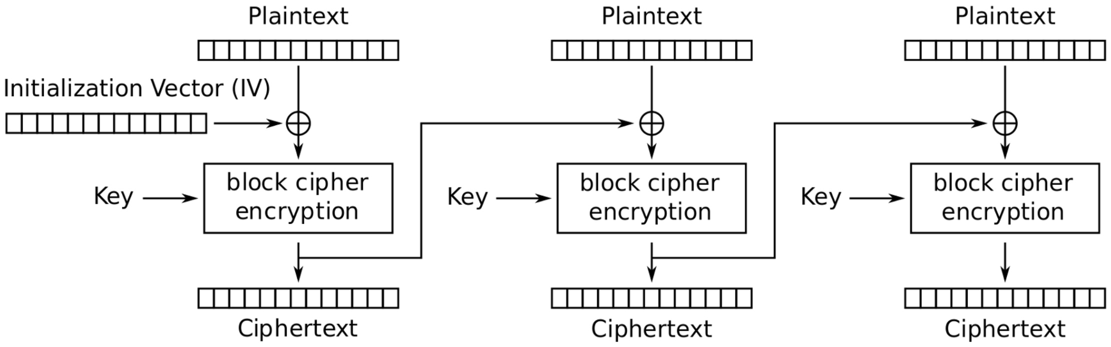


Cipher Block Chaining (CBC) mode decryption

CBC Mode: Padding

ITIS 6200 / 8200

- What if you want to encrypt a message that isn't a multiple of the block size?
  - AES-CBC is only defined if the plaintext length is a multiple of the block size
- Solution: Pad the message until it's a multiple of the block size
  - **Padding**: Adding dummy bytes at the end of the message until it's the proper length



Cipher Block Chaining (CBC) mode encryption

CBC Mode: Padding

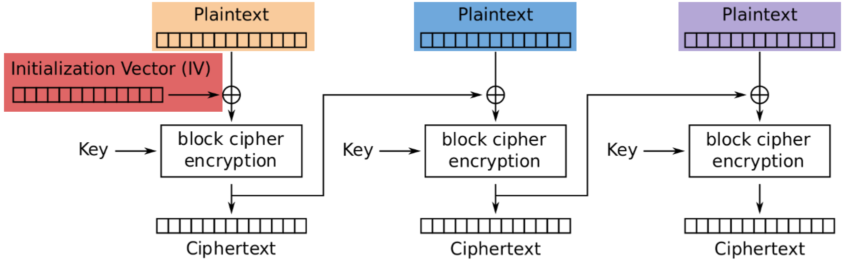
- What padding scheme should we use?
  - Padding with 0’s?
    - Doesn’t work: What if our message already ends with 0’s?
  - Padding with 1’s?
    - Same problem
- We need a scheme that can be unpadded without ambiguity
  - One scheme that works: Append a 1, then pad with 0’s
    - If plaintext is multiple of n, you still need to pad with an entire block
  - Another scheme: Pad with the number of padding bytes
    - So if you need 1 byte, pad with **01**; if you need 3 bytes, pad with **03 03 03**
    - If you need 0 padding bytes, pad an entire dummy block

CBC Mode: Security

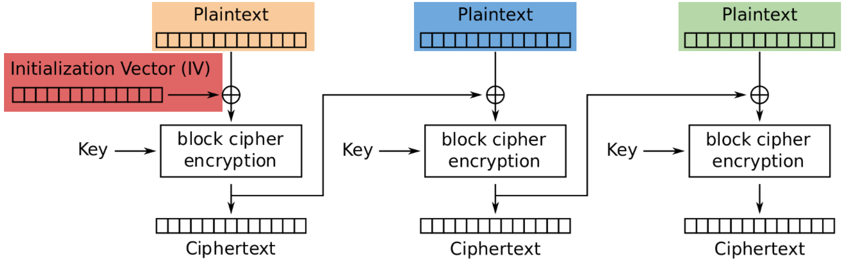
- AES-CBC is IND-CPA secure. With what assumption?
    - The IV must be randomly generated and never reused
- What happens if you reuse the IV?
    - The scheme becomes deterministic: No more IND-CPA security

CBC Mode: IV Reuse

- Consider two three-block messages: $P_1P_2P_3$ and $P_1P_2P_4$
  - The first two blocks are the same for both messages, but the last block is different
  - What if we encrypt them with the same IV?
- When the IV is reused, CBC mode reveals when two messages start with the same plaintext blocks, up to the first different plaintext block



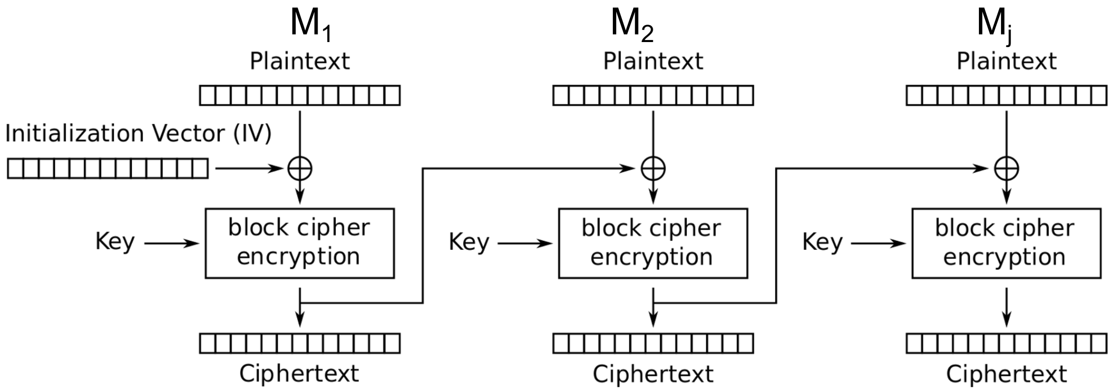
Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode encryption

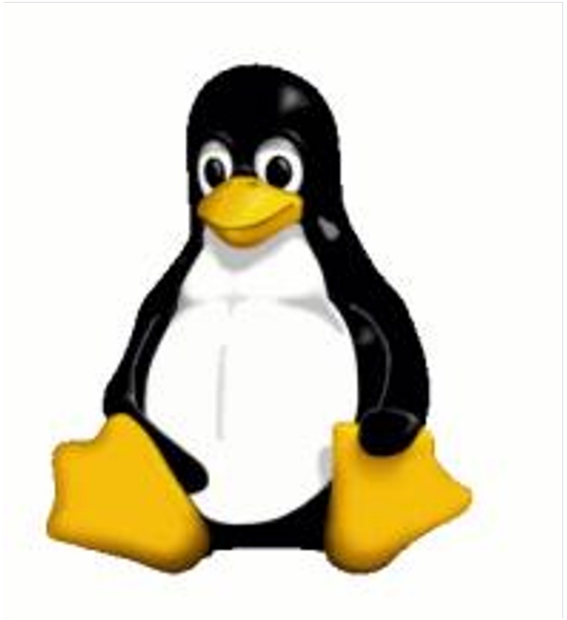
CBC Mode is IND-CPA (when used correctly)

- Enc(*K*, *M*):
  - Split M into j plaintext blocks M_1 … M_j each of size *n*
  - Choose random IV, compute and output (IV, C_1, …, C_j) as the overall ciphertext
- Why IND-CPA?
  - If there exists an attacker that wins in the IND-CPA game, then there exists an attacker that breaks the block cipher security. Proof is out of scope.

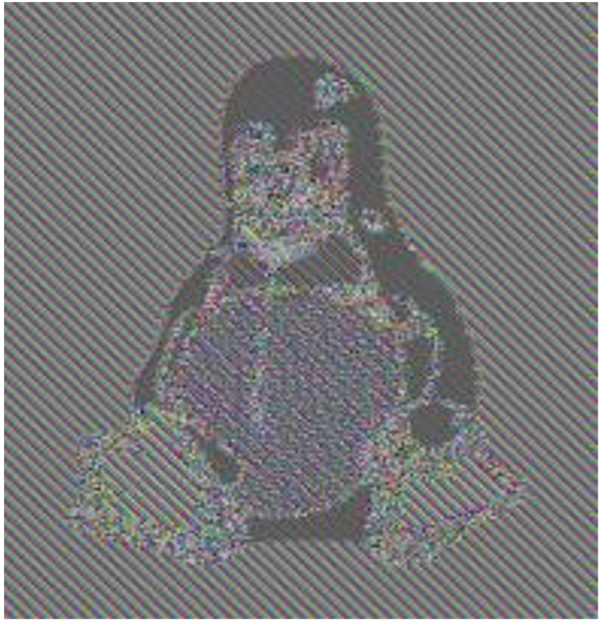


Cipher Block Chaining (CBC) mode encryption

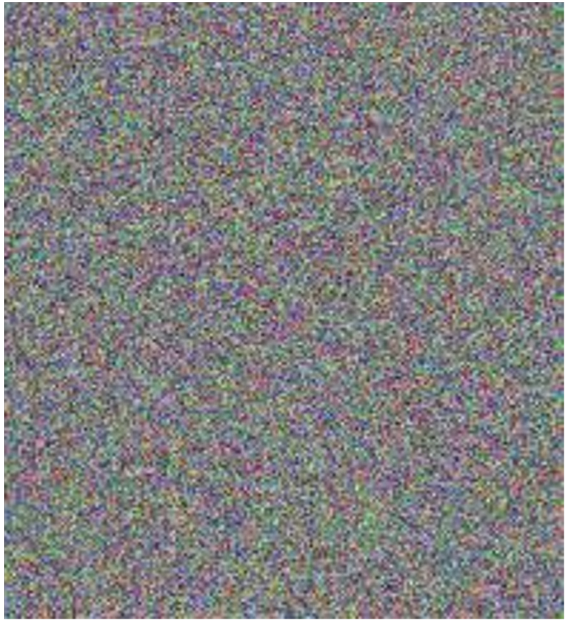
CBC Mode: Penguin



Original image



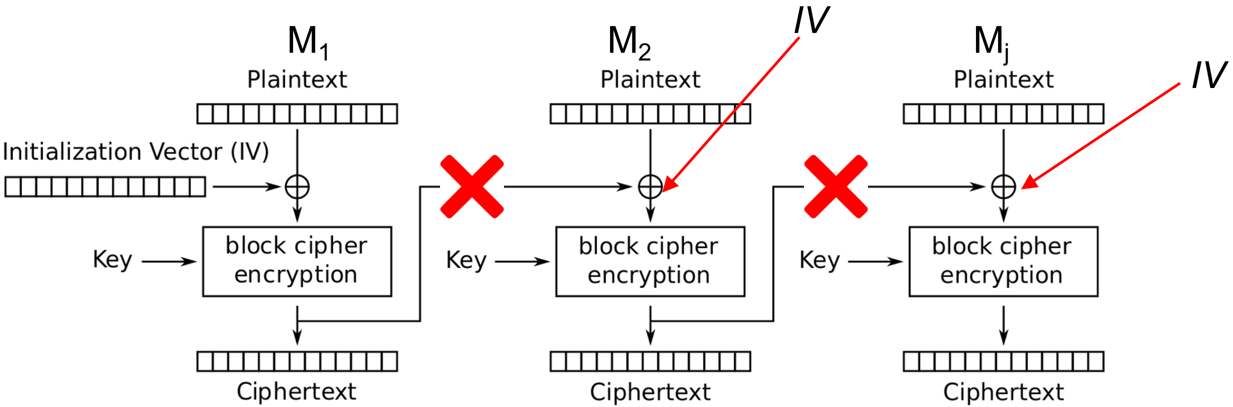
Encrypted with ECB



Encrypted with CBC
with random IVs

CBC Mode (Design Question)

- Q: Why do we need chain? Why not just use IV for every block?
  - What if multiple blocks have the same message?
  - Remember frequency attack in substitution cypher?



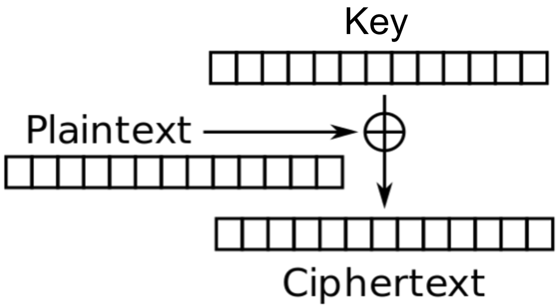
Cipher Block Chaining (CBC) mode encryption

CBC Mode: Problems

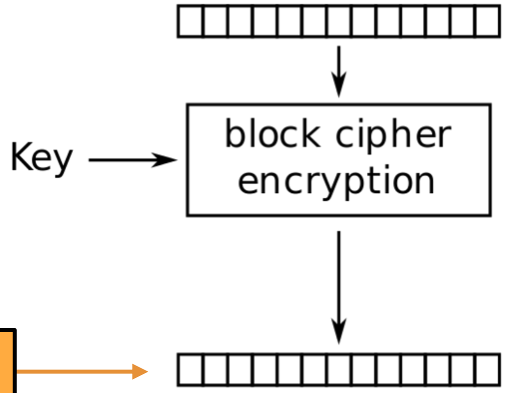
- If I want to change the plaintext of one block, I must re-encrypt every following block
  - It’s a chain, remember?
- For some cases, this is bad
  - Encrypted file systems, for example

CTR Mode Scratchpad: Let’s design it together

One-time pads are secure if we never reuse the key.



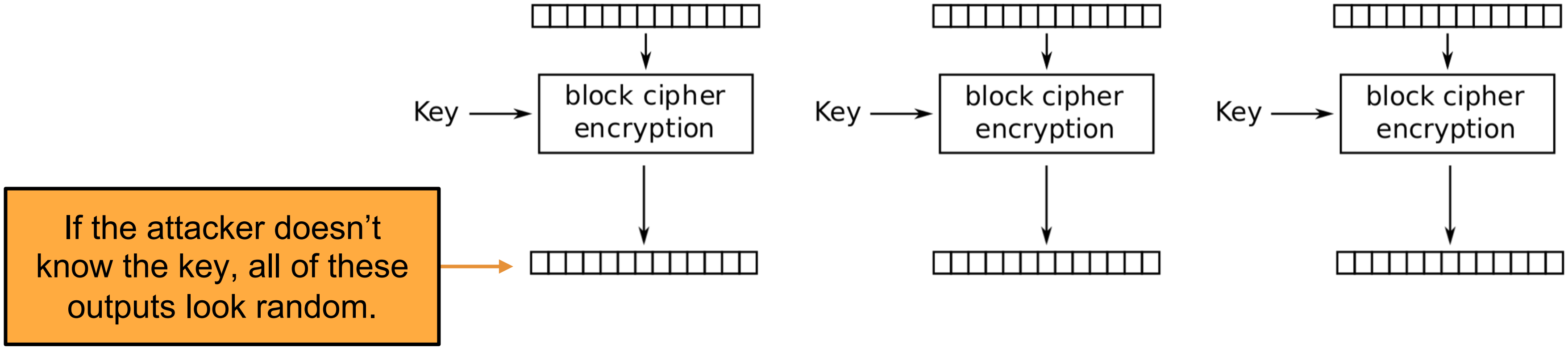
CTR Mode Scratchpad: Let’s design it together



If the attacker doesn’t know the key, the block cipher output looks random.

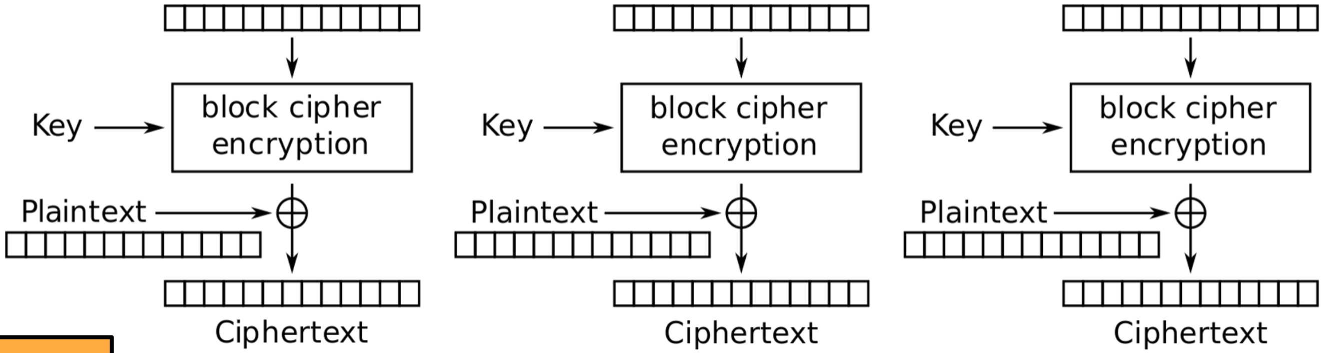
New idea: Can we use block ciphers to simulate a one-time pad?

CTR Mode Scratchpad: Let’s design it together



If the attacker doesn’t know the key, all of these outputs look random.

CTR Mode Scratchpad: Let's design it together

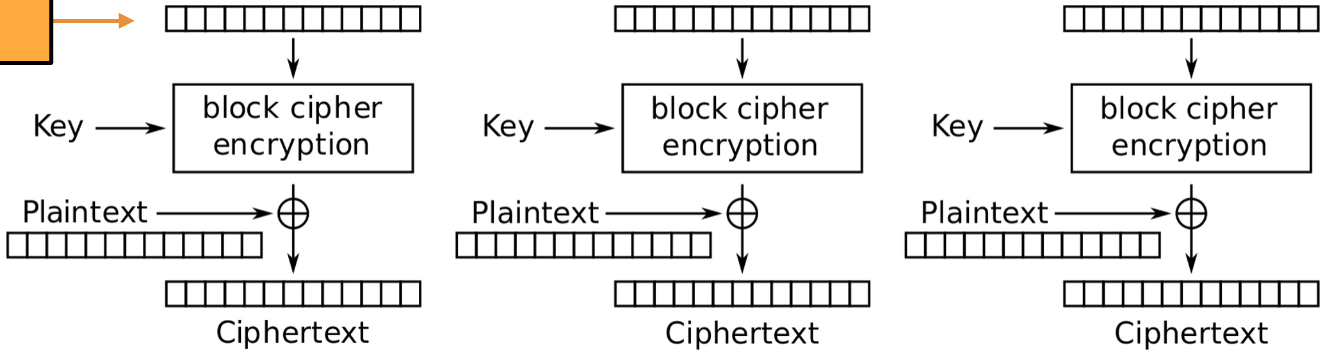


Idea: Use this random-looking output as a one-time pad!

Remember one-time pads: XOR the pad with plaintext to get ciphertext

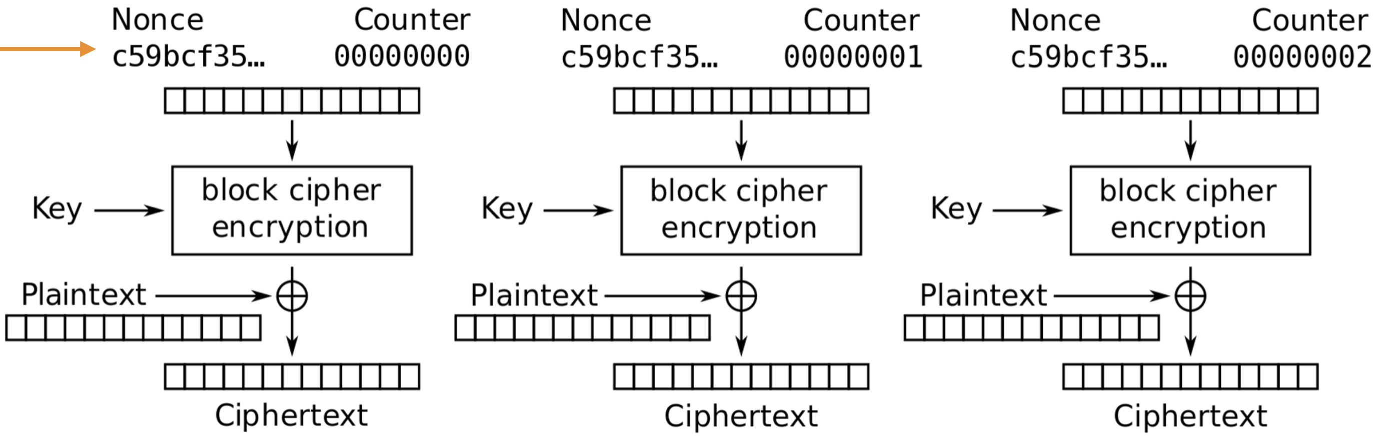
CTR Mode Scratchpad: Let’s design it together

What do we use as input to the block cipher?



CTR Mode Scratchpad: Let’s design it together

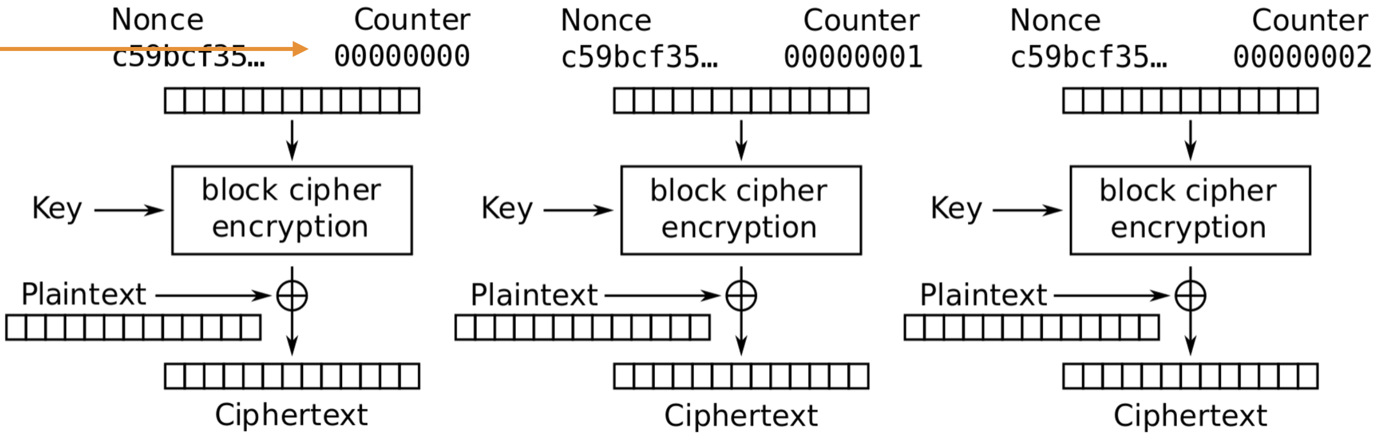
IND-CPA schemes need randomness, so let’s put a random **nonce** here!



CTR Mode Scratchpad: Let’s design it together

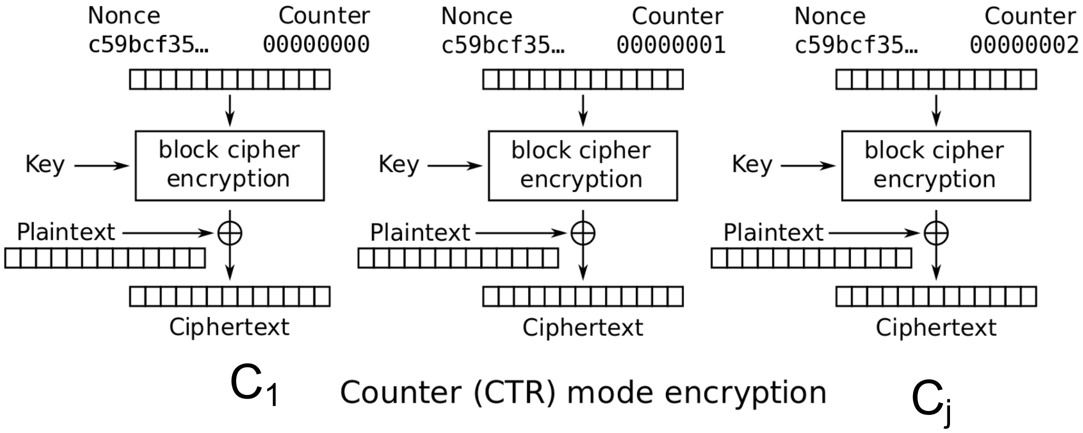
ITIS 6200 / 8200

The **counter** increments per block to ensure each block cipher output is different.



CTR (Counter) Mode

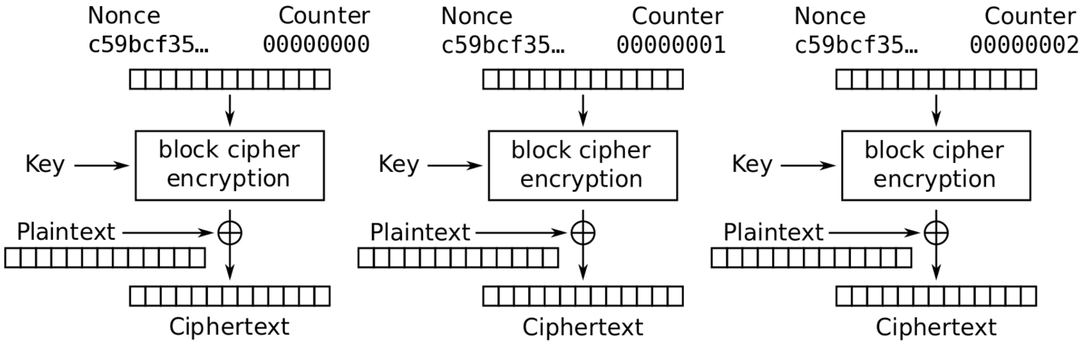
- Note: the random value is named the nonce here, but the idea is the same as the IV in CBC mode
- Overall ciphertext is (Nonce, C_1, …, C_j)



Counter (CTR) mode encryption

CTR Mode

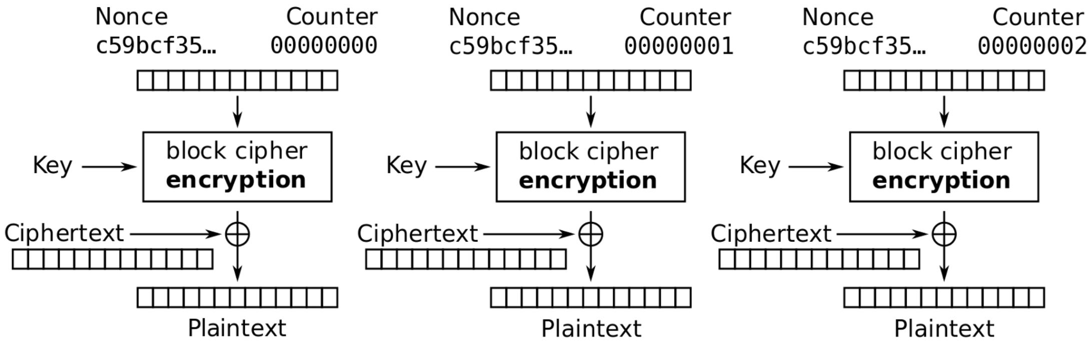
- Enc(K, M):
  - Split M in plaintext blocks $M_1 \ldots M_j$ (each of block size n)
  - Choose random nonce
  - Compute and output (Nonce, C_1, …, C_j)
- How do you decrypt?



C_1 Counter (CTR) mode encryption C_j

CTR Mode: Decryption

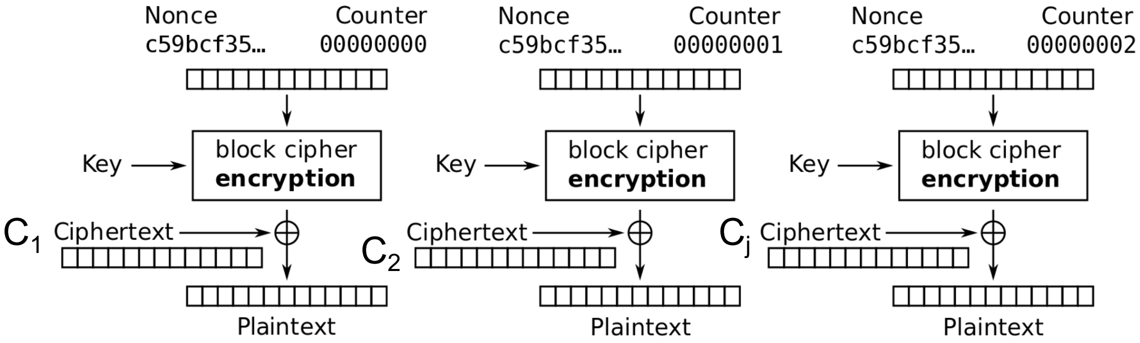
- Recall one-time pad: XOR with ciphertext to get plaintext
- Note: we are only using block cipher **encryption**, not decryption



Counter (CTR) mode decryption

CTR Mode: Decryption

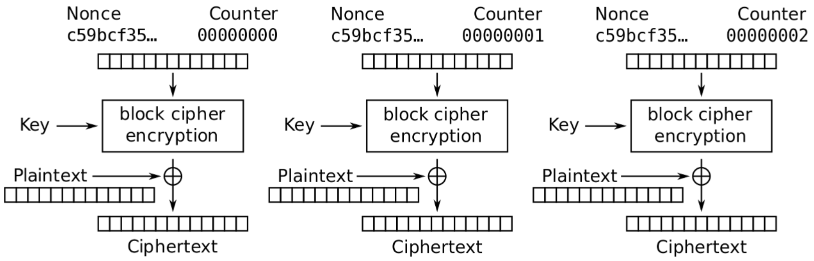
- Dec(K, C):
  - Parse C into (nonce, C_1, …, C_j)
  - Compute M_i by XORing Ci with output of E_k on nonce and counter
  - Concatenate resulting plaintexts and output M = M_1 … M_j



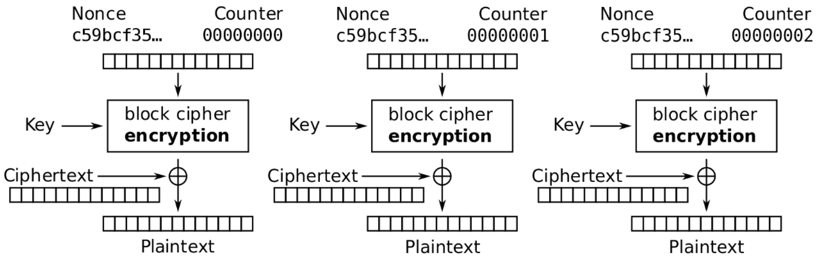
Counter (CTR) mode decryption

CTR Mode: Efficiency

- Can encryption be parallelized?
    - Yes
- Can decryption be parallelized?
    - Yes



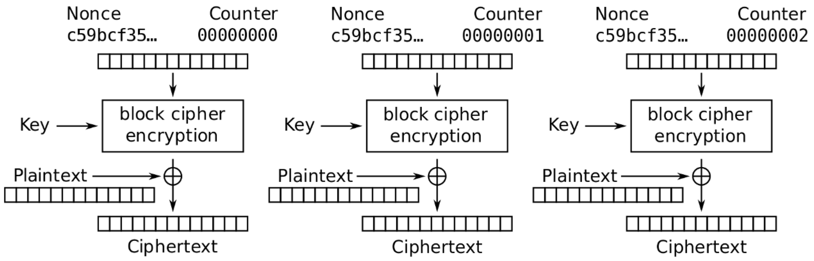
Counter (CTR) mode encryption



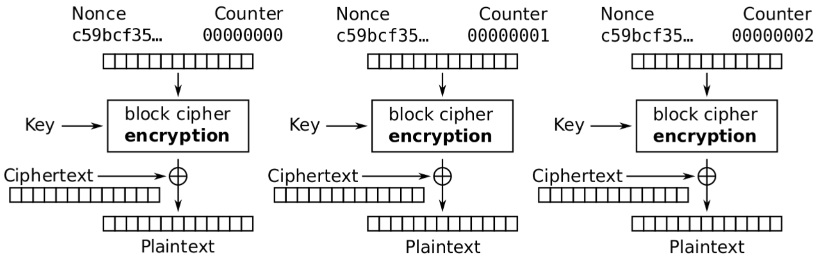
Counter (CTR) mode decryption

CTR Mode: Padding

- Do we need to pad messages?
  - No! We can just cut off the parts of the XOR that are longer than the message.



Counter (CTR) mode encryption

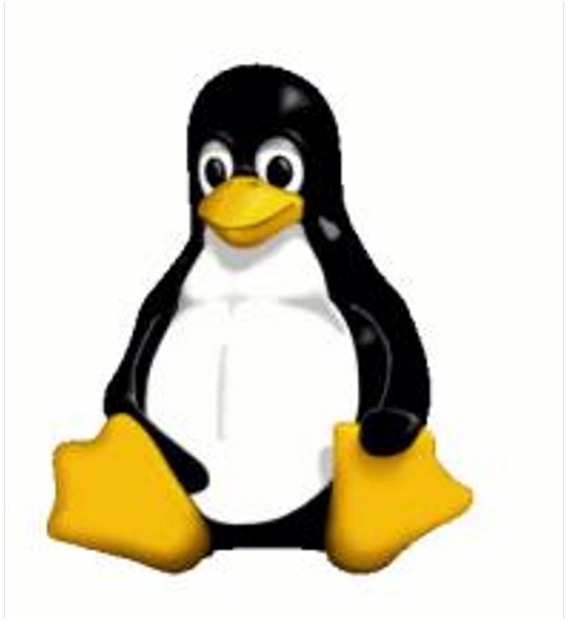


Counter (CTR) mode decryption

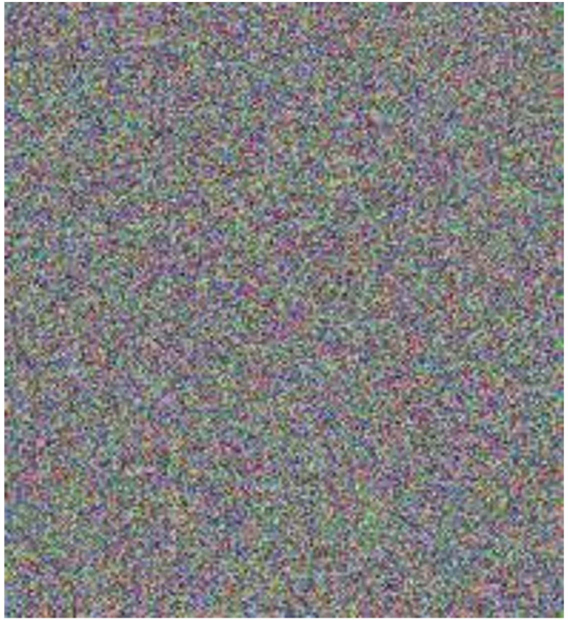
CTR Mode: Security

- AES-CTR is IND-CPA secure. With what assumption?
- The nonce must be randomly generated and never reused
- What happens if you reuse the nonce?
- Equivalent to reusing a key in a one-time pad
  - Recall: Key reuse in a one-time pad is catastrophic: usually leaks enough information for an attacker to deduce the entire plaintext

CTR Mode: Penguin



Original image



Encrypted with CTR, with random nonces

IVs and Nonces

- **Initialization vector** (**IV**): A random, but public, one-use value to introduce randomness into the algorithm
    - For CTR mode, we say that you use a **nonce** (number used once), since the value has to be unique, not necessarily random.
    - In this class, we use IV and nonce interchangeably
- **Never reuse IVs**
    - In some algorithms, IV/nonce reuse leaks limited information (e.g. CBC)
    - In some algorithms, IV/nonce reuse leads to catastrophic failure (e.g. CTR)

IVs and Nonces

- Thinking about the consequences of IV/nonce reuse is hard

- What if the IV/nonce is not reused, but the attacker can predict future values?

- Solution: Randomly generate a new IV/nonce for every encryption
    - If the nonce is 128 bits or longer, the probability of generating the same IV/nonce twice is astronomically small (basically 0)
    - Now you don’t ever have to think about IV/nonce reuse attacks!

Comparing Modes of Operation

- If you need high performance, which mode is better?
    - CTR mode, because you can parallelize both encryption and decryption
- If you're paranoid about security, which mode is better?
    - CBC mode is better
- Theoretically, CBC and CTR mode are equally secure if used properly
    - However, if used improperly (IV/nonce reuse), CBC only leaks partial information, and CTR fails catastrophically
        - Consider human factors: Systems should be as secure as possible even when implemented *incorrectly*
    - IV failures on CTR mode have resulted in multiple real-world security incidents!

Lack of Integrity and Authenticity

- Block ciphers are designed for *confidentiality* (IND-CPA)

- If an attacker tampers with the ciphertext, we are not guaranteed to detect it

- Remember Mallory: An *active* manipulator who wants to tamper with the message

Lack of Integrity and Authenticity

- Consider CTR mode
- What if Mallory tampers with the ciphertext using XOR?

	P	a	y		M	a	l		$	1	0	0
M	0x50	0x61	0x79	0x20	0x4d	0x61	0x6c	0x20	0x24	0x31	0x30	0x30

$$\oplus$$

$E_K(i)$	0x8a	0xe3	0x5e	0xcf	0x3b	0x40	0x46	0x57	0xb8	0x69	0xd2	0x96

=

C	0xda	0x82	0x27	0xef	0x76	0x21	0x2a	0x77	0x9c	0x58	0xe2	0xa6

Lack of Integrity and Authenticity

- Suppose Mallory knows the message *M*
- How can Mallory change the *M* to say `Pay Mal $900`?

	P	a	y		M	a	l		$	**1**	0	0
M	0x50	0x61	0x79	0x20	0x4d	0x61	0x6c	0x20	0x24	**0x31**	0x30	0x30

$$\oplus$$

$E_K(i)$	0x8a	0xe3	0x5e	0xcf	0x3b	0x40	0x46	0x57	0xb8	**0x69**	0xd2	0x96

=

C	0xda	0x82	0x27	0xef	0x76	0x21	0x2a	0x77	0x9c	**0x58**	0xe2	0xa6

Lack of Integrity and Authenticity

$C_i = M_i \oplus \text{Pad}_i$	$\texttt{0x58} = \texttt{0x31} \oplus \text{Pad}_i$	Definition of CTR
$\text{Pad}_i = M_i \oplus C_i$	$\text{Pad}_i = \texttt{0x58} \oplus \texttt{0x31}$ $= \texttt{0x69}$	Solve for the *i*th byte of the pad
$C'_i = M'_i \oplus \text{Pad}_i$	$C'_i = \texttt{0x39} \oplus \texttt{0x69}$ $= \texttt{0x50}$	Compute the changed *i*th byte

C	0xda	0x82	0x27	0xef	0x76	0x21	0x2a	0x77	0x9c	**0x58**	0xe2	0xa6
C'	0xda	0x82	0x27	0xef	0x76	0x21	0x2a	0x77	0x9c	**0x50**	0xe2	0xa6

Lack of Integrity and Authenticity

- What happens when we decrypt *C*'?
  - The message looks like “Pay Mal $900” now!
  - Note: Mallory didn’t have to know the key; no integrity or authenticity for CTR mode!

C'	0xda	0x82	0x27	0xef	0x76	0x21	0x2a	0x77	0x9c	**0x50**	0xe2	0xa6

$$\oplus$$

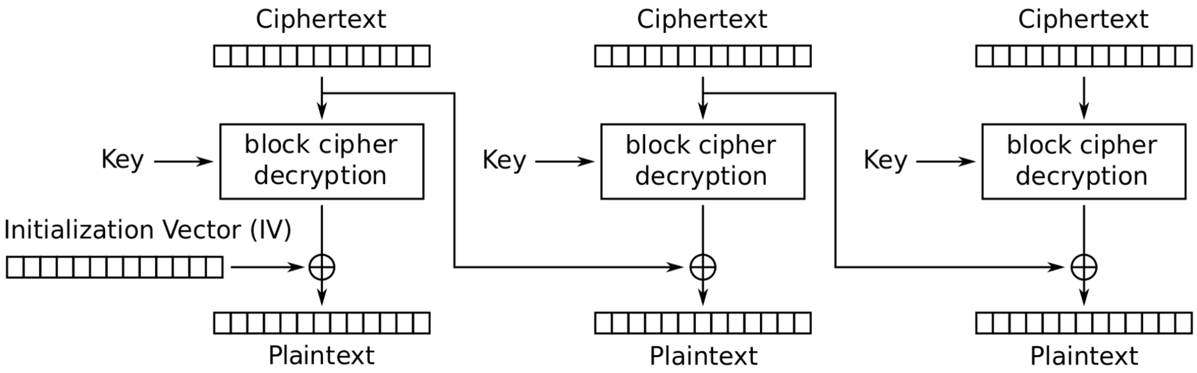
$E_K(i)$	0x8a	0xe3	0x5e	0xcf	0x3b	0x40	0x46	0x57	0xb8	**0x69**	0xd2	0x96

=

M'	0x50	0x61	0x79	0x20	0x4d	0x61	0x6c	0x20	0x24	**0x39**	0x30	0x30
	P	a	y		M	a	l		$	**9**	0	0

Lack of Integrity and Authenticity

- What about CBC?
  - Altering a bit of the ciphertext causes some blocks to become random gibberish
  - However, Bob cannot prove that Alice did not send random gibberish, so it still does *not* provide integrity or authenticity



Cipher Block Chaining (CBC) mode decryption

Block Cipher Modes of Operation: Summary

- ECB mode: Deterministic, so not IND-CPA secure
- CBC mode
  - IND-CPA secure, assuming no IV reuse
  - Encryption is not parallelizable
  - Decryption is parallelizable
  - Must pad plaintext to a multiple of the block size
  - IV reuse leads to leaking the existence of identical blocks at the start of the message
- CTR mode
  - IND-CPA secure, assuming no IV reuse
  - Encryption and decryption are parallelizable
  - Plaintext does not need to be padded
  - Nonce reuse leads to losing all security
- Lack of integrity and authenticity