

CMSC414 Computer and Network Security

Block Cipher Chaining Modes (cont'd) & Cryptographic Hashes

Yizheng Chen | University of Maryland
surrealyz.github.io

Mar 24, 2026

Credits: original slides from instructors and staff from CS161 at UC Berkeley. Blue slides will not be tested.

Announcement

- Project 3: git pull
- Project 3 deadline extended, due on Monday, April 6

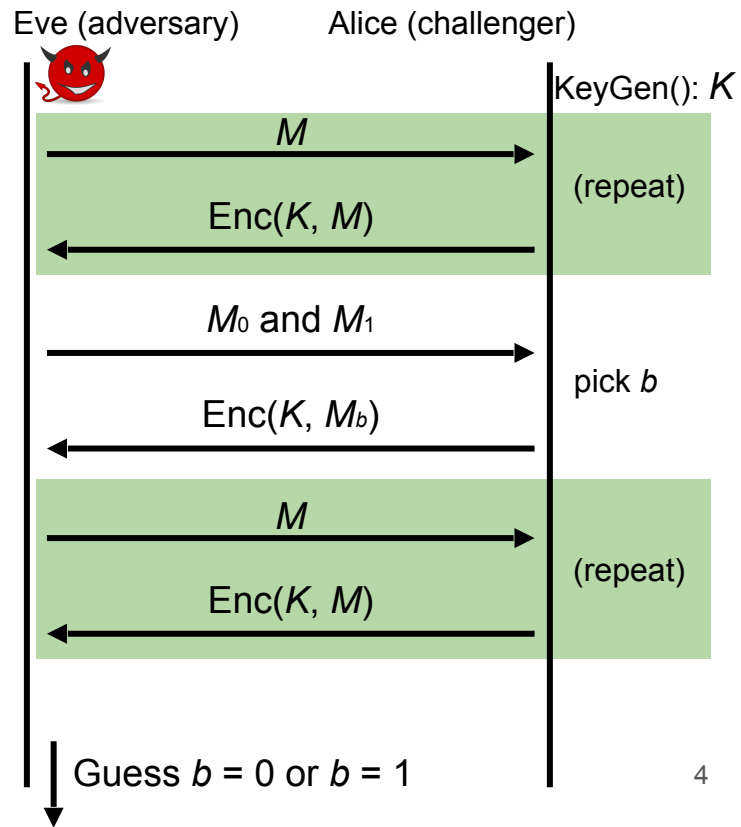
Agenda

- Block Cipher Chaining Modes
- Cryptographic Hashes

IND-CPA: Putting it together

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
 - Alice does not tell Eve which one was encrypted!
4. Eve may again choose plaintexts to send to Alice and receives their ciphertexts
5. Eventually, Eve outputs a guess as to whether Alice encrypted M_0 or M_1

- An encryption scheme is IND-CPA secure if for all polynomial time attackers Eve:
 - Eve can win with probability $\leq 1/2 + \epsilon$, where ϵ is negligible.



Last Time: 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

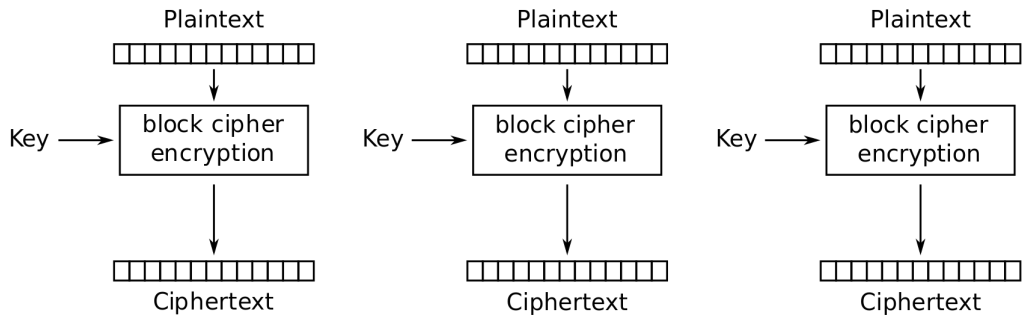
Recall: ECB Mode

- **Electronic Code Book (ECB) mode**

- $\text{Enc}(K, M) = C_1 \parallel C_2 \parallel \dots \parallel C_m$
- Assume m is the number of blocks of plaintext in M , each of size n

- **AES-ECB is not IND-CPA secure. Why?**

- Because ECB is deterministic



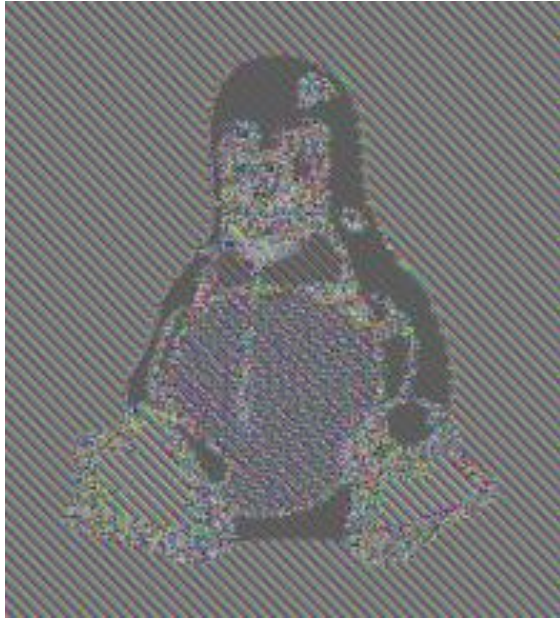
Electronic Codebook (ECB) mode encryption

ECB Mode: Penguin



Original image

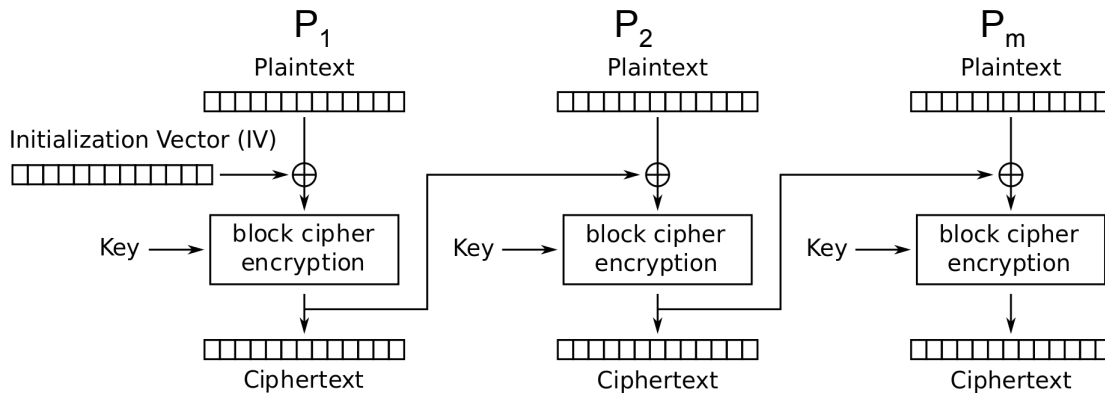
ECB Mode: Penguin



Encrypted with ECB

Recall: CBC Mode

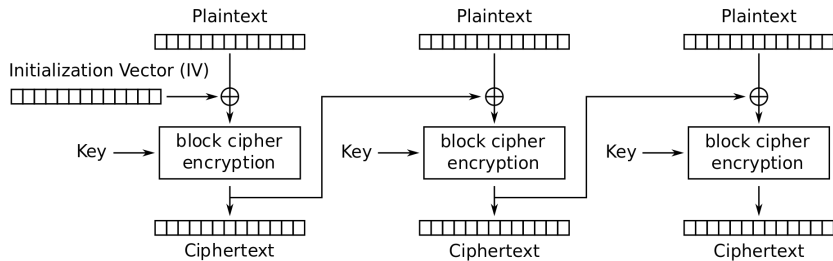
- Cipher Block Chaining (CBC) mode
- $C_i = E_K(M_i \oplus C_{i-1}); C_0 = IV$
- $Enc(K, M)$:
 - Split M in m plaintext blocks $P_1 \dots P_m$ each of size n
 - Choose a random IV
 - Compute and output (IV, C_1, \dots, C_m) as the overall ciphertext



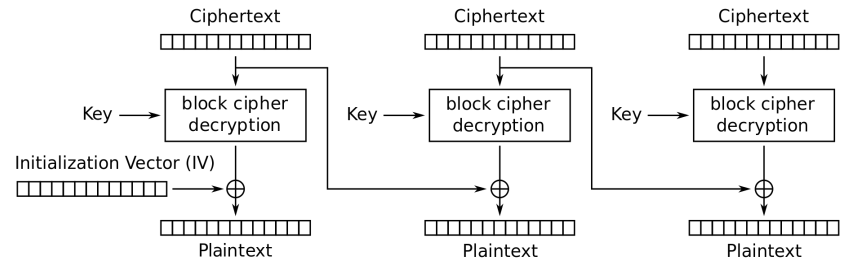
Cipher Block Chaining (CBC) mode encryption

CBC Mode: Decryption

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



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

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

$$D_K(C_i) \oplus C_{i-1} = M_i \oplus C_{i-1} \oplus C_{i-1}$$

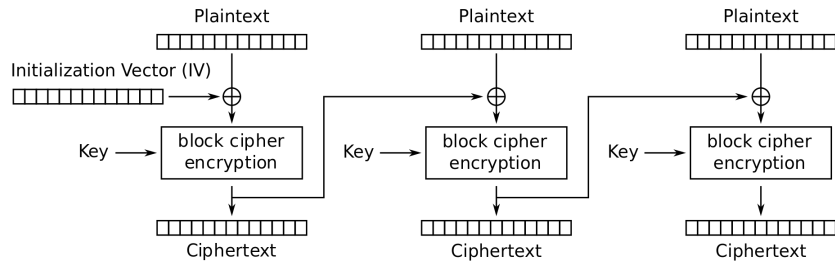
XOR both sides with C_{i-1}

$$D_K(C_i) \oplus C_{i-1} = M_i$$

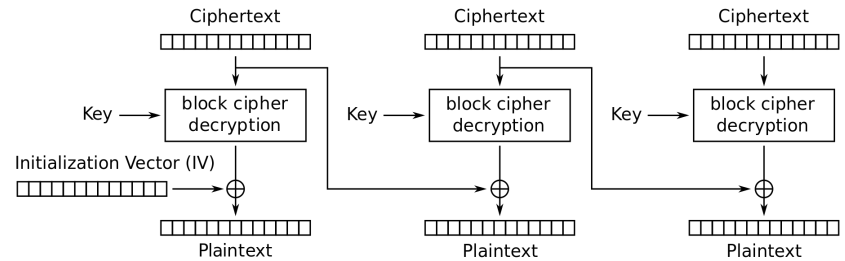
XOR property

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



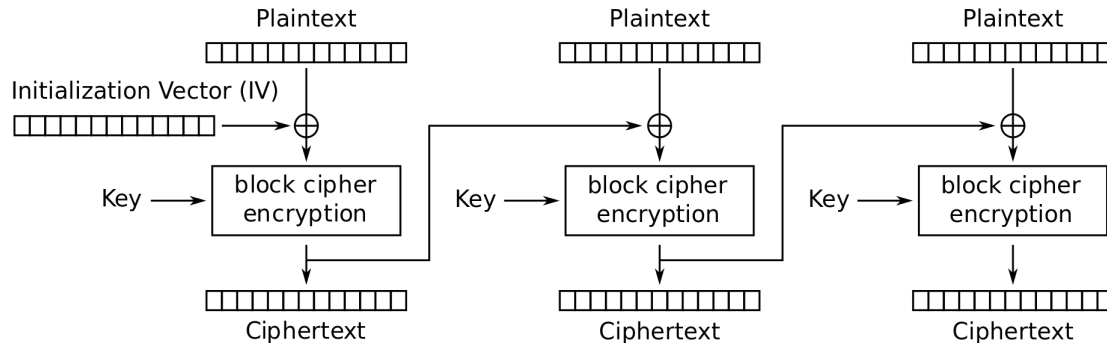
Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

CBC Mode: Padding

- 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

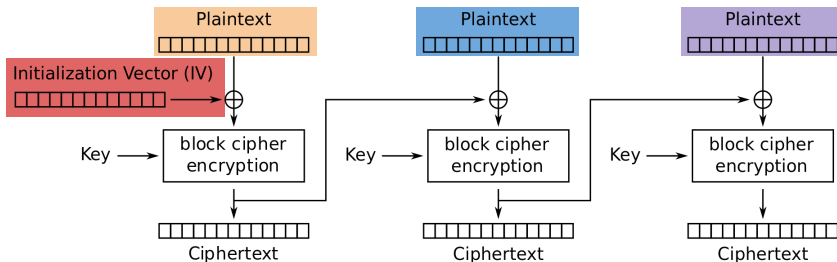
- What padding scheme should we use?
 - Padding with 0's?
 - Doesn't work: What if our message already ends with 0's?
 - Padding with 1's?
 - Same problem
- We need a scheme that can be unpadding without ambiguity
 - One scheme that works: Append a 1, then pad with 0's
 - If plaintext is multiple of 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
 - This is called PKCS #7

CBC Mode: Security

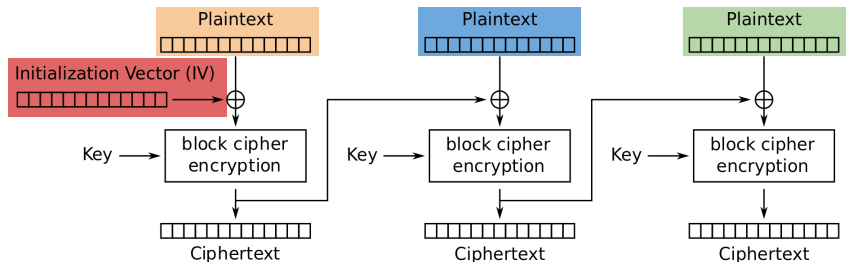
- AES-CBC is IND-CPA secure. With what assumption?
 - The IV must be randomly generated and never reused
- What happens if you reuse the IV?
 - The scheme becomes deterministic: No more IND-CPA security

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



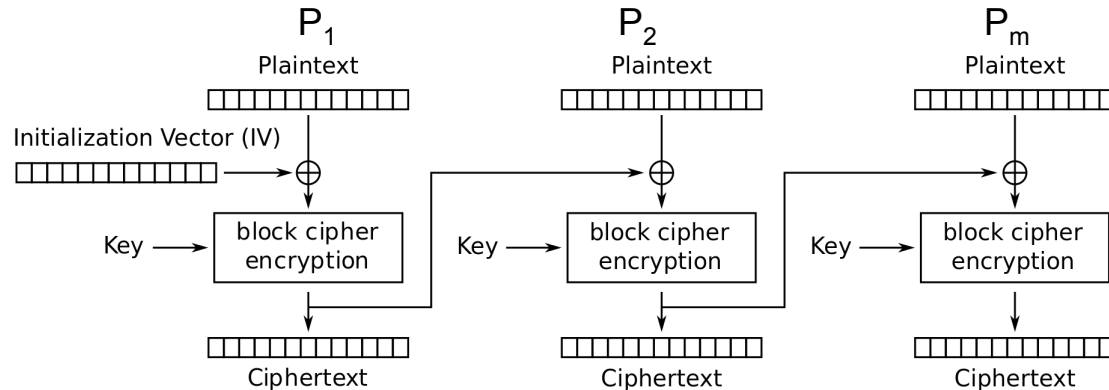
Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode encryption

CBC Mode is IND-CPA (when used correctly)

- $\text{Enc}(K, M)$:
 - Split M in m plaintext blocks $P_1 \dots P_m$ each of size n
 - Choose random IV , compute and output (IV, C_1, \dots, C_m) 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.



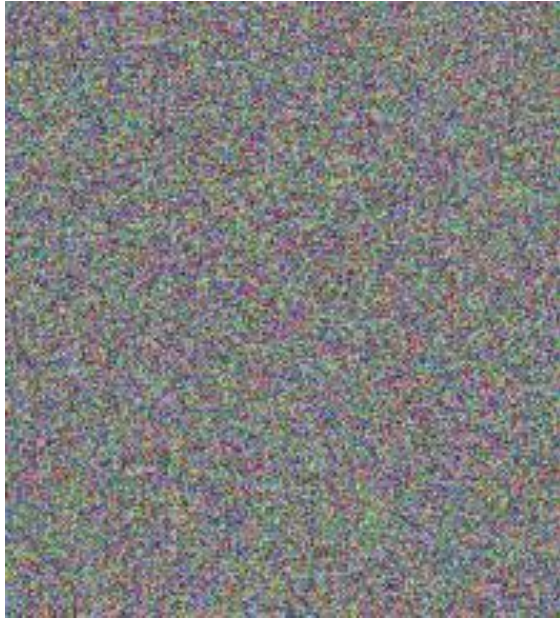
Cipher Block Chaining (CBC) mode encryption

CBC Mode: Penguin



Original image

CBC Mode: Penguin



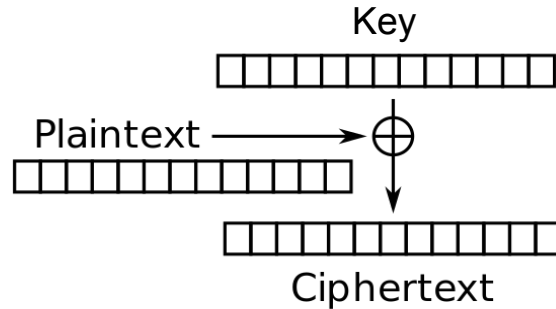
Encrypted with CBC, with random IVs

Block Cipher Modes of Operation: Summary

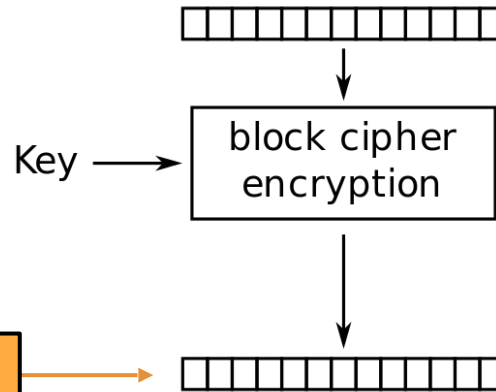
- ECB mode: Deterministic, so not IND-CPA secure
- CBC mode
 - IND-CPA secure, assuming no IV reuse
 - Encryption is not parallelizable
 - Decryption is parallelizable
 - Must pad plaintext to a multiple of the block size
 - IV reuse leads to leaking the existence of identical blocks at the start of the message

CTR Mode Scratchpad: Let's design it together

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



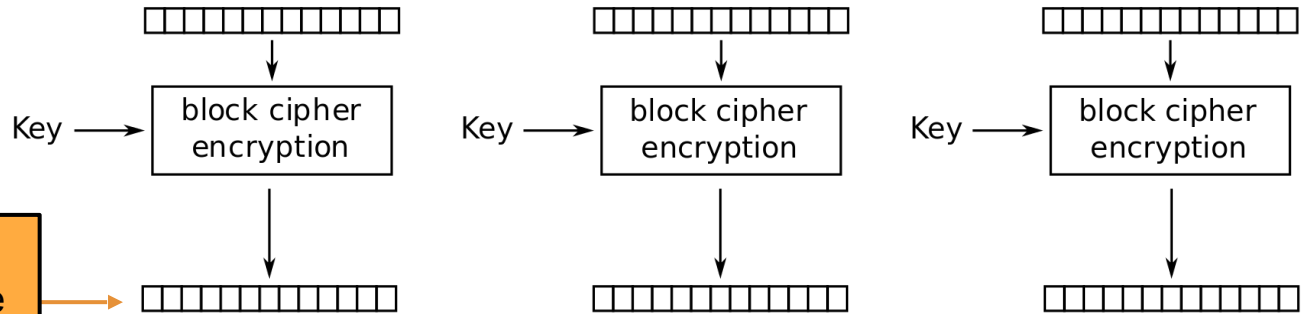
CTR Mode Scratchpad: Let's design it together



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

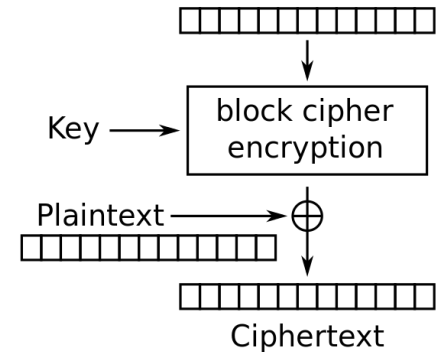
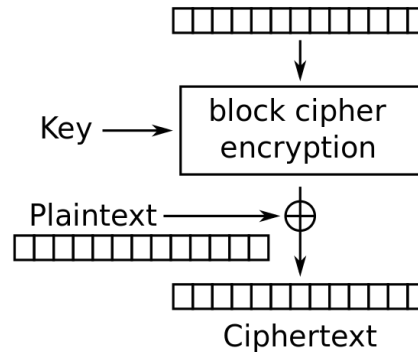
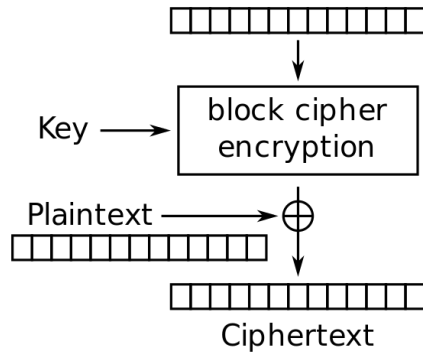


If the attacker doesn't know the key, all of these outputs look random.

CTR Mode Scratchpad: Let's design it together

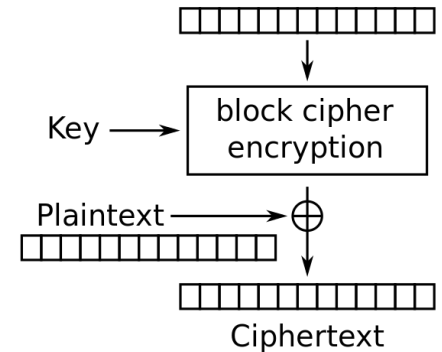
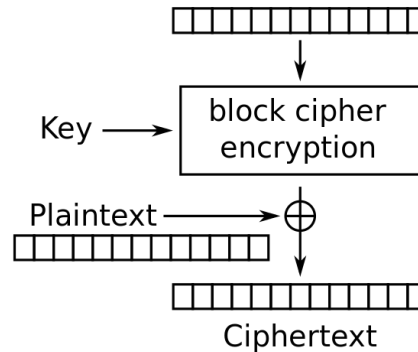
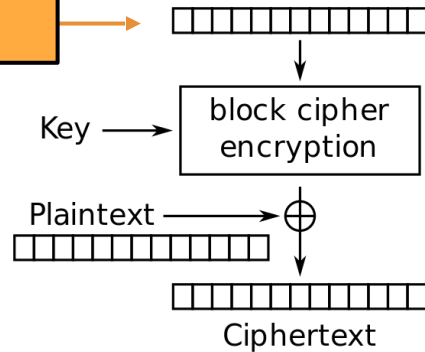
Idea: Use this random-looking output as a one-time pad!

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



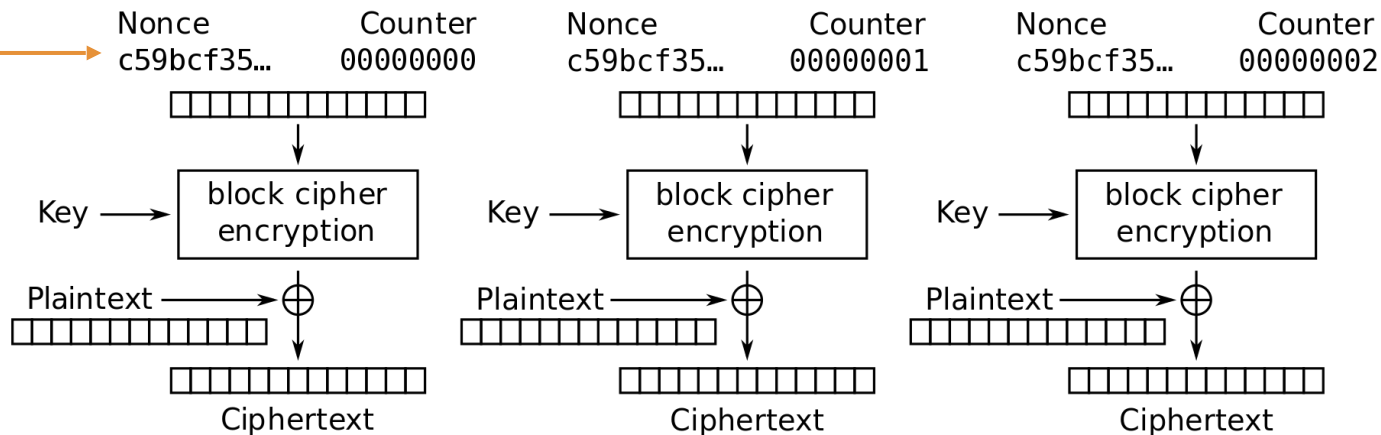
CTR Mode Scratchpad: Let's design it together

What do we use as input to the block cipher?



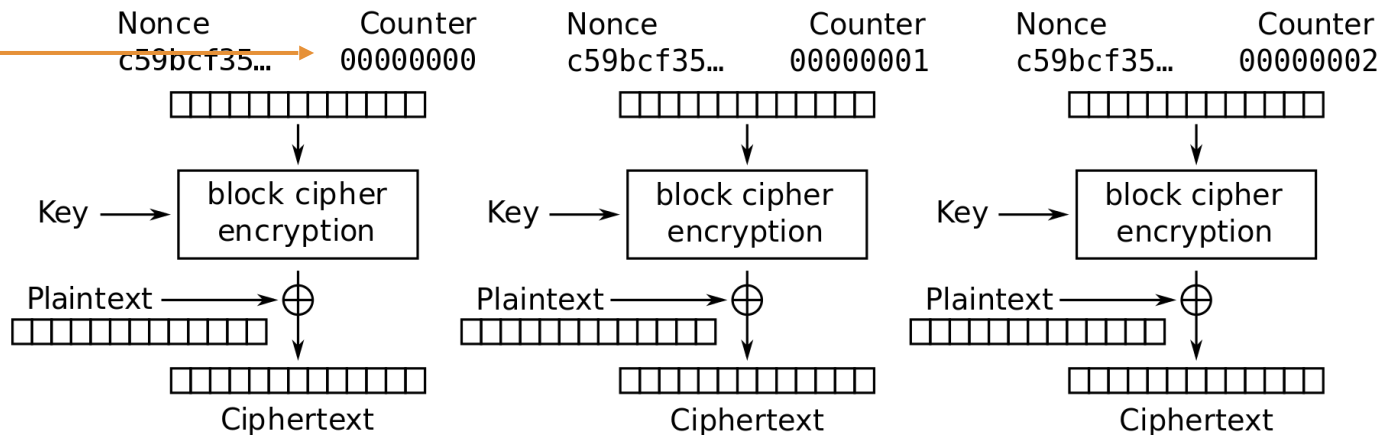
CTR Mode Scratchpad: Let's design it together

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



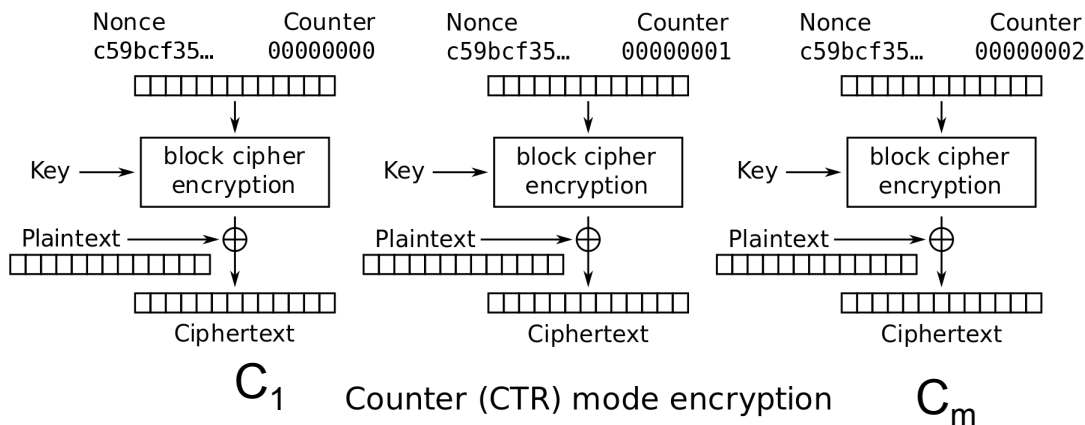
CTR Mode Scratchpad: Let's design it together

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



CTR (Counter) Mode

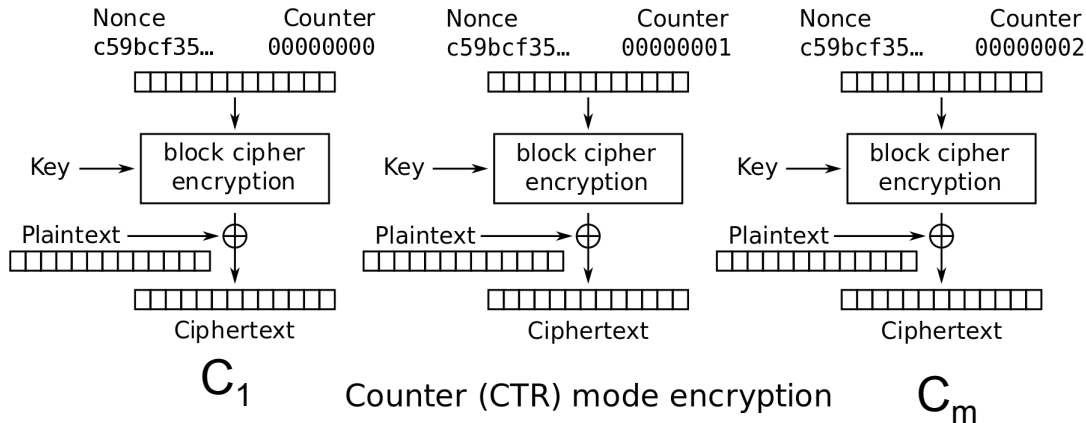
- Note: the random value is named the nonce here, but the idea is the same as the IV in CBC mode
- Overall ciphertext is (Nonce, C_1, \dots, C_m)



CTR Mode

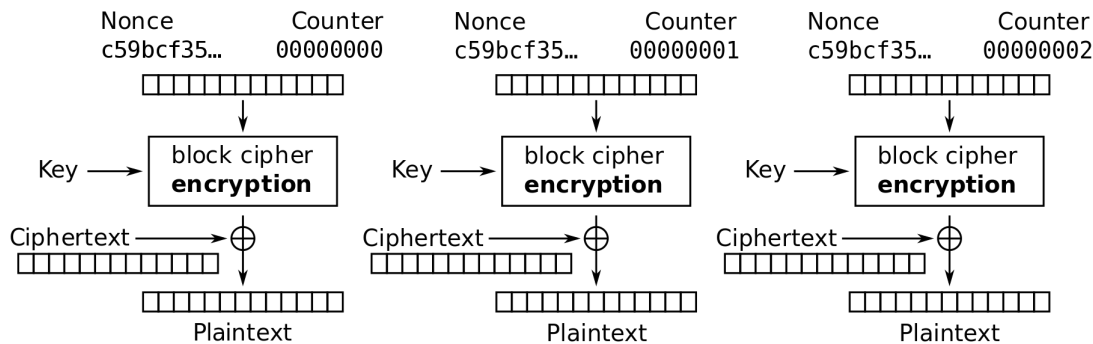
- $\text{Enc}(K, M)$:
 - Split M in plaintext blocks $P_1 \dots P_m$ (each of block size n)
 - Choose random nonce
 - Compute and output $(\text{Nonce}, C_1, \dots, C_m)$

How do you decrypt?



CTR Mode: Decryption

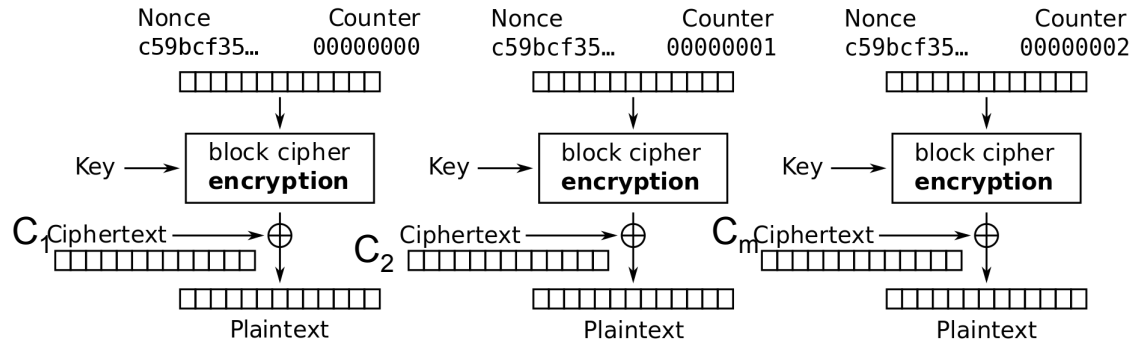
- Recall one-time pad: XOR with ciphertext to get plaintext
- Note: we are only using block cipher encryption, not decryption



Counter (CTR) mode decryption

CTR Mode: Decryption

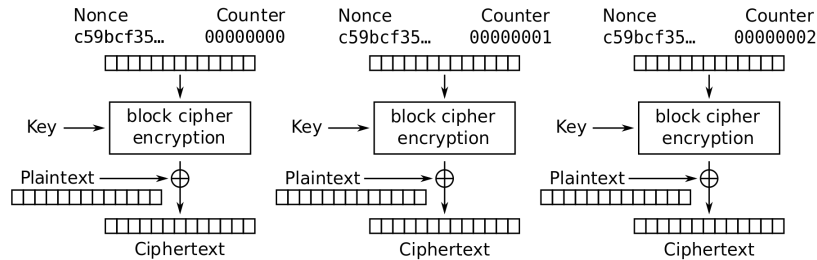
- Dec(K, C):
 - Parse C into (nonce, C_1, \dots, C_m)
 - Compute P_i by XORing C_i with output of E_k on nonce and counter
 - Concatenate resulting plaintexts and output $M = P_1 \dots P_m$



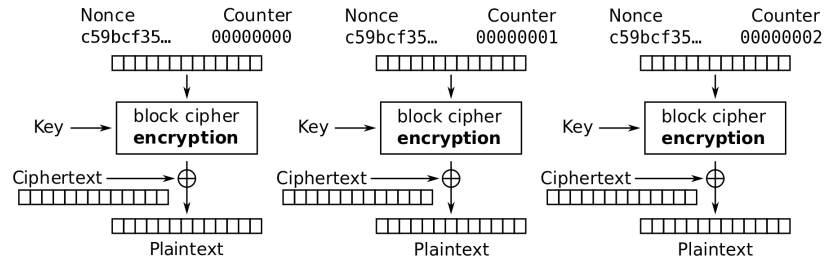
Counter (CTR) mode decryption

CTR Mode: Efficiency

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



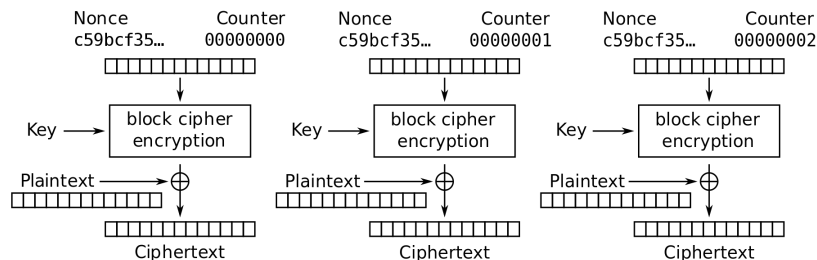
Counter (CTR) mode encryption



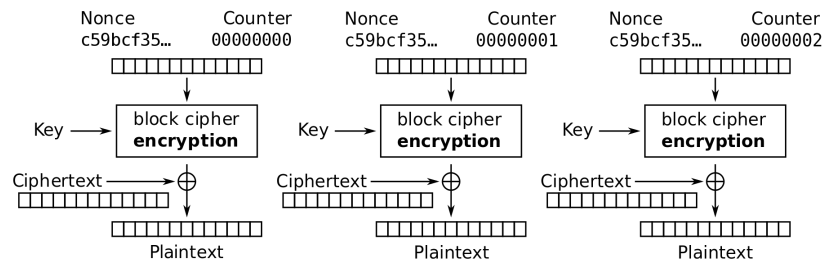
Counter (CTR) mode decryption

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

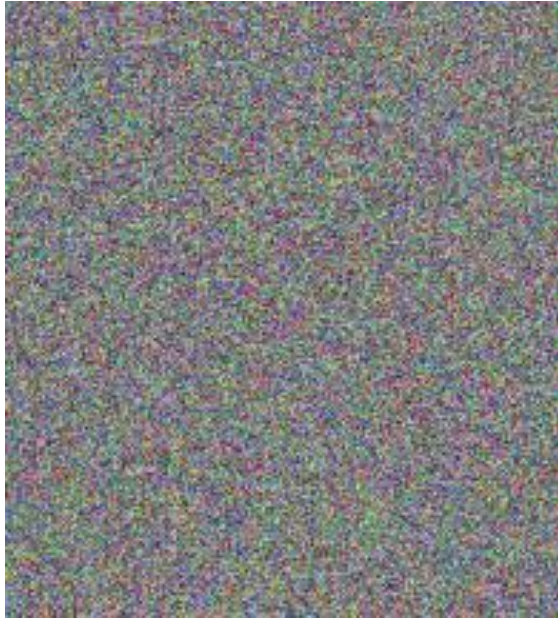
- AES-CTR is IND-CPA secure. With what assumption?
- The nonce should never be reused (random generation helps here)
 - And in general less than $2^{n/2}$ blocks are encrypted
- What happens if you reuse the nonce?
- Equivalent to reusing a key in a one-time pad
 - Recall: Key reuse in a one-time pad is catastrophic: usually leaks enough information for an attacker to deduce the entire plaintext

CTR Mode: Penguin



Original image

CTR Mode: Penguin



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
- **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)
- 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

Comparing Modes of Operation

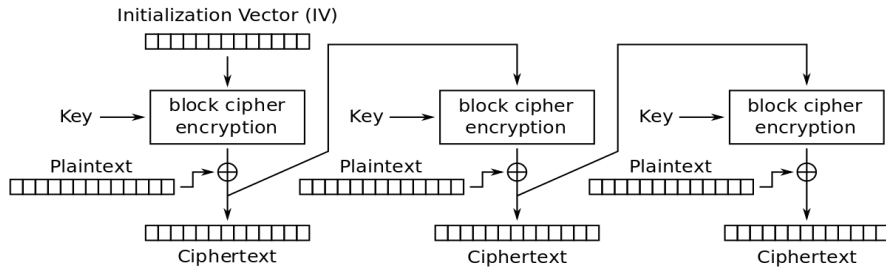
- If you need high performance, which mode is better?
 - CTR mode, because you can parallelize both encryption and decryption
- If you're paranoid about security, which mode is better?
 - CBC mode is better
- Theoretically, CBC and CTR mode are equally secure if used properly
 - However, if used improperly (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!

Other Modes of Operation

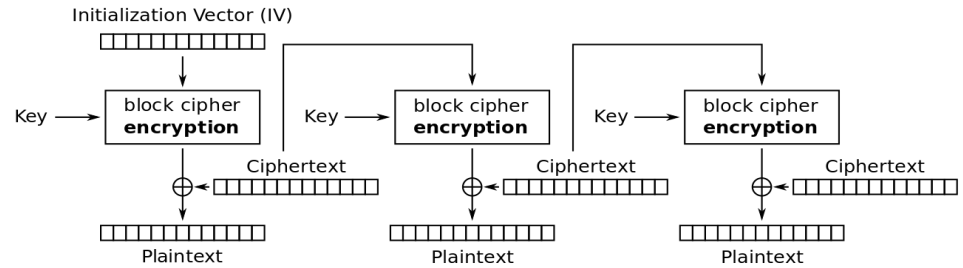
- Other modes exist besides CBC and CTR
- Trade-offs:
 - Do we need to pad messages?
 - How robust is the scheme if we use it incorrectly?
 - Can we parallelize encryption/decryption?

CFB Mode (Ciphertext Feedback Mode)

- Also IND-CPA
- Only encryption algorithm is used in CFB mode decryption

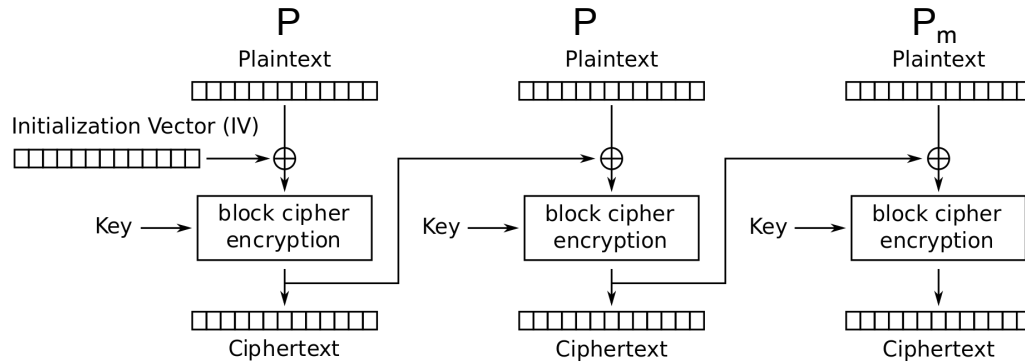


Cipher Feedback (CFB) mode encryption

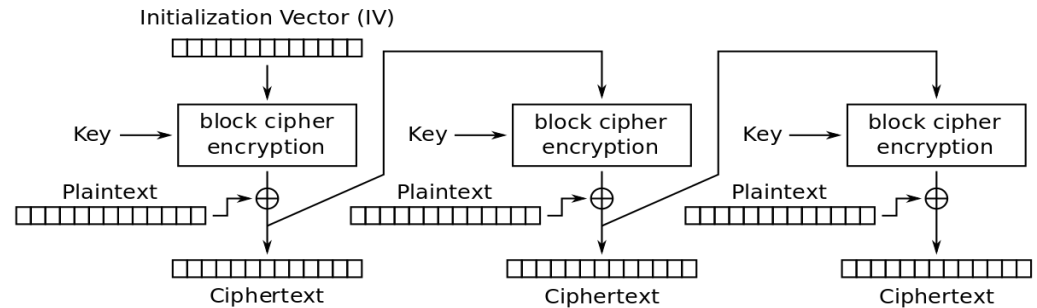


Cipher Feedback (CFB) mode decryption

Recall: CBC Mode



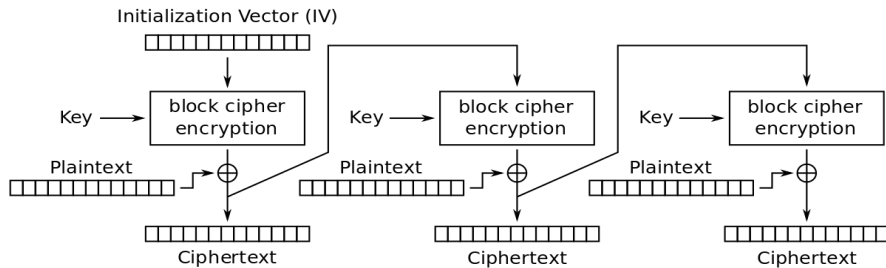
Cipher Block Chaining (CBC) mode encryption



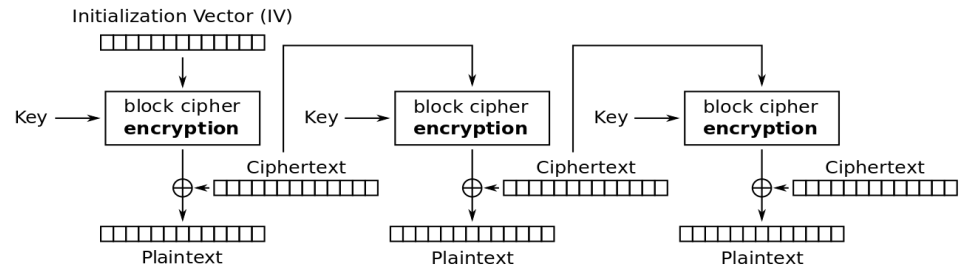
Cipher Feedback (CFB) mode encryption

CFB Mode

- Also IND-CPA
- Try to analyze the trade-offs yourself:
 - Do we need to pad messages?
 - No
 - How robust is the scheme if we use it incorrectly?
 - Similar effects as CBC mode, but a bit worse if you reuse the IV
 - Can we parallelize encryption/decryption?
 - Only decryption is parallelizable



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

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
 - **Integrity:** An adversary cannot *change* our messages without being detected.
 - **Authenticity:** I can prove that this message came from the person who claims to have written 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
<i>M</i>	0x50	0x61	0x79	0x20	0x4d	0x61	0x6c	0x20	0x24	0x31	0x30	0x30
	\oplus											
<i>E_K(i)</i>	0x8a	0xe3	0x5e	0xcf	0x3b	0x40	0x46	0x57	0xb8	0x69	0xd2	0x96
	=											
<i>C</i>	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?**
 - Change 0x31 to 0x39

	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

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

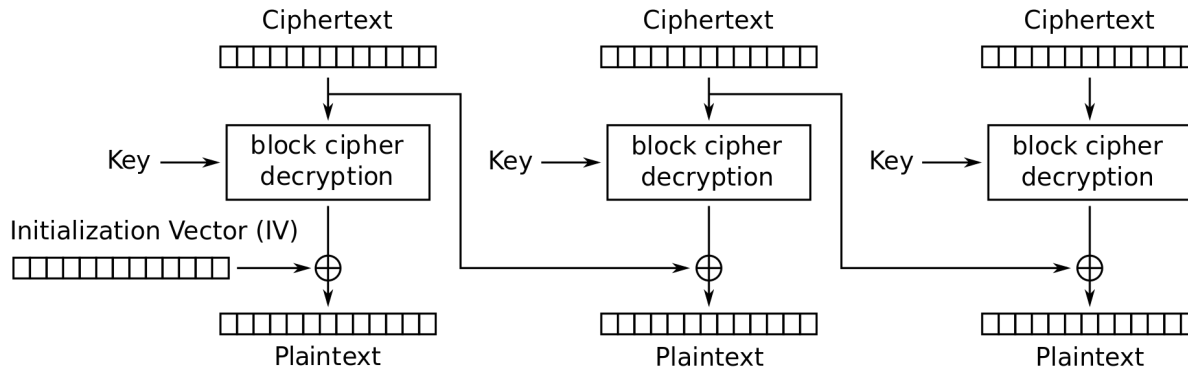
Lack of Integrity and Authenticity

- 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
	=											
P'	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 doesn't know that Alice did not send random gibberish, so it still does *not* provide integrity or authenticity



Cipher Block Chaining (CBC) mode decryption

Agenda

- Block Cipher Chaining Modes
- Cryptographic Hashes

Cryptographic Hash Function: Definition

- Hash function: $H(M)$
 - Input: *Arbitrary* length message M
 - Output: *Fixed* length, n -bit hash
 - Sometimes written as $\{0, 1\}^* \rightarrow \{0, 1\}^n$

Cryptographic Hash Function: Properties

- **Correctness:** Deterministic
 - Hashing the same input always produces the same output
- **Efficiency:** Efficient to compute
- **Security:** One-way-ness (“preimage resistance”)
- **Security:** Collision-resistance
- **Security:** Random/unpredictability, no predictable patterns for how changing the input affects the output
 - Changing 1 bit in the input causes the output to be completely different
 - Also called “random oracle” assumption

Hash Function: Intuition

- A hash function provides a fixed-length “fingerprint” over a sequence of bits
- Example: Document comparison
 - If Alice and Bob both have a 1 GB document, they can both compute a hash over the document and (securely) communicate the hashes to each other
 - If the hashes are the same, the files must be the same, since they have the same “fingerprint”
 - If the hashes are different, the files must be different

Hash Function: One-way-ness or Preimage Resistance

- **Informal:** Given an output y , it is **infeasible** to find **any** input x such that $H(x) = y$
- **Intuition:** Here's an output. Can you find an input that hashes to this output?
 - Note: The adversary just needs to find *any* input, not necessarily the input that was actually used to generate the hash

Hash Function: Collision Resistance

- **Collision:** Two different inputs with the same output
- **Collision resistance:** It is infeasible to (i.e. no polynomial time attacker can) find any pair of inputs $x' \neq x$ such that $H(x) = H(x')$
- Intuition: Can you find *any* two inputs that collide with the same hash output for *any* output?

Hash Function: Second Preimage Resistance

- **Second preimage resistance:** Given an input x , it is infeasible to (i.e. no polynomial time attacker can) find another input $x' \neq x$ such that $H(x) = H(x')$
- Collision resistance implies second preimage resistance

Hash Function: Examples

- MD5
 - Output: 128 bits
 - Security: Completely broken
- SHA-1
 - Output: 160 bits
 - Security: Completely broken in 2017
 - Was known to be weak before 2017, but still used sometimes
- SHA-2
 - Output: 256, 384, or 512 bits (sometimes labeled SHA-256, SHA-384, SHA-512)
 - Not currently broken, but some variants are vulnerable to a length extension attack
 - Current standard
- SHA-3 (Keccak)
 - Output: 256, 384, or 512 bits
 - Current standard (not meant to replace SHA-2, just a different construction)

Length Extension Attacks

- **Length extension attack:** Given $H(x)$ and the length of x , but not x , an attacker can create $H(x || m)$ for any m of the attacker's choosing
 - Note: This doesn't violate any property of hash functions but is undesirable in some circumstances
- SHA-256 (256-bit version of SHA-2) is vulnerable
- SHA-3 is not vulnerable

Do hashes provide integrity?

- It depends on your threat model
- Scenario
 - Mozilla publishes a new version of Firefox on some download servers
 - Alice downloads the program binary
- Idea: use cryptographic hashes
- Threat model: We assume the attacker cannot modify the hash on the website
 - We have integrity, as long as we can communicate the hash securely

Do hashes provide integrity?

- It depends on your threat model
- Scenario
 - Alice and Bob want to communicate over an insecure channel
 - Mallory might tamper with messages
- Idea: Use cryptographic hashes
 - Alice sends her message with a cryptographic hash over the channel
- Threat model: Mallory can modify the message *and the hash*
 - No integrity!

Do hashes provide integrity?

- It depends on your threat model
- If the attacker can modify the hash, hashes don't provide integrity
- Main issue: Hashes are *unkeyed functions*
 - There is no secret key being used as input, so any attacker can compute a hash on any value
- Next: Use hashes to design schemes that provide integrity