CMSC414 Computer and Network Security

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

Yizheng Chen | University of Maryland
surrealyz.github.io

Mar 28, 2024

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

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)

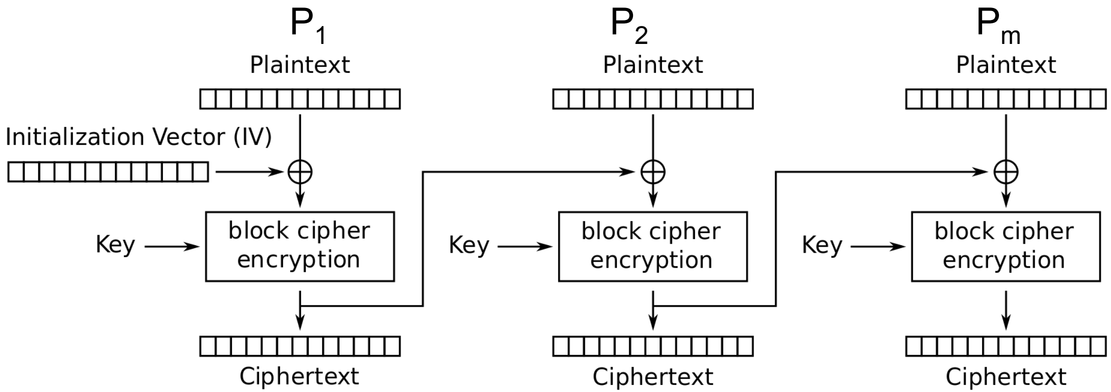
- Implementation: AES is the modern standard
- Issues
  - Not IND-CPA secure because they're deterministic
  - Can only encrypt *n*-bit messages

Block Cipher Modes of Operation: Summary

- ECB mode: Deterministic, so not IND-CPA secure
  - ECB stands for Electronic Code Book

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 $P_1 \ldots P_m$ each of size n
  - Choose a random IV
  - Compute and output (IV, $C_1, \ldots, C_m$) as the overall ciphertext



Cipher Block Chaining (CBC) mode encryption

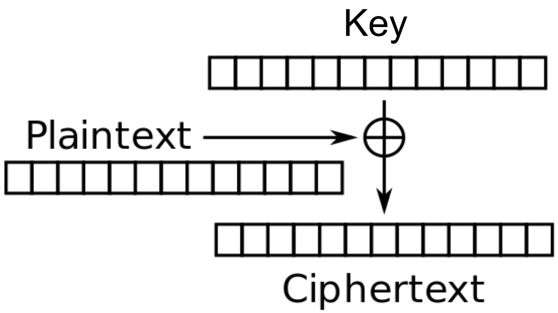
Block Cipher Modes of Operation: Summary

- ECB mode: Deterministic, so not IND-CPA secure

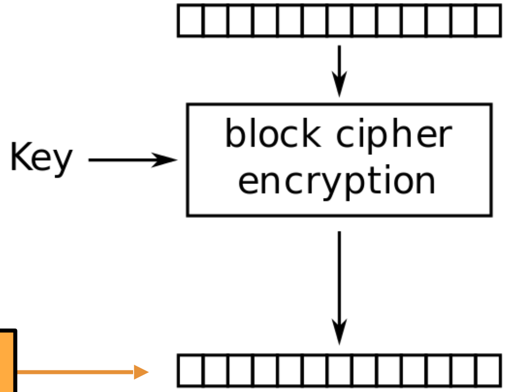
- CBC mode
  - IND-CPA secure, assuming no IV reuse
  - Encryption is not parallelizable
  - Decryption is parallelizable
  - Must pad plaintext to a multiple of the block size
  - IV reuse leads to leaking the existence of identical blocks at the start of the message

CTR Mode Scratchpad: Let's design it together

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



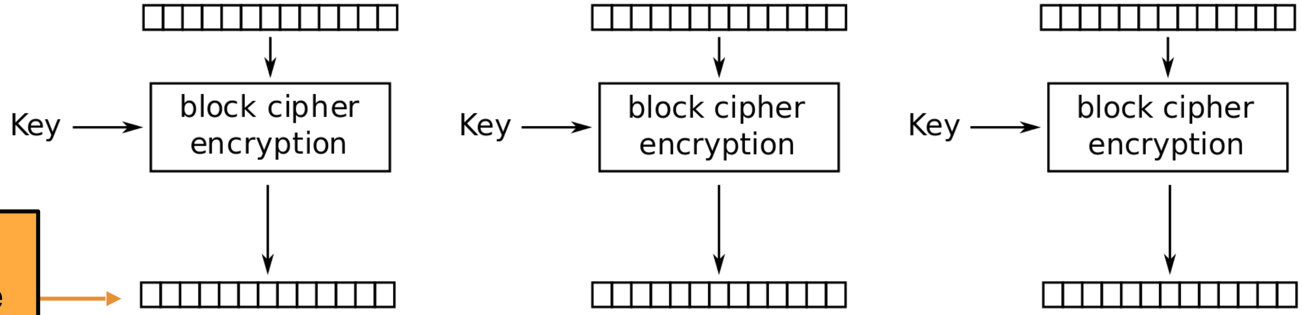
CTR Mode Scratchpad: Let's design it together



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

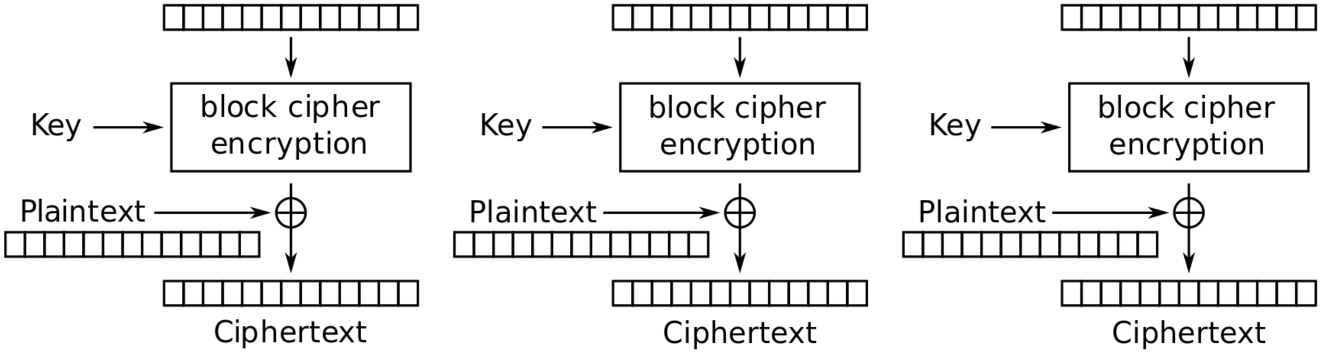
New idea: Can we use block ciphers to simulate a one-time pad?

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

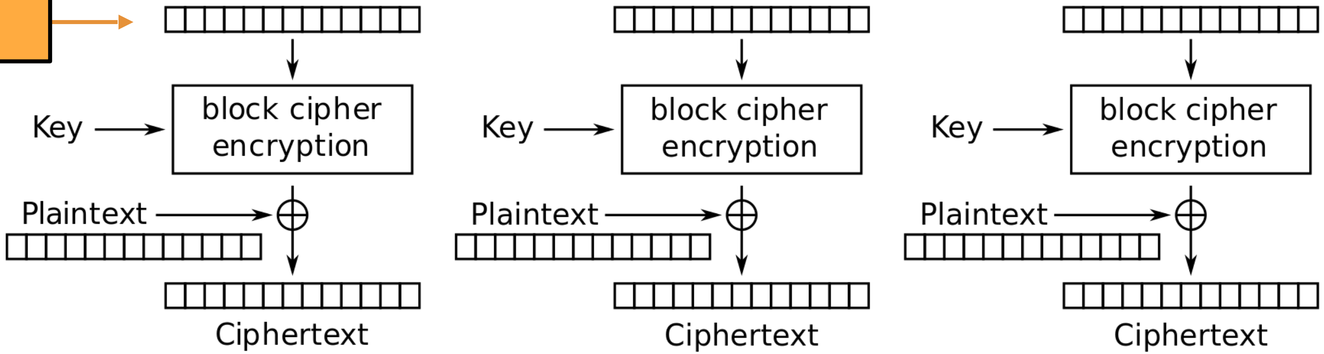


Idea: Use this random-looking output as a one-time pad!

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

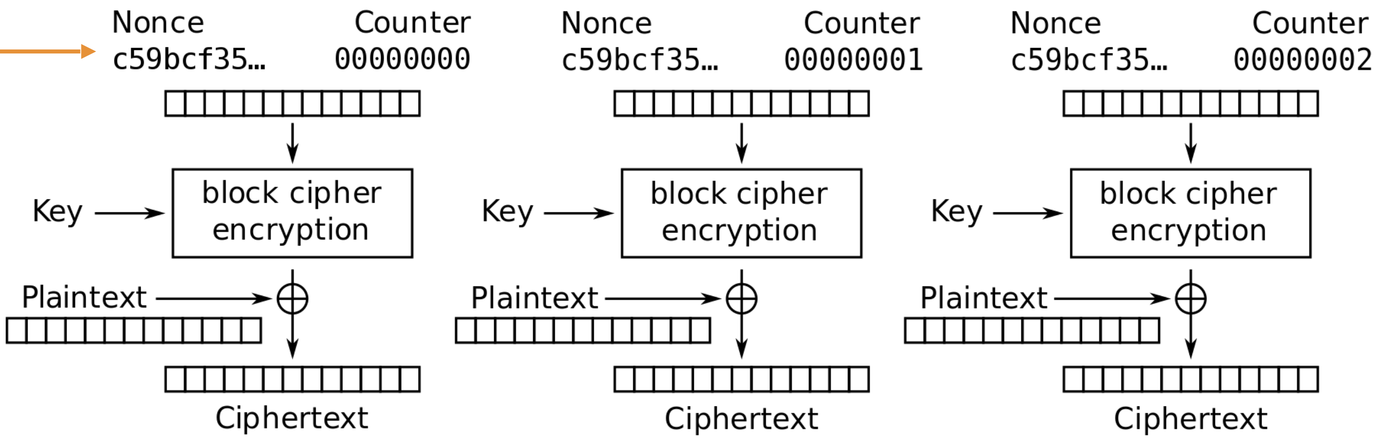
CTR Mode Scratchpad: Let's design it together

What do we use as input to the block cipher?



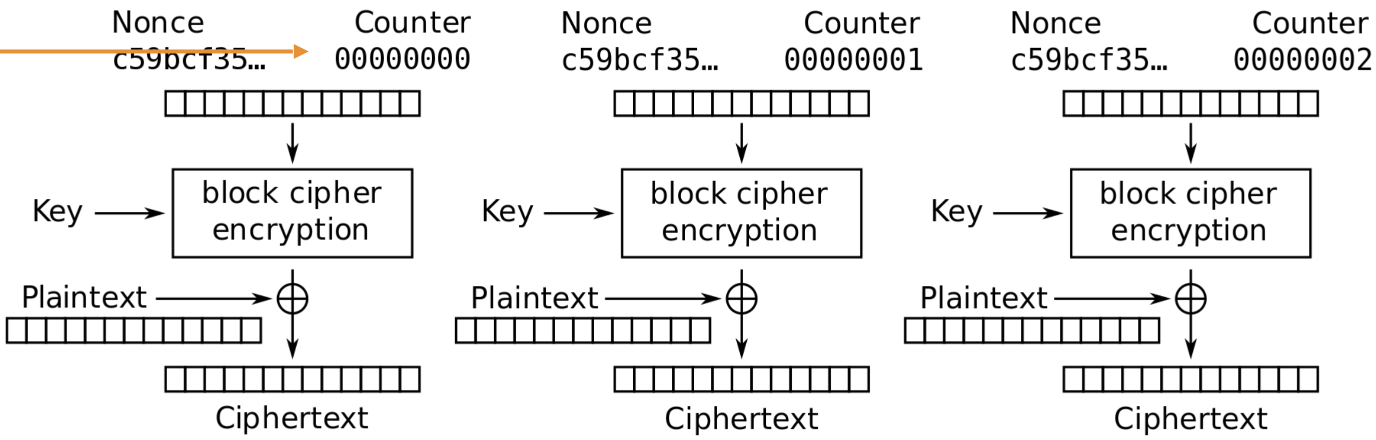
CTR Mode Scratchpad: Let's design it together

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



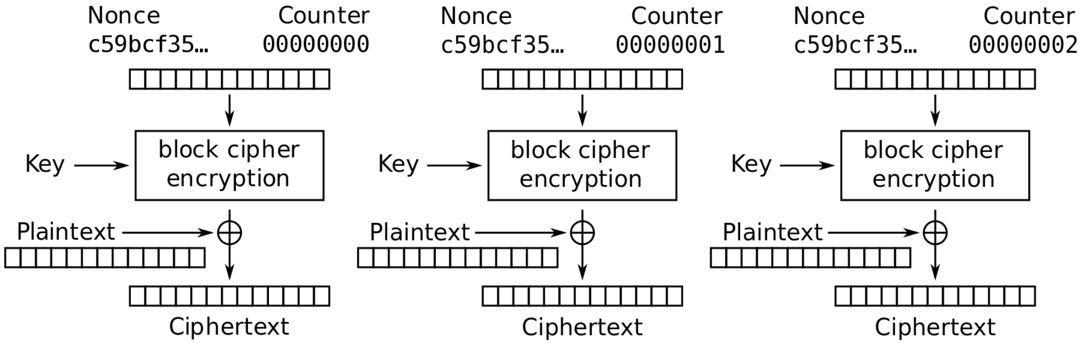
CTR Mode Scratchpad: Let's design it together

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, …, C_m)

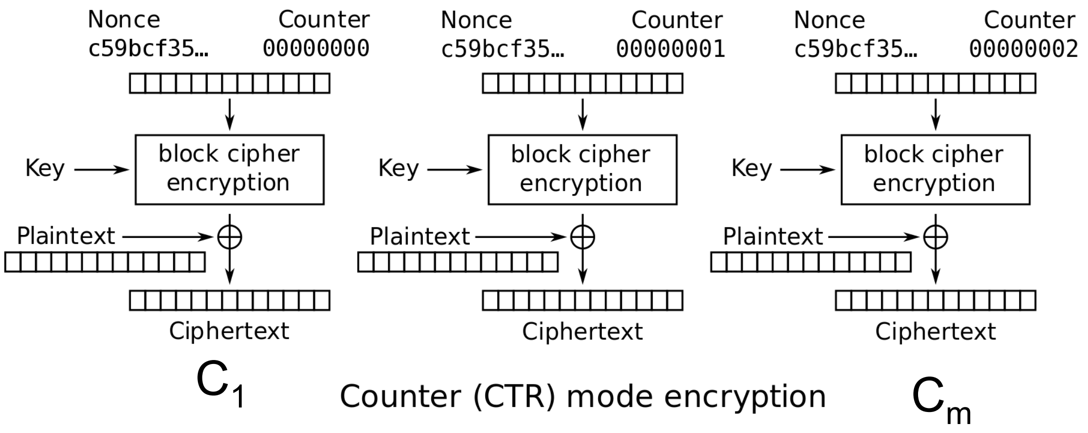


C_1 Counter (CTR) mode encryption C_m

CTR Mode

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

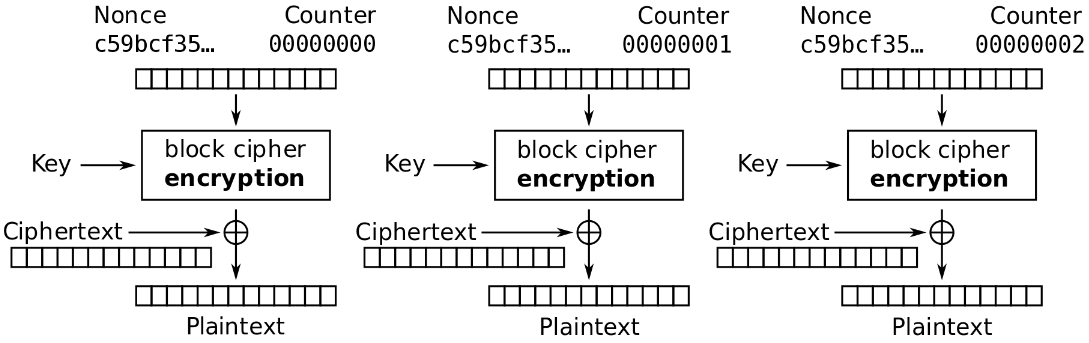
How do you decrypt?



Counter (CTR) mode encryption

CTR Mode: Decryption

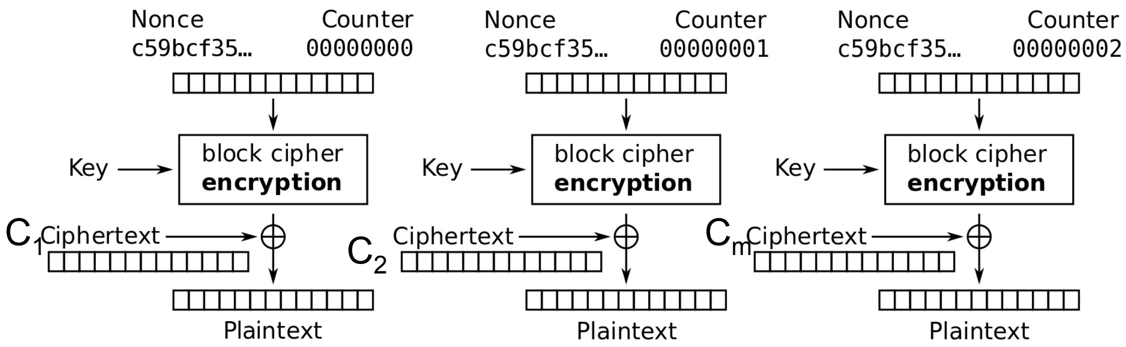
- Recall one-time pad: XOR with ciphertext to get plaintext
- Note: we are only using block cipher encryption, not decryption



Counter (CTR) mode decryption

CTR Mode: Decryption

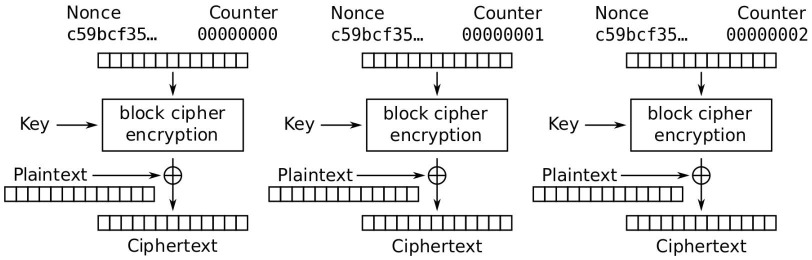
- Dec(K, C):
  - Parse C into (nonce, C_1, …, C_m)
  - Compute P_i by XORing Ci with output of E_k on nonce and counter
  - Concatenate resulting plaintexts and output M = P_1 … P_m



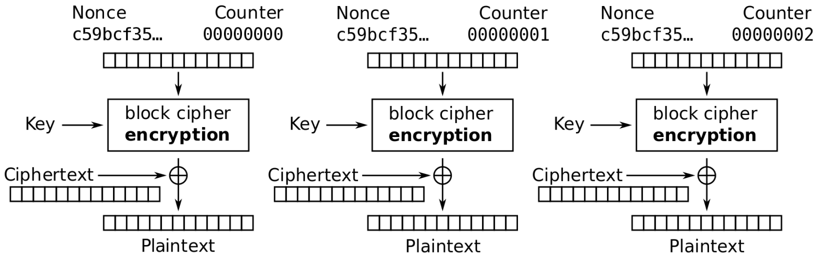
Counter (CTR) mode decryption

CTR Mode: Efficiency

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



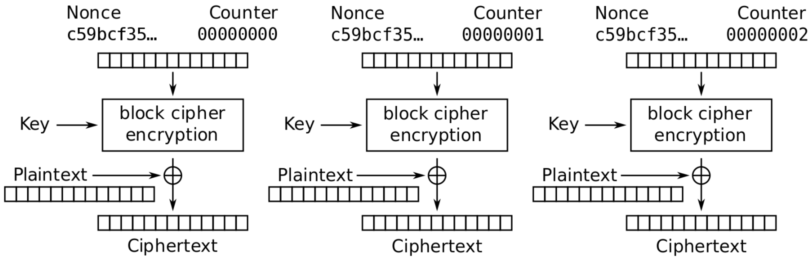
Counter (CTR) mode encryption



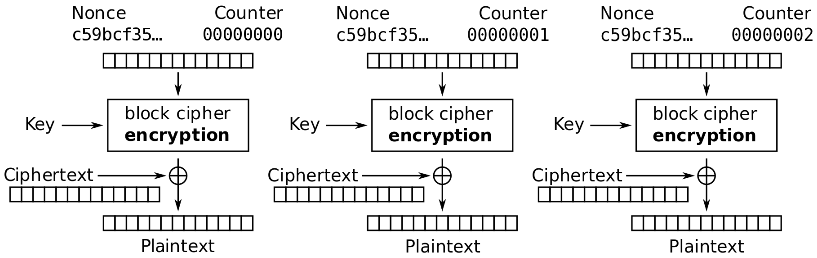
Counter (CTR) mode decryption

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

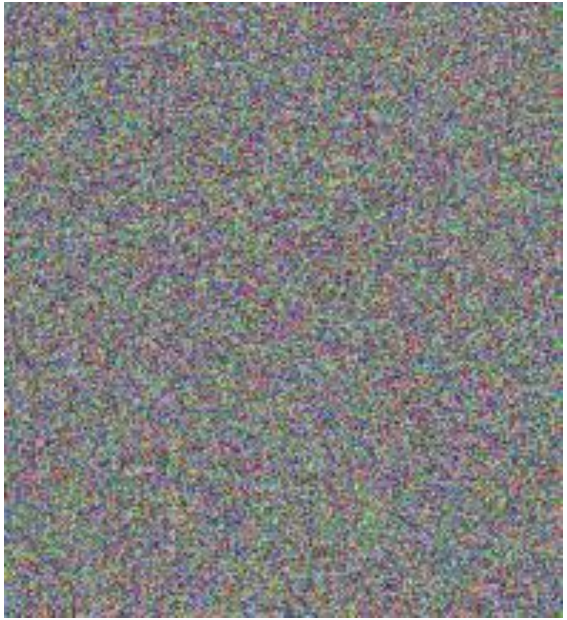
- What happens if you reuse the nonce?

- 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

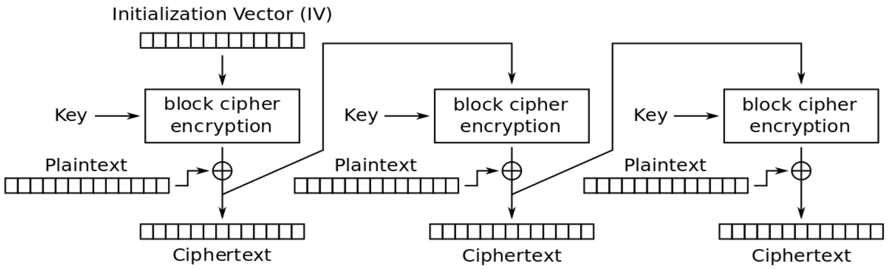
- Theoretically, CBC and CTR mode are equally secure if used properly
  - However, if used improperly (IV/nonce reuse), CBC only leaks partial information, and CTR fails catastrophically
    - Consider human factors: Systems should be as secure as possible even when implemented *incorrectly*
  - IV failures on CTR mode have resulted in multiple real-world security incidents!

Other Modes of Operation

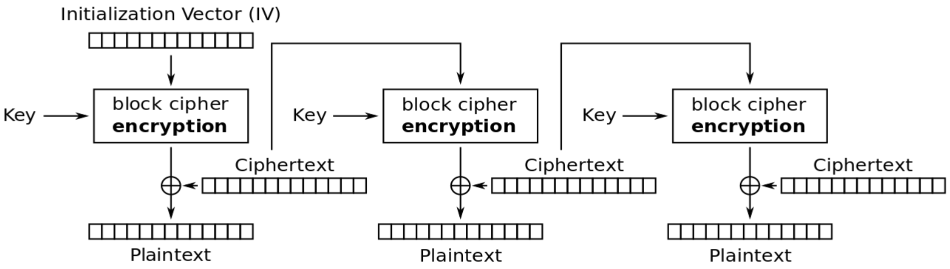
- Other modes exist besides CBC and CTR
- Trade-offs:
  - Do we need to pad messages?
  - How robust is the scheme if we use it incorrectly?
  - Can we parallelize encryption/decryption?

CFB Mode

- Also IND-CPA
- Try to analyze the trade-offs yourself:
  - Do we need to pad messages?
  - How robust is the scheme if we use it incorrectly?
  - Can we parallelize encryption/decryption?



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

CFB Mode

- 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

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`**?
  - 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

⊕

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

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

⊕

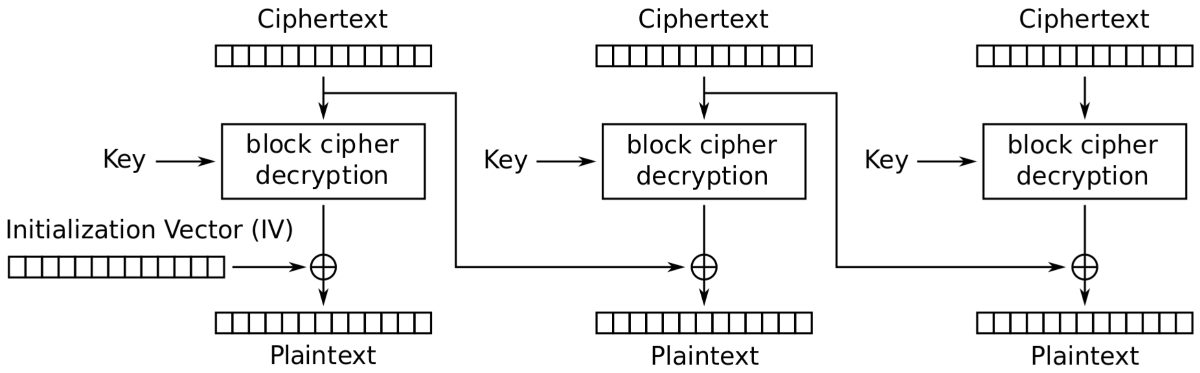
$E_K(i)$	0x8a	0xe3	0x5e	0xcf	0x3b	0x40	0x46	0x57	0xb8	**0x69**	0xd2	0x96

=

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

Today: Cryptography Hashes and MACs

- Hashing
  - Definition
  - Security: one-way, second preimage resistant, collision resistant
  - Examples
  - Length extension attacks
  - Application: Lowest-hash scheme
  - Do hashes provide integrity?
- MACs
  - Definition
  - Security: unforgeability
  - Example: HMAC
  - Do MACs provide integrity?
- Authenticated Encryption
  - Definition
  - Key Reuse
  - MAC-then-Encrypt or Encrypt-then-MAC?
  - AEAD Encryption Modes

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

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: 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

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)

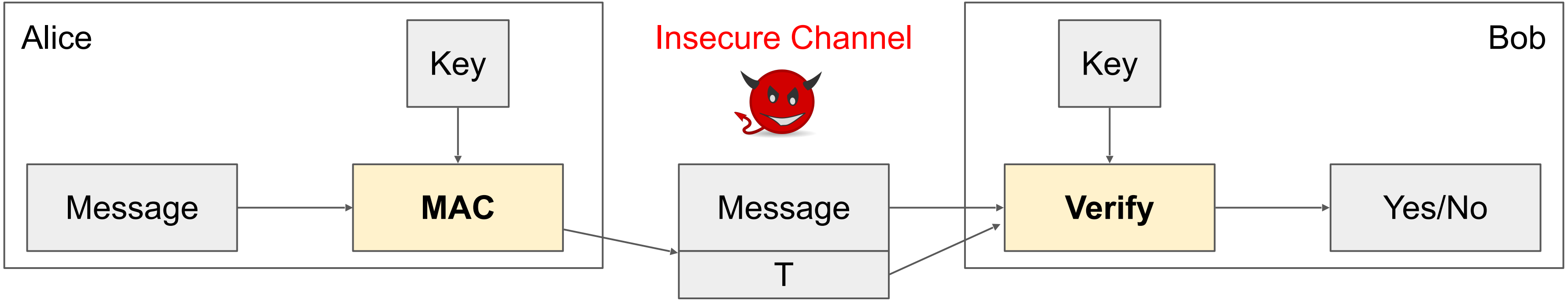
- Key management (certificates)
- Password management

How to Provide Integrity

- Reminder: We're still in the symmetric-key setting
  - Assume that Alice and Bob share a secret key, and attackers don't know the key
- We want to attach some piece of information to *prove* that someone with the key sent this message
  - This piece of information can only be generated by someone with the key

Message Authentication Codes (MACs)

- Alice wants to send M to Bob, but doesn't want Mallory to tamper with it
- Alice sends M and T = MAC(K, M) to Bob
- Bob recomputes MAC(K, M) and checks that it matches T
- If the MACs match, Bob is confident the message has not been tampered with (integrity)



MACs: Definition

- Two parts:
  - KeyGen() → K: Generate a key K
  - MAC(K, M) → T: Generate a tag T for the message M using key K
    - Inputs: A secret key and an arbitrary-length message
    - Output: A fixed-length **tag** on the message
- Properties
  - **Correctness**: Determinism
    - Note: Some more complicated MAC schemes have an additional Verify(K, M, T) function that don't require determinism, but this is out of scope
  - **Efficiency**: Computing a MAC should be efficient
  - **Security**: EU-CPA (existentially unforgeable under chosen plaintext attack)

Defining Integrity: EU-CPA

- A secure MAC is **existentially unforgeable**: without the key, an attacker cannot create a valid tag on a message
  - Mallory cannot generate MAC(K, M') without K
  - Mallory cannot find any $M' \neq M$ such that MAC(K, M') = MAC(K, M)

- Formally defined by a security game: existential unforgeability under chosen-plaintext attack, or EU-CPA

- MACs should be unforgeable under chosen plaintext attack
  - Intuition: Like IND-CPA, but for integrity and authenticity
  - Even if Mallory can trick Alice into creating MACs for messages that Mallory chooses, Mallory cannot create a valid MAC on a message that she hasn't seen before

Defining Integrity: EU-CPA

1. Mallory may send messages to Alice and receive their tags
2. Eventually, Mallory creates a message-tag pair (M', T')
   - M' cannot be a message that Mallory requested earlier
   - If T' is a valid tag for M', then Mallory wins. Otherwise, she loses.
3. A scheme is EU-CPA secure if for *all* polynomial time adversaries, the probability of winning is 0 or negligible