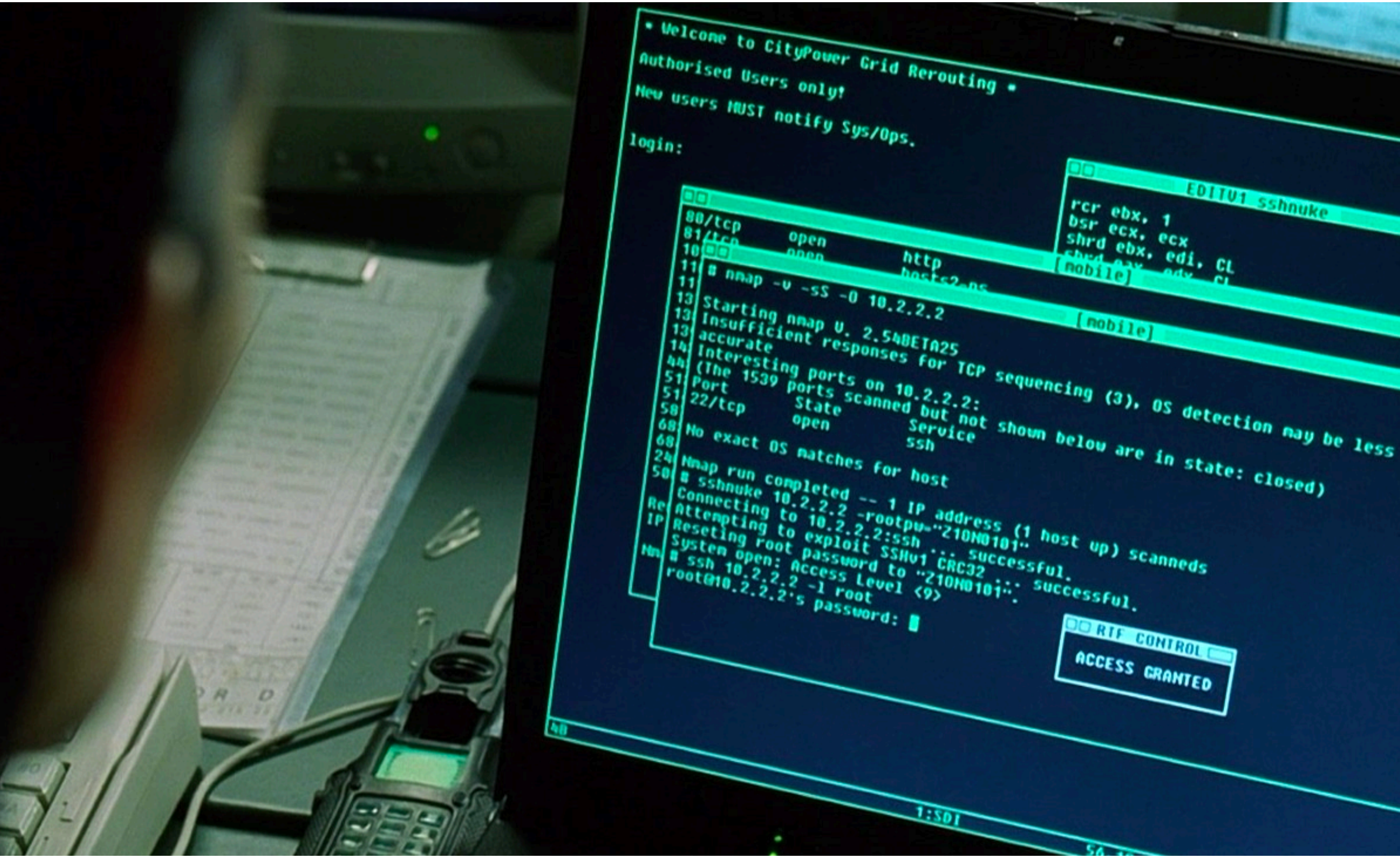
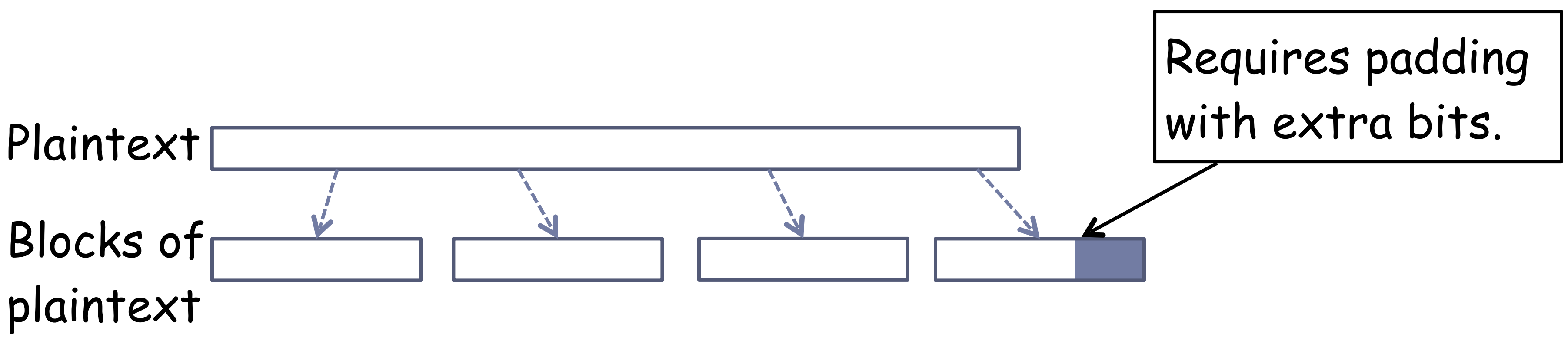


* Slides adopted from Prof. Manos Antonakakis

Continuing with symmetric crypto

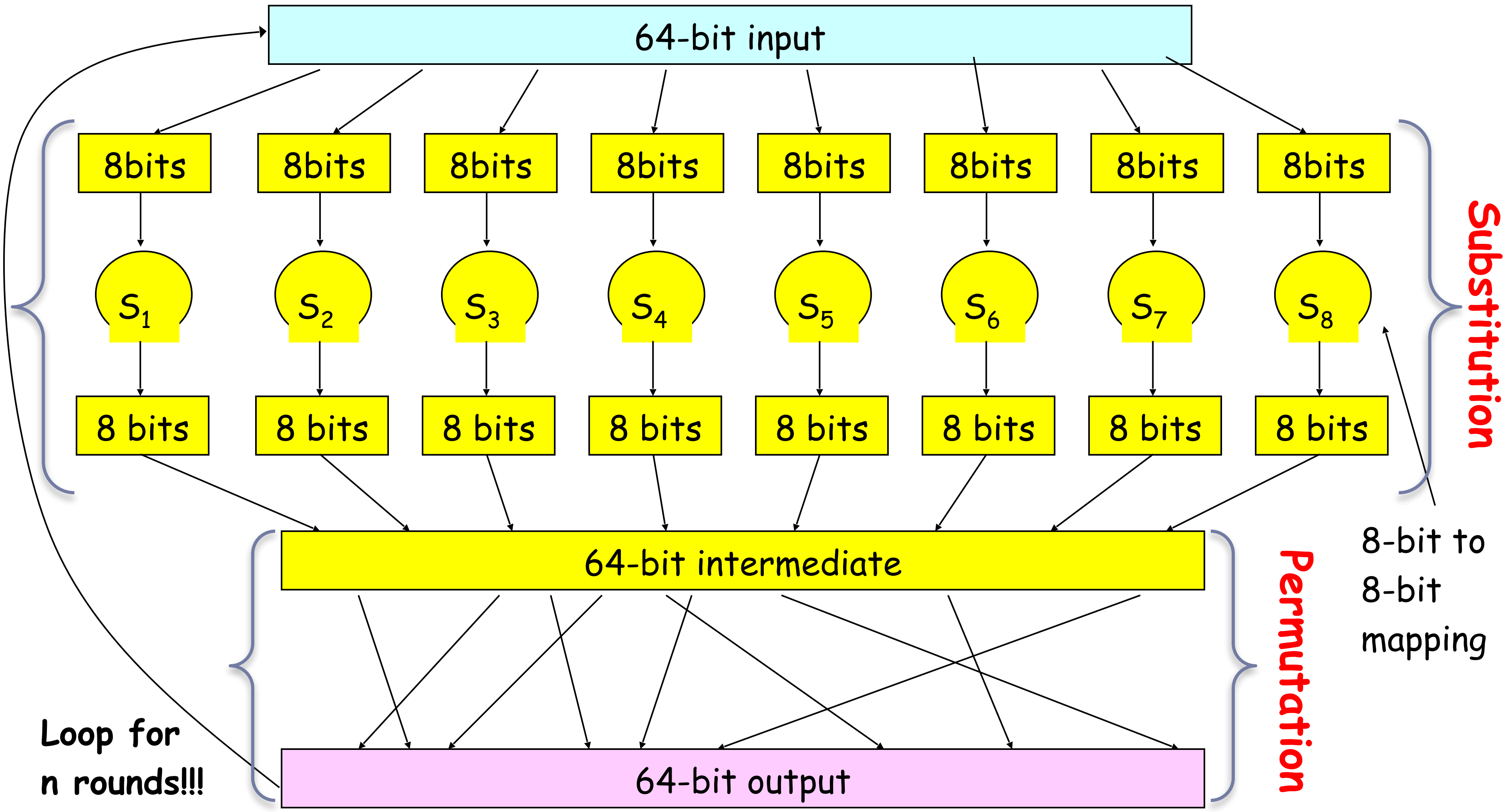
Block Ciphers

- In a **block cipher:**
  - Each message is divided into a sequence of blocks and encrypted or decrypted in terms of its blocks.



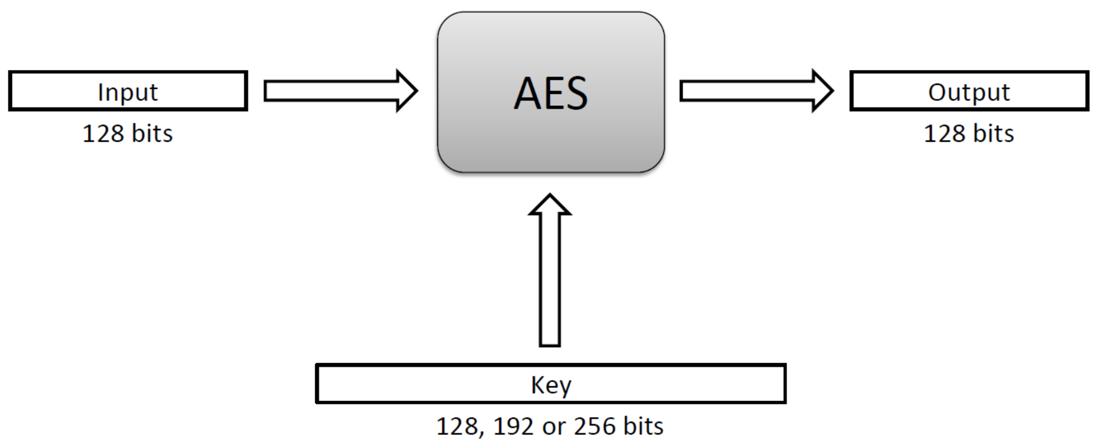
From Kaufman et al

Prototype function: Block Encryption



The Advanced Encryption Standard (AES)

- AES is a block cipher that operates on 128-bit blocks. It is designed to be used with keys that are 128, 192, or 256 bits long, yielding ciphers known as AES-128, AES-192, and AES-256.



Encrypting Large Messages

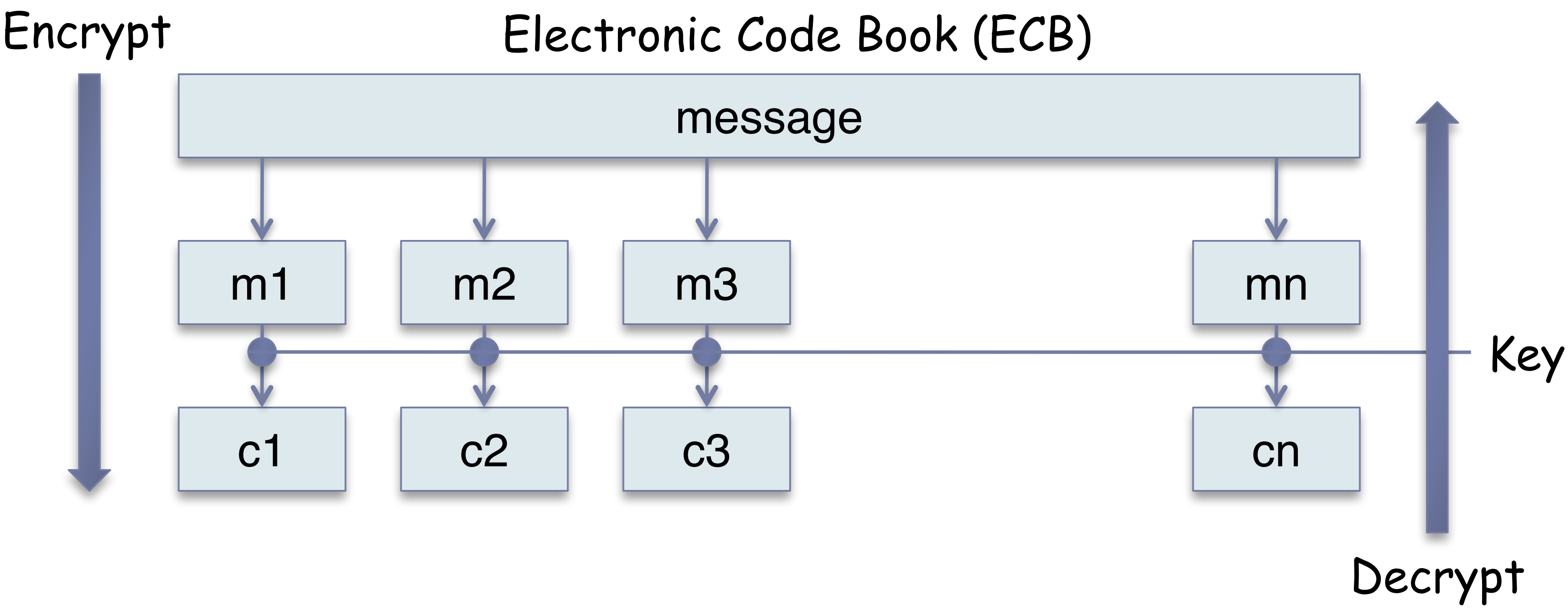
- Block ciphers only operate on messages of a fixed size (e.g.,128-bit)

- How do you encrypt messages larger than the block size?
  Block cipher modes of operation:
  - Electronic Codebook (ECB)
  - Cipher Block Chaining (CBC)
  - Counter Mode (CTR)
  - (Many more modes)

http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

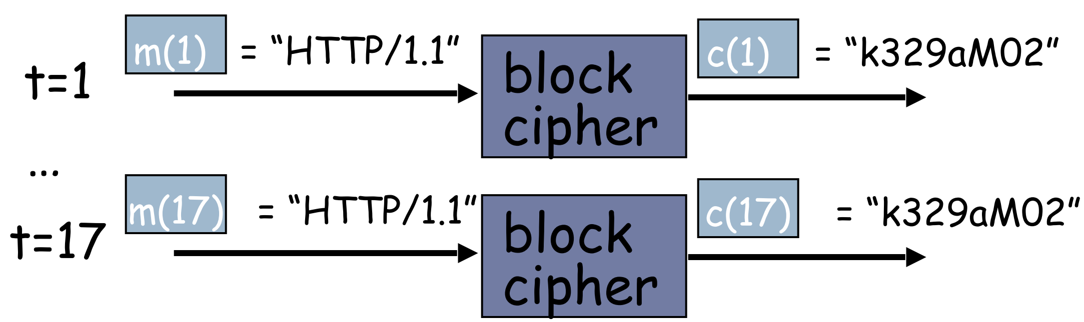
Electronic Codebook (ECB)

- Why not just break message in 128-bit blocks, encrypt each block separately with AES?



Electronic Codebook (ECB)

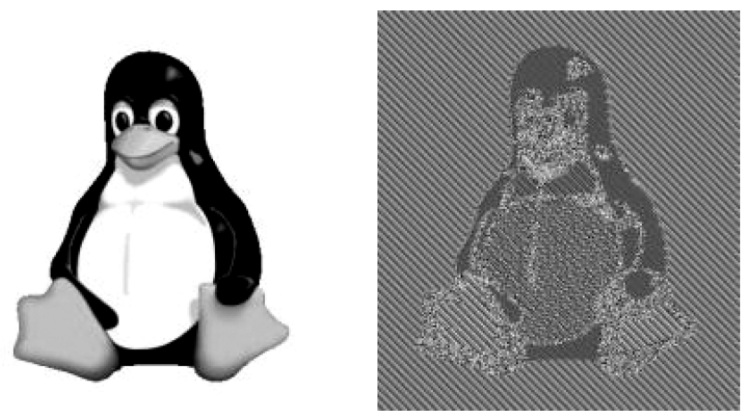
- Why not just break message in 128-bit blocks, encrypt each block separately with AES?
  - If same block of plaintext appears twice, will give same ciphertext
  - May facilitate cryptanalysis



Strengths and Weaknesses of ECB

- Strengths:
  - Is very simple
  - Parallel encryptions/ decryption of blocks
  - Can tolerate the loss or damage of a block

- Weakness:
  - Documents and images are not suitable for ECB encryption since patterns in the plaintext are repeated in the ciphertext:

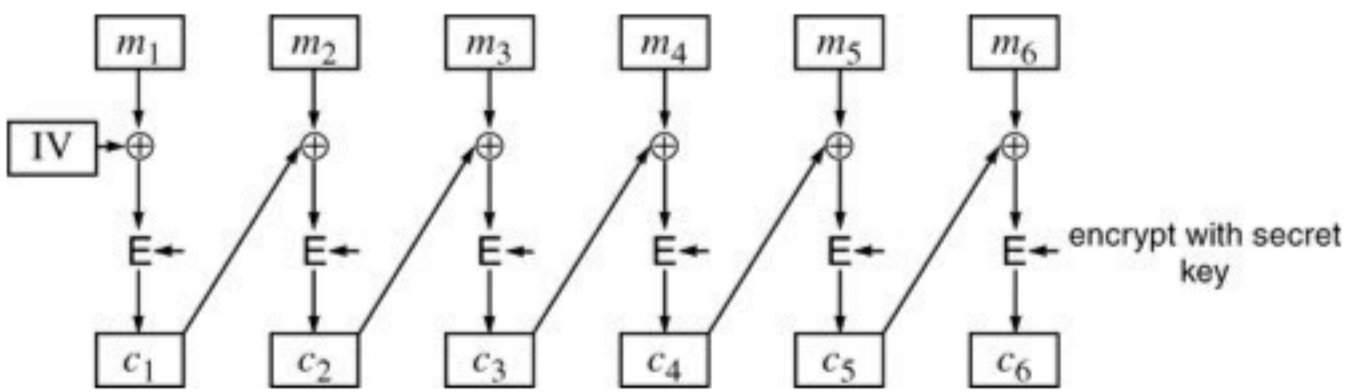


(a) (b)

Figure 8.6: How ECB mode can leave identifiable patterns in a sequence of blocks: (a) An image of Tux the penguin, the Linux mascot. (b) An encryption of the Tux image using ECB mode. (The image in (a) is by Larry Ewing, lewing@isc.tamu.edu, using The Gimp; the image in (b) is by Dr. Juzam. Both are used with permission via attribution.)

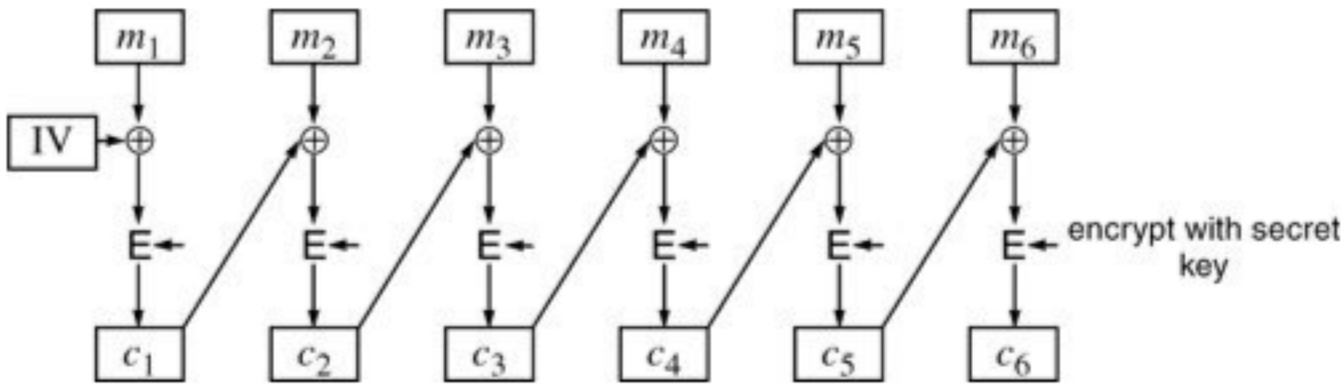
Cipher Block Chaining (CBC)

- CBC aims to "jumble" each input block before encryption
  - Use previous ciphertext block to XOR with current plaintext block
- How do we encrypt first block?
  - Initialization vector (IV): random block = c(0)
  - IV does not have to be secret, usually sent with the ciphertext
- Change IV for each message (or session)
  - Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time

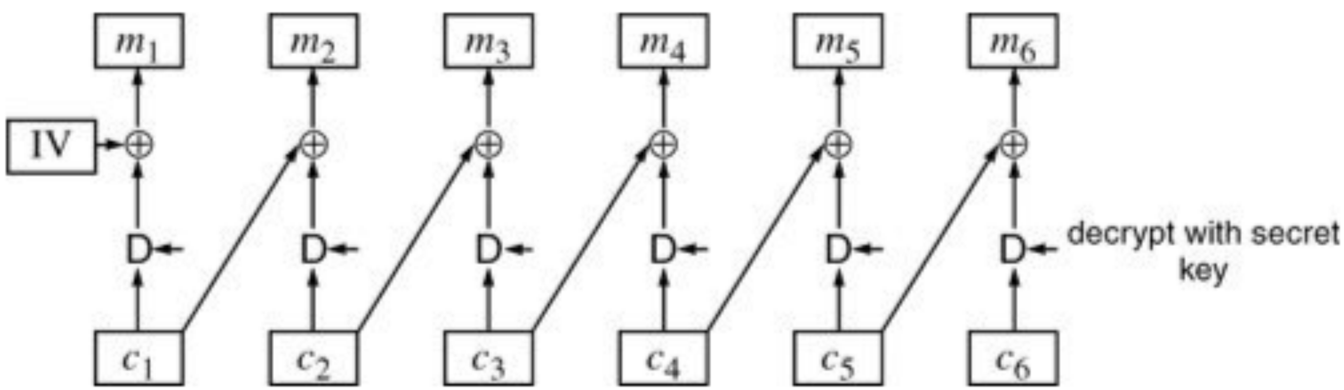


CBC Encryption and Decryption

CBC Encryption

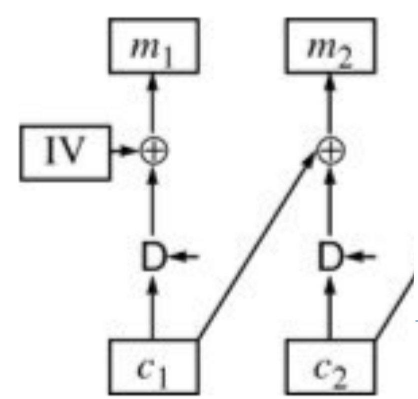


CBC Decryption



CBC Malleability

- CBC provides confidentiality but not integrity.
  - Normally, m_i = D_k(c_i) ⊕ c_(i-1)
  - What happens if an attacker intercepts the ciphertext and changes c_(i-1) to c'_(i-1)?
    - m'_i = D_k(c_i) ⊕ c'_(i-1)
    - = D_k(c_i) ⊕ 0 ⊕ c'_(i-1)
    - = D_k(c_i) ⊕ c_(i-1) ⊕ c_(i-1) ⊕ c'_(i-1)
    - = m_i ⊕ [c_(i-1) ⊕ c'_(i-1)]
  - So if attacker knows m_i and observes c_(i-1), they can control the decrypted value by changing it to c'_(i-1).

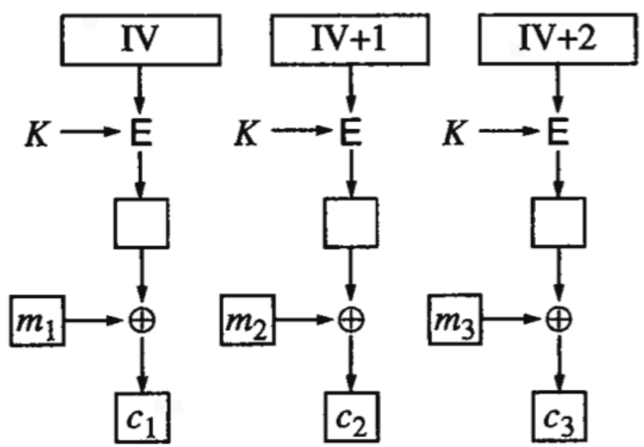


Strengths and Weaknesses of CBC

- Strengths:
  - Doesn’t show patterns in the plaintext
  - Is the most common mode
  - Is fast and relatively simple
  - Parallel decryption
- Weaknesses:
  - CBC requires the reliable transmission of all the blocks sequentially (may not be suitable for high packet-loss traffic, like video/music streaming)
  - Existence of threats (e.g., malleability)

Counter Mode (CTR)

- Encrypts increments of IV to generate keystream
- Advantages:
  - Decryption can start anywhere, as long as you know the block number you are considering
  - Useful in case of encrypted random access files, for example



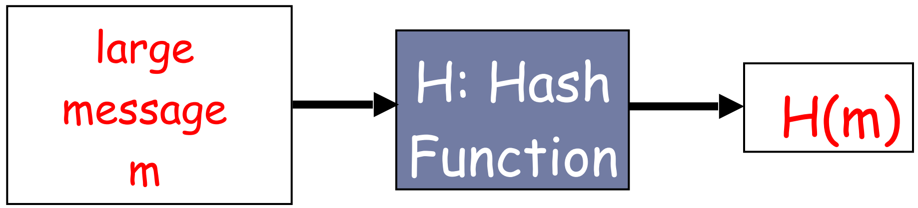
Hashes and Message Digests

Message Integrity

- Allows communicating parties to verify that received messages have not been tampered with.

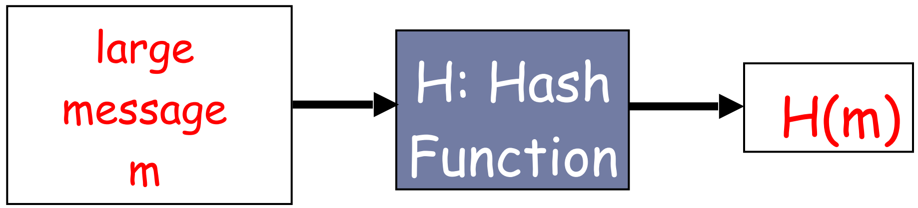
- Approach: "Summarize" the message in a short way that can be verified given the full message, and is hard to spoof (e.g., create a different message with the same summary). This summary is the *message digest*.

Message Digest Building Block



- A hash function H maps a plaintext P to a fixed-length value H(P) called hash value or message digest of P
- A collision is a pair of plaintexts P and Q that map to the same hash value, h(P) = h(Q)
  - Collisions are unavoidable (**Why?**)

Message Digest Building Block



- A hash function H maps a plaintext P to a fixed-length value H(P) called hash value or message digest of P
- A collision is a pair of plaintexts P and Q that map to the same hash value, h(P) = h(Q)
  - Collisions are unavoidable (**Why? Pigeon Hole Principal**)

Cryptographic Hash Functions

- A cryptographic hash function satisfies additional properties
  - Preimage resistance (aka one-way)
    - Given hash value x, it is hard to find plaintext P such that h(P) = x
  - Second preimage resistance (aka weak collision resistance)
    - Given plaintext P, it is hard to find plaintext Q such that h(Q) = h(P)
  - Collision resistance (aka strong collision resistance)
    - It is hard to find a pair of plaintexts P and Q such that h(Q) = h(P)
- Collision resistance implies second preimage resistance

Checksum: Poor Message Digest

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of input
- is many-to-one

- But given message with given hash value, it is easy to find another message with same hash value.
- Example: Simplified checksum: add 4-byte chunks at a time:

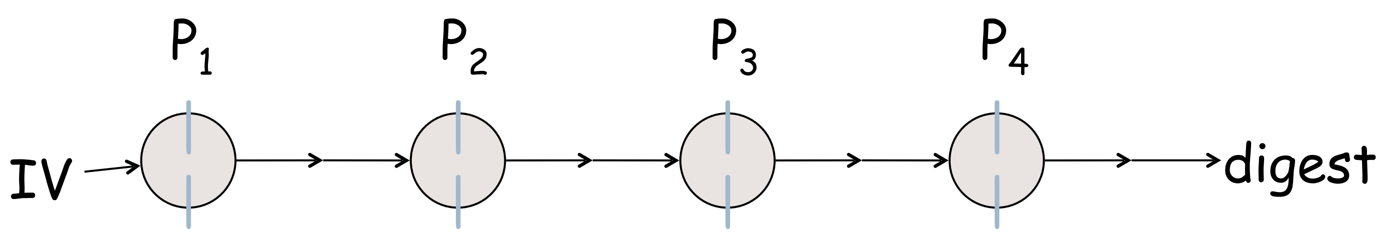
message				ASCII format			
I	O	U	1	49	4F	55	31
0	0	.	9	30	30	2E	39
9	B	O	B	39	42	D2	42
				B2	C1	D2	AC

message				ASCII format			
I	O	U	9	49	4F	55	39
0	0	.	1	30	30	2E	31
9	B	O	B	39	42	D2	42
				B2	C1	D2	AC

different messages but identical checksums!

Iterated Hash Function Design

- A compression function works on input values of fixed length
  - Inputs: X,Y with len(X)=m, len(Y)=n; Output: Z with len(Z)=n
- An iterated hash function extends a compression function to inputs of arbitrary length
  - padding, initialization vector (although not random in this case), and chain of compression functions
  - inherits collision resistance of compression function
- Also known as the Merkle–Damgård construction
- MD5 and SHA are two popular iterated hash functions



Message-Digest Algorithm 5 (MD5)

- Developed by Ron Rivest in 1991
- Uses 128-bit hash values
- Still widely used in legacy applications although considered insecure
- Various severe vulnerabilities discovered
  - Example: Chosen-prefix collisions attacks found by Marc Stevens, Arjen Lenstra and Benne de Weger
    - Start with two arbitrary plaintexts P and Q
    - One can compute suffixes S1 and S2 such that P||S1 and Q|| S2 collide under MD5 by making 250 hash evaluations
    - **Using this approach, a pair of different executable files or PDF documents with the same MD5 hash can be easily computed**

Secure Hash Algorithm (SHA)

- Developed by NSA and approved as a federal standard by NIST
- SHA-0 and SHA-1 (1993)
  - 160-bits output
  - Considered insecure (although attackers are still very complex and computational expensive, so still more secure than MD5)
  - Still found in legacy applications
- SHA-2 family (2002)
  - 256 bits (SHA-256) or 512 bits (SHA-512)
  - Still considered secure in practice despite theoretical attack techniques
- Public competition for SHA-3 announced in 2007, decided in 2015 (but adoption is slow)

Providing Message Integrity

<u>Problem Statement</u>

- Assume we want to send a message and are *only* concerned with integrity
  - Not concerned with authentication or confidentiality.

<u>Proposed Solution</u>

- What if we send the following?
  - m' = m || SHA256(m)
  - The receiver can extract m, compute SHA256(m), and check if this matches the SHA256 that was sent
- Does this guarantee integrity?

Message Authentication Code (MAC)

- Designed to provide both *authentication* and *integrity*.
- How can we use hash functions to compute a MAC?
  - *H* is a hash function
  - *m* is a message
  - *K* is a secret key.
- Let's talk about different constructions
  - Secret Prefix Construction => $H(K \| m)$
  - Secret Suffix Construction => $H(m \| K)$
  - HMAC => $H((K \oplus opad) \| H((K \oplus ipad) \| m))$

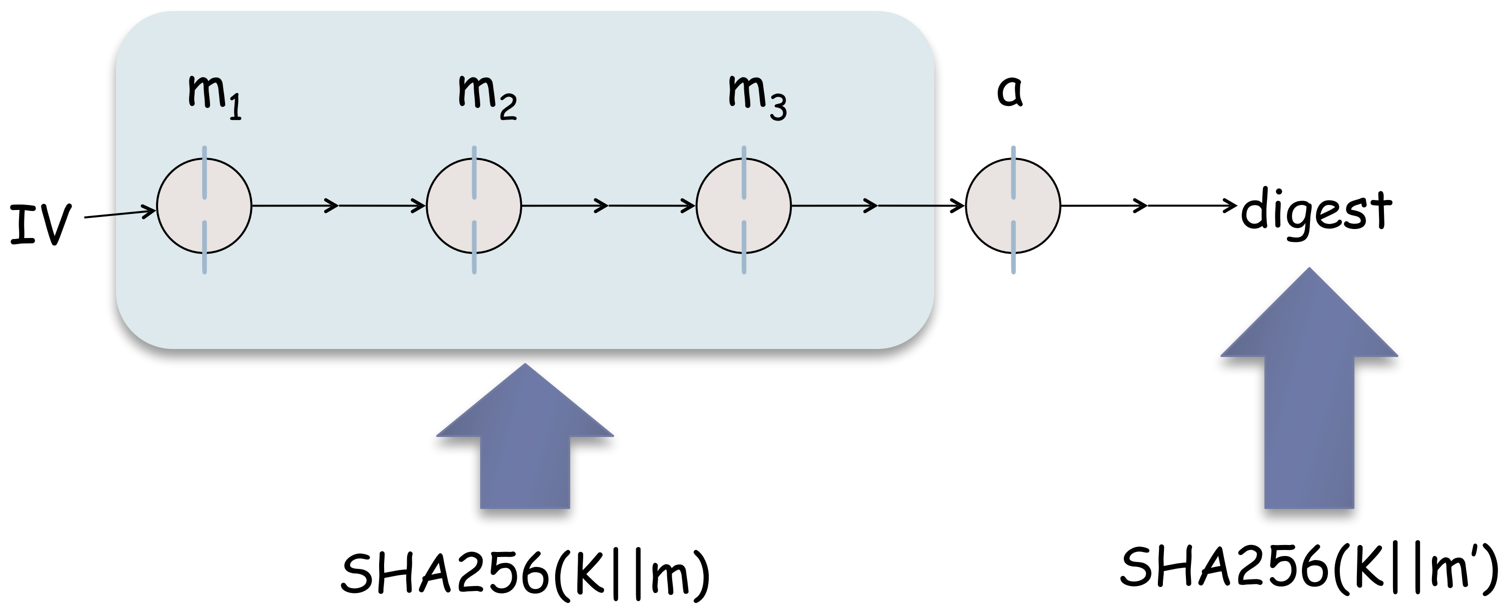
Secret Prefix Construction

$$MAC = H(K \mathbin{\|} m)$$

H = cryptographic hash function
K = secret key
m = message to send

Uh oh! Length Extension Attack!

- Because most hash functions are iterated hash functions
  - Attacker knows the message m and H(K || m)
  - They could append something to m to get m' = m || a, and use H(K || m) to initialize the computation of H(K || m')



Secret Suffix Construction

$$MAC = H(m \| K)$$

H = cryptographic hash function

K = secret key

m = message to send

Better! But if H has weaknesses, MAC isn't as secure.

HMAC

Inner Hash

$$\mathrm{HMAC}(K, m) = \mathrm{H}\Big((K' \oplus opad) \,\|\, \boxed{\mathrm{H}\big((K' \oplus ipad) \,\|\, m\big)}\Big)$$

Outer Pad

Inner Pad

<u>Short Summary</u>

1. Two rounds of hashing (inner and outer)
2. Inner Hash
   - XORS key with *ipad* create inner pad
   - Prepends inner pad to message and hashes
3. Outer Hash
   - XOR key with *opad* to create outer pad
   - Prepends outer pad to inner hash, and hash one more time

HMAC

Inner Hash

$$\mathrm{HMAC}(K,m) = \mathrm{H}\Big((K' \oplus opad) \,\|\, \boxed{\mathrm{H}\big((K' \oplus ipad) \,\|\, m\big)}\Big)$$

Outer Pad

Inner Pad

Recommended!

Even if H is weaker, HMAC remains much more secure than other schemes.

(Significantly harder to forge HMAC compared to secret suffix construction).