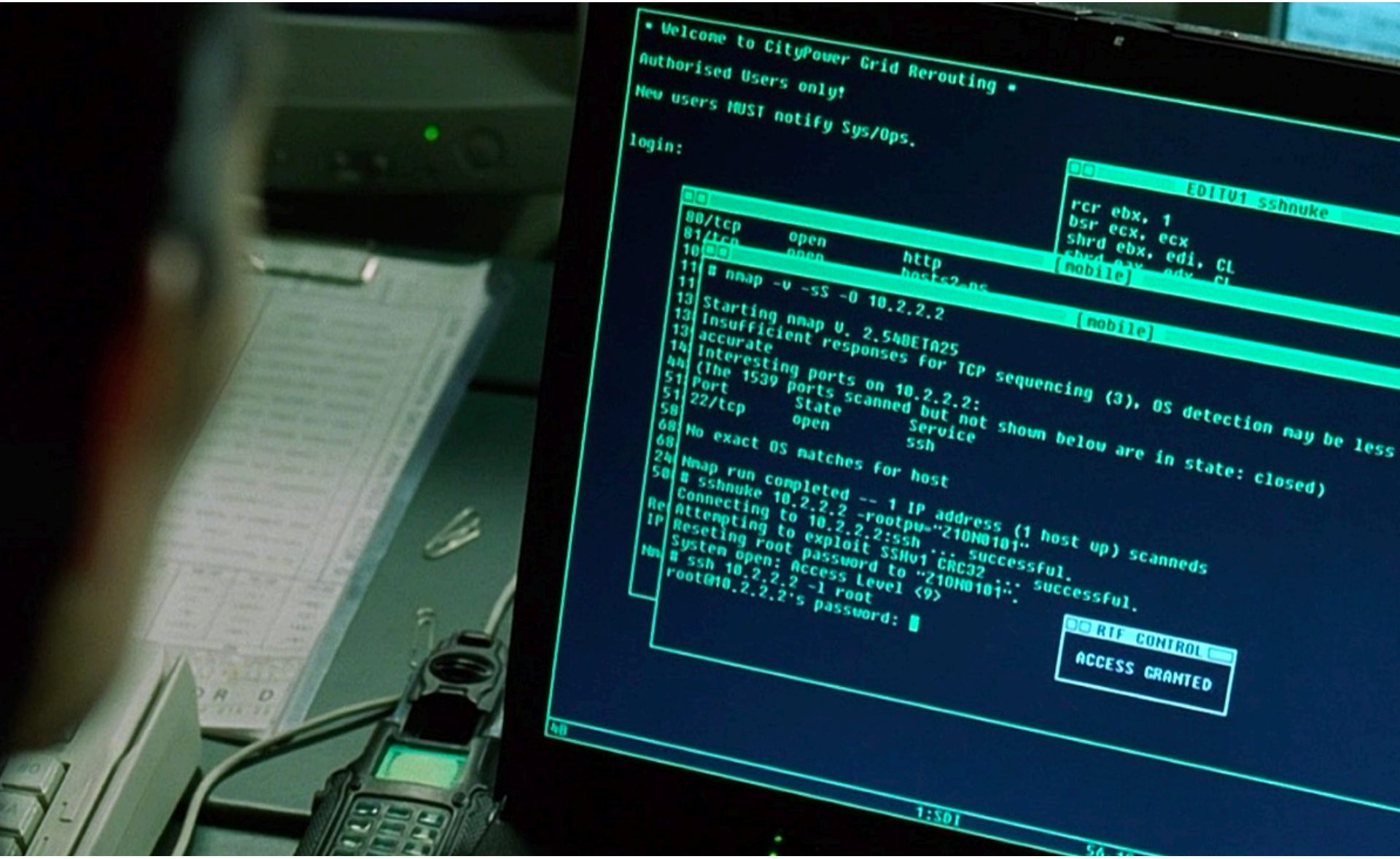


* Slides adopted from Prof. Manos Antonakakis

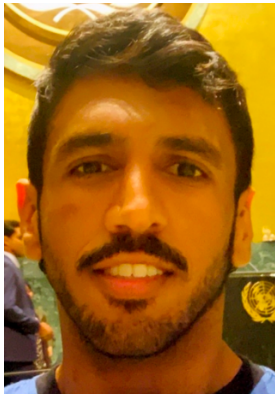
Office Hours

OHs will be on Zoom (in-person via request). Times listed are Eastern Time.

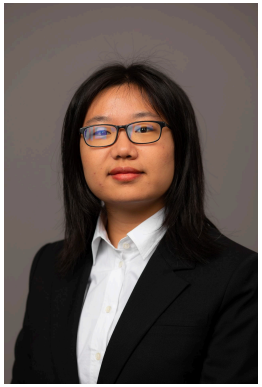
Class TAs:

Alicia
Mon@3pm

Qinge
Wed@9am



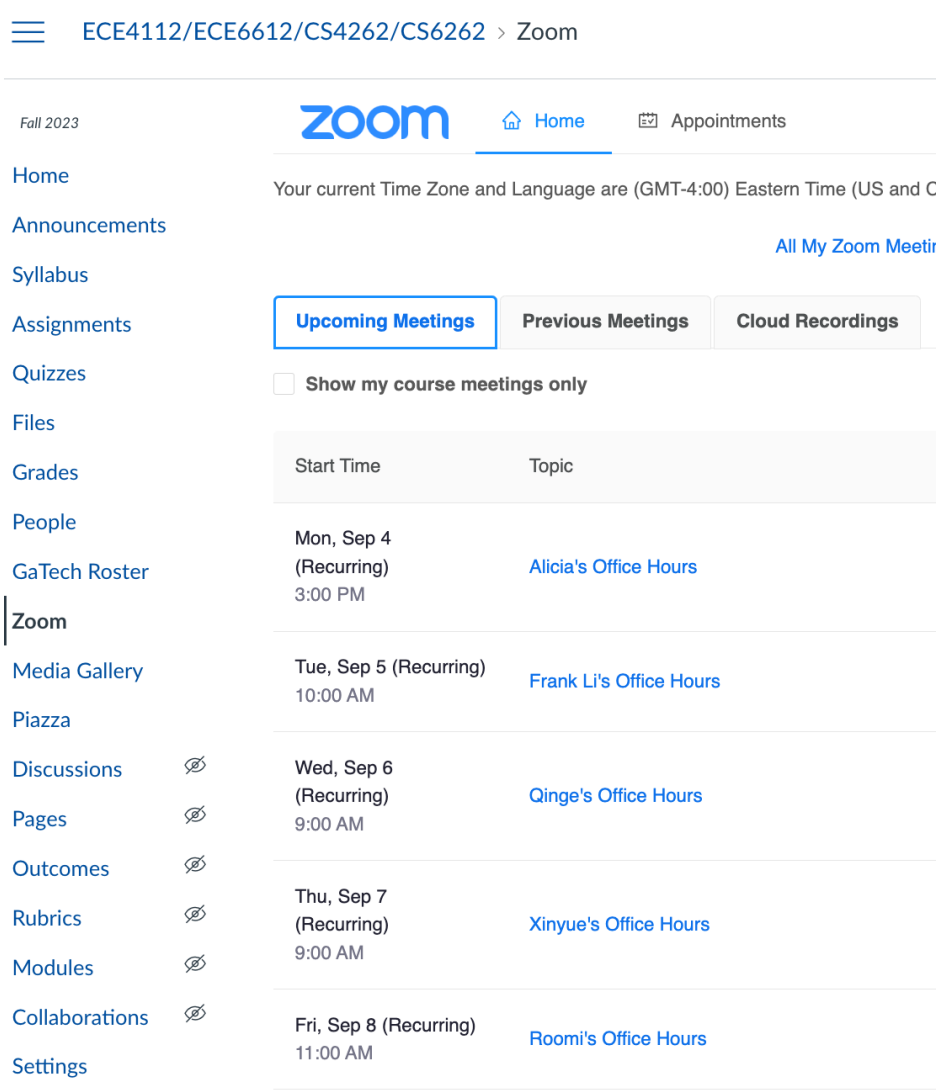
Roomi
Fri@11am



Xinyue
Thu@9am

Frank: Tue@10am

Office Hours



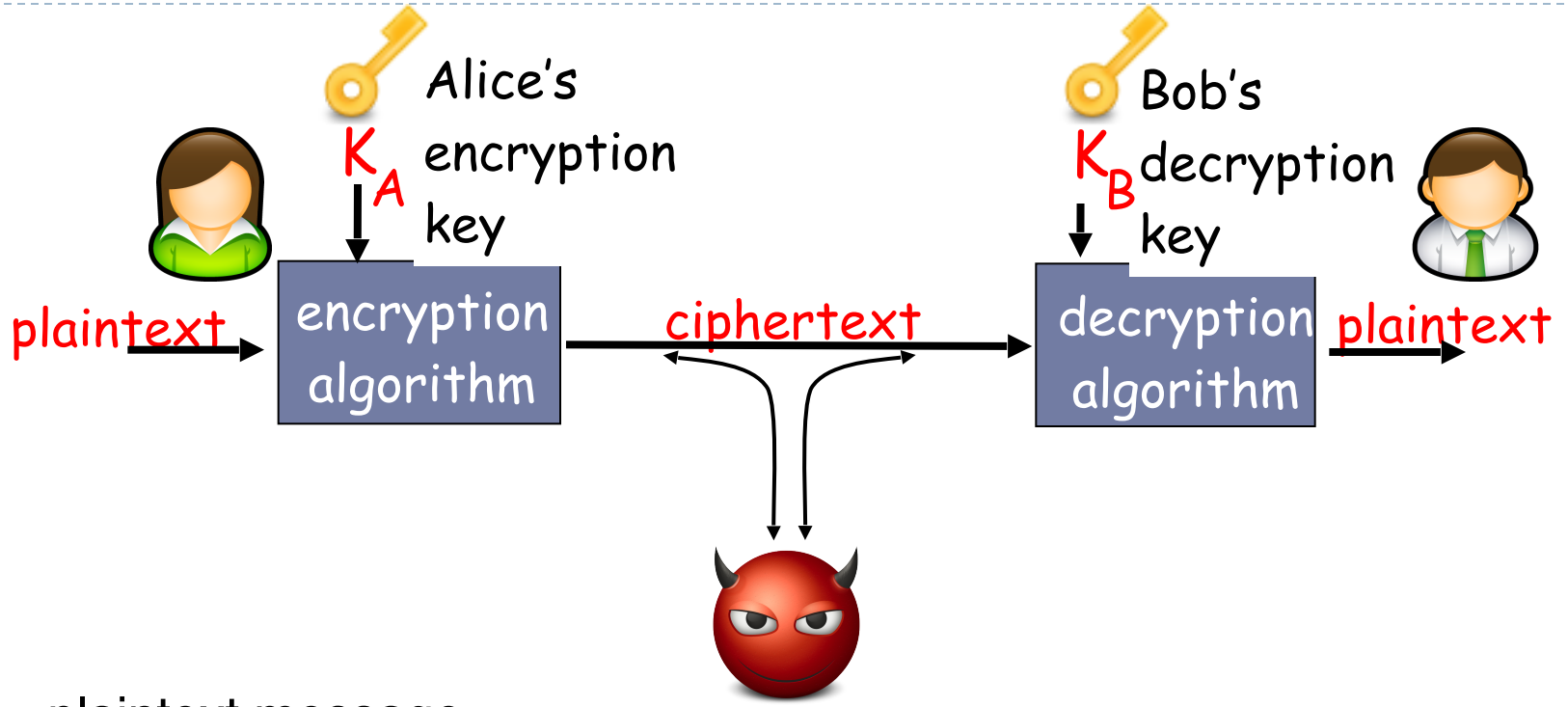
Course Logistics

- HW1 released (on cryptography), due Sept 11
    - Submissions auto-graded, following instructions carefully (and ask if you're unsure)!!
- Project proposal assigment released, due Sept 29
    - Teams: 3-5 students
    - Proposal: 1 page only, **non-binding**, see assignment for what to include
    - Open-ended: could be developing attacks, defenses, tools, or measuring netsec in the real-world

Today's Class: Symmetric Crypto

Date	Session Topic
Tue, Aug 22	Course Overview + Logistics
Thu, Aug 24	Network Protocols Overview
Tue, Aug 29	Cryptography: Symmetric Crypto
Thu, Aug 31	Cryptography: Hash + MACs
Tue, Sept 5	Cryptography: Public-Key Crypto

The language of cryptography



m = plaintext message

c = $Enc(K_A,m)$ = ciphertext, encrypted with key K_A

$Dec(K_B, c) = Dec(K_B, Enc(K_A,m)) => m$

Cryptography vs. Cryptanalysis

- Cryptographers invent new clever cryptographic schemes
  - Objective: make it infeasible to recover the plaintext
    - Computational difficulty: efficient to compute ciphertext, but hard to "reverse" without the key
- Cryptanalysis studies cryptographic schemes
  - Objective: try to find flaws in the schemes
    - E.g., recover some info about the plaintext, or recover the key
- **Fundamental Tenet of Cryptography**
  - ***"If lots of smart people have failed to solve a problem, then it probably won't be solved (soon)"***

Main Cryptographic Tools

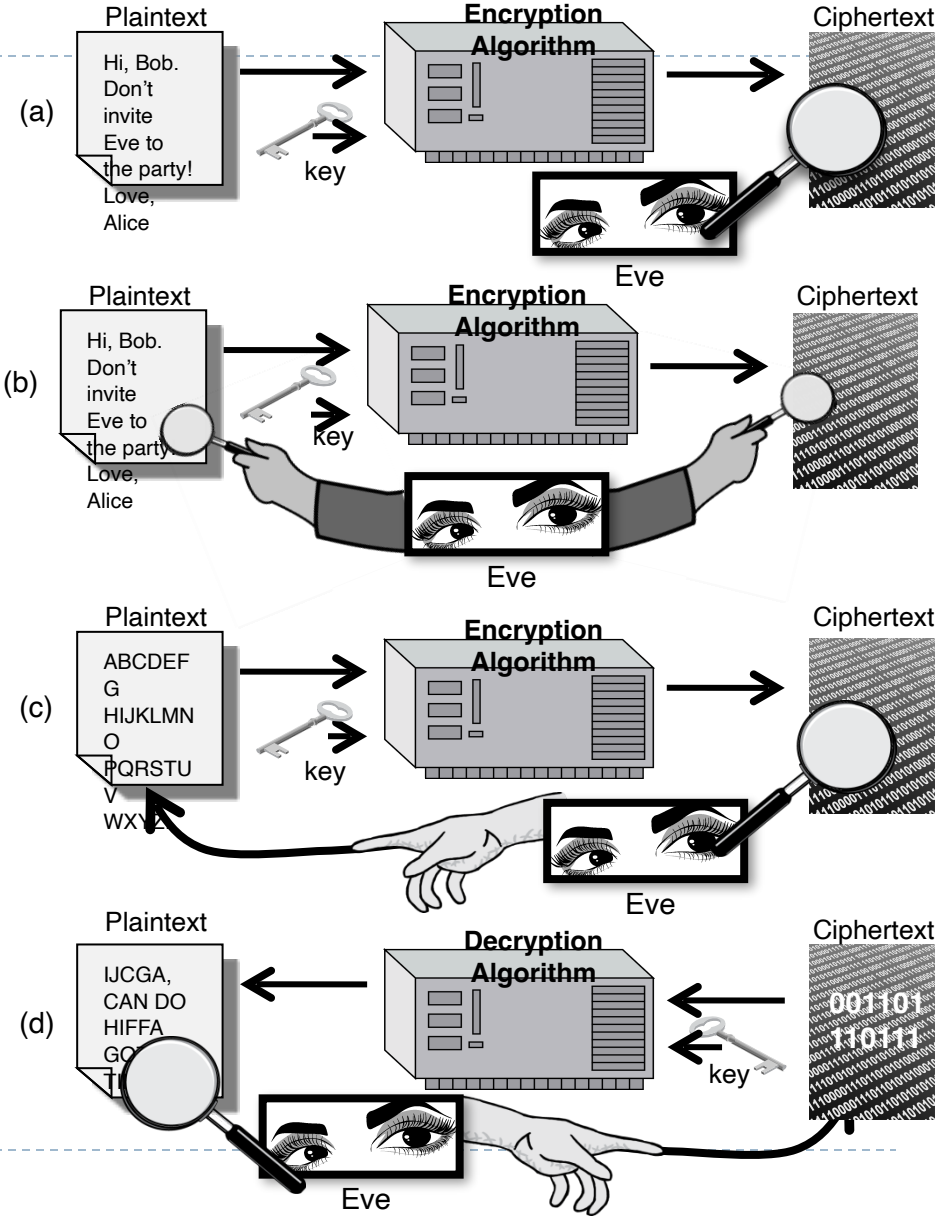
- Cryptography often uses keys:
  - Algorithm is known to everyone (Kerckhoff's principle)
  - Only “keys” are secret
- Symmetric key cryptography (today)
  - Involves one key, must be secret
- Public key cryptography (next week)
  - Involves two keys, one secret and one public
- Hash functions (next class)
  - No keys involved
  - Nothing secret but input should be hard to determine given the output

Randomness and Random Number Gen.

- The ciphertext needs to seemingly be random:
  - This means that, assuming you do not have the secret key, you cannot make sense of the ciphertext AND
  - The mappings between an input value and the output value appears to have been generated by a random number generator (RNG)
- Conceptual mapping scheme:
  - To get the mapping from input *in* to output *out* (of some bit-length *L*), flip a coin *L* times to get a random value for *out*.
  - Since the mapping must be **one-to-one**, if the *out* value already exists in the mapping, start all over again. Otherwise, save the mapping of *in->out.*
  - Mapping must be tied to the secret key. So every key will have a different mapping (and without the key, can't figure out *in* just given *out)*

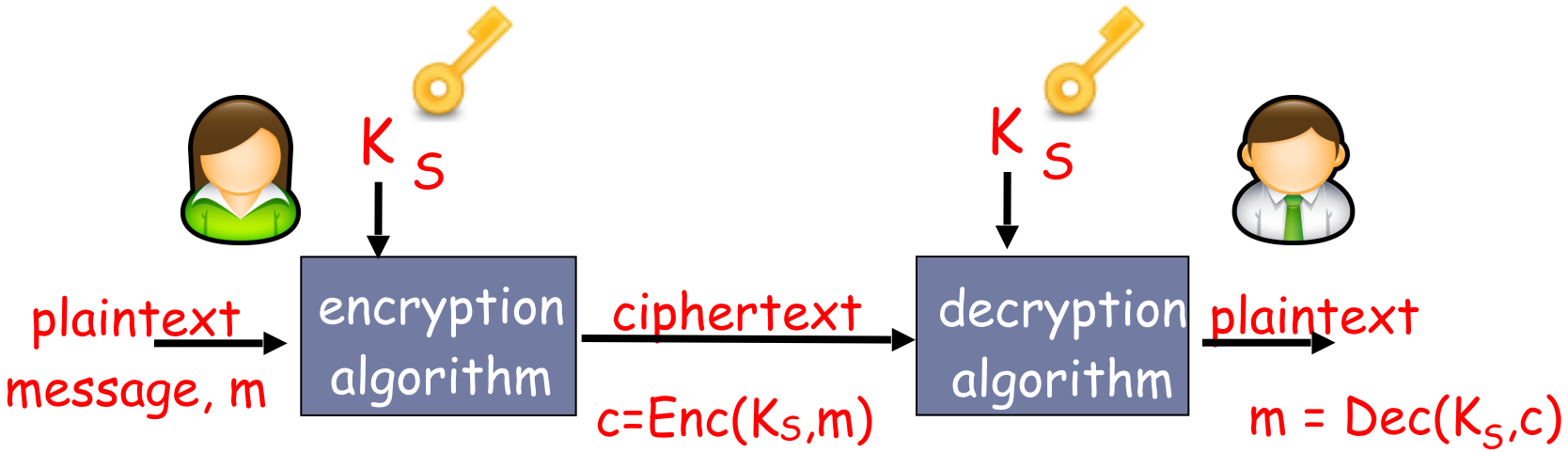
Cryptography Threat Models

- Attacker may have
  a) collection of ciphertexts (ciphertext only attack)
  b) collection of plaintext/ciphertext pairs (known plaintext attack)
  c) collection of plaintext/ciphertext pairs for plaintexts selected by the attacker (chosen plaintext attack)
  d) collection of plaintext/ciphertext pairs for ciphertexts selected by the attacker (chosen ciphertext attack)



Symmetric Cryptography

Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K

Q: how do Bob and Alice agree on key value?

Two types of symmetric ciphers

- Stream ciphers
  - encrypt one bit at time
- Block ciphers
  - Break plaintext message in equal-size blocks
  - Encrypt each block as a unit

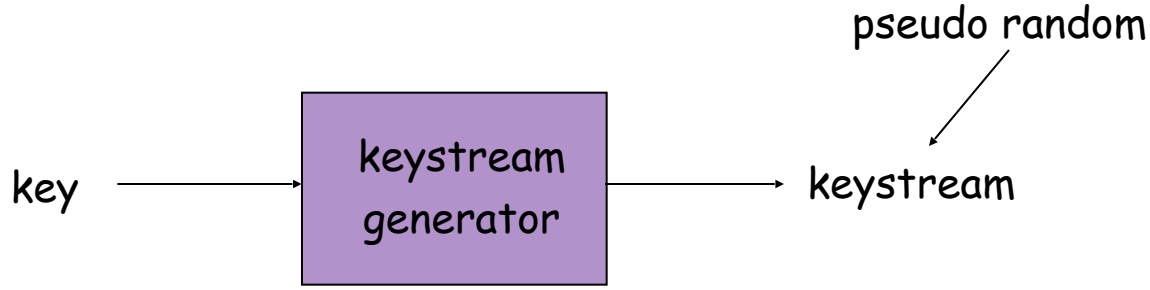
Understanding Exclusive-Or (XOR)

- Most cryptographic algorithms rely on performing XOR operations of random bits against some plaintext bit string.
- XOR operations are very fast
- The symbol for XOR is ⊕
- Key properties:
  - P ⊕ P = 0 and P ⊕ 0 = P
  - Associative: P ⊕ (P' ⊕ P'') = (P ⊕ P') ⊕ P''
  - Commutative: P ⊕ P' = P' ⊕ P
  - *As a result: P ⊕ (P' ⊕ P) = P'*

Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

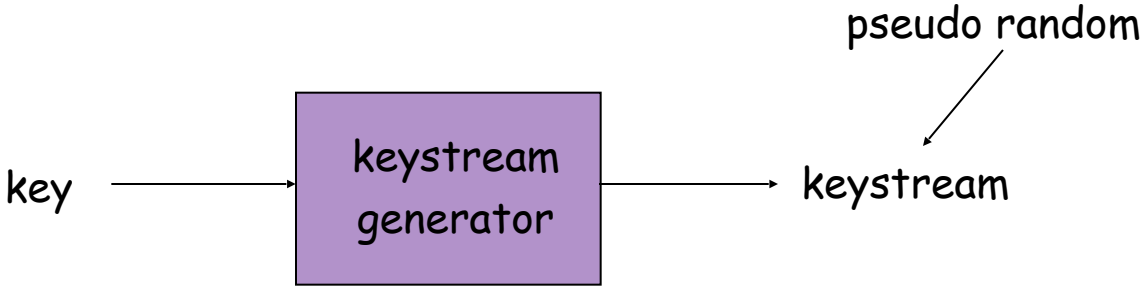
Truth table for XOR of two inputs.

Stream Ciphers



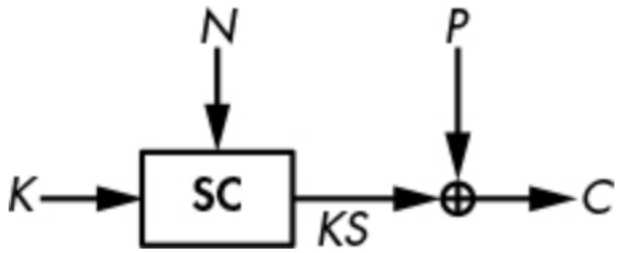
- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
  - m(i) = ith bit of message
  - $k_s(i)$ = ith bit of keystream
  - c(i) = ith bit of ciphertext
  - $c(i) = k_s(i) \oplus m(i)$ ($\oplus$ = exclusive or)
  - $m(i) = k_s(i) \oplus c(i)$

Stream Ciphers



- Problem:
  - If attacker knows portion of plaintext P for ciphertext C, they can change C so it decrypts to a desired malicious plaintext P'. How?
  - $C = KS \oplus P$
  - Attacker: $C' = C \oplus (P \oplus P') = (KS \oplus P) \oplus (P \oplus P') = KS \oplus P'$
  - Key lesson: Encryption provides confidentiality, but not integrity.

Stream Ciphers in Practice



How stream ciphers encrypt, taking a secret key, K, and a public nonce, N

- KS = SC(K, N) produces keystream from *key* and *nonce*.
- *Never reuse the same (K, N) pair to encrypt different plaintexts.*

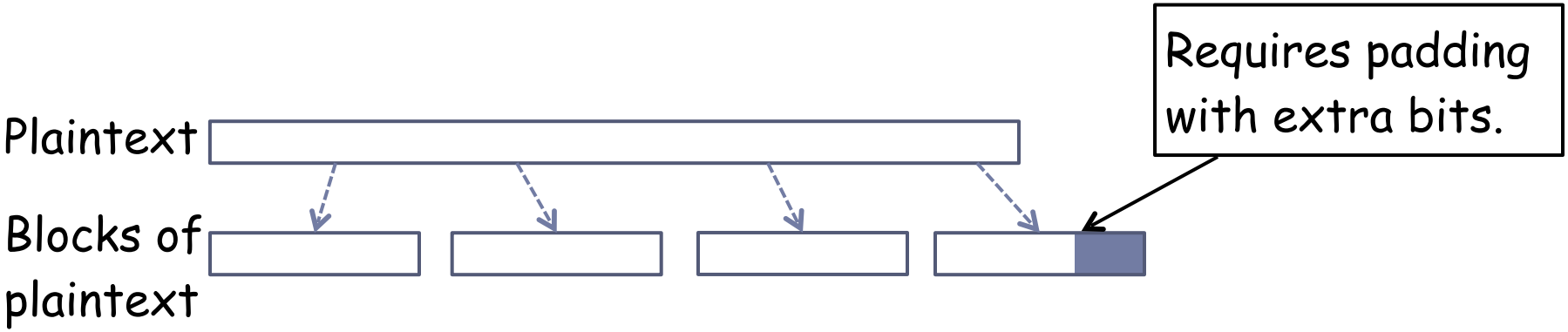
RC4 Stream Cipher

- RC4 is a popular stream cipher
  - Extensively analyzed and *no longer considered good*
  - Key can be from 1 to 256 bytes
  - Still used in practice in 802.11 (WiFi) encryption and SSL.
  - Better schemes today: ChaCha

```
void
rc4_crypt(struct rc4_state *const state,
        const u_char *inbuf, u_char *outbuf, int buflen)
{
        int i;
        u_char j;

        for (i = 0; i < buflen; i++) {

                /* Update modification indicies */
                state->index1++;
                state->index2 += state->perm[state->index1];

                /* Modify permutation */
                swap_bytes(&state->perm[state->index1],
                    &state->perm[state->index2]);

                /* Encrypt/decrypt next byte */
                j = state->perm[state->index1] + state->perm[state->index2];
                outbuf[i] = inbuf[i] ^ state->perm[j];
        }
}
```

Block Ciphers

- In a **block cipher:**
  - Plaintext and ciphertext have fixed length b (e.g., 128 bits)
  - A plaintext of length n is partitioned into a sequence of m **blocks**, P[0], …, P[m–1], where $n \le bm < n + b$
- Each message is divided into a sequence of blocks and encrypted or decrypted in terms of its blocks.



Block ciphers

- Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

Example with k=3:

input	output
000	110
001	111
010	101
011	100

input	output
100	011
101	010
110	000
111	001

What is the ciphertext for 010110001111 ?

Block ciphers

- Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

Example with k=3:

input	output
000	110
001	111
010	101
011	100

input	output
100	011
101	010
110	000
111	001

What is the ciphertext for 010|110|001|111 ?

101|000|111|001

Choosing Block Sizes

- Security of block cipher depends on **both** block size and key size.
- Blocks *shouldn't be too large* to minimize length of ciphertext and memory footprint.
  - Smaller block sizes are able to fit into the registers of most CPUs.
- Blocks also *shouldn't be too small* or they're susceptible to codebook attacks (i.e., build a lookup table of all possible input/output pairs).
  - These are attacks against block ciphers that are only efficient when smaller blocks are used.

The Codebook Attack

- The codebook attack works like this with 16-bit blocks:
  - Get the 65536 (2^{16}) ciphertexts corresponding to each 16-bit plaintext block.
  - Build a lookup table—the *codebook*—mapping each ciphertext block to its corresponding plaintext block.
  - To decrypt an unknown ciphertext block, look up its corresponding plaintext block in the table.
- When 16-bit blocks are used, the lookup table needs only $2^{16} \times 16 = 2^{20}$ bits of memory, or 128 kilobytes.
- With 32-bit blocks, memory needs grow to 16 gigabytes, which is still manageable.
- But with 64-bit blocks, you'd have to store 2^{70} bits (a zetabit, or 128 exabytes), so forget about it.
- Codebook attacks won't be an issue for larger blocks.

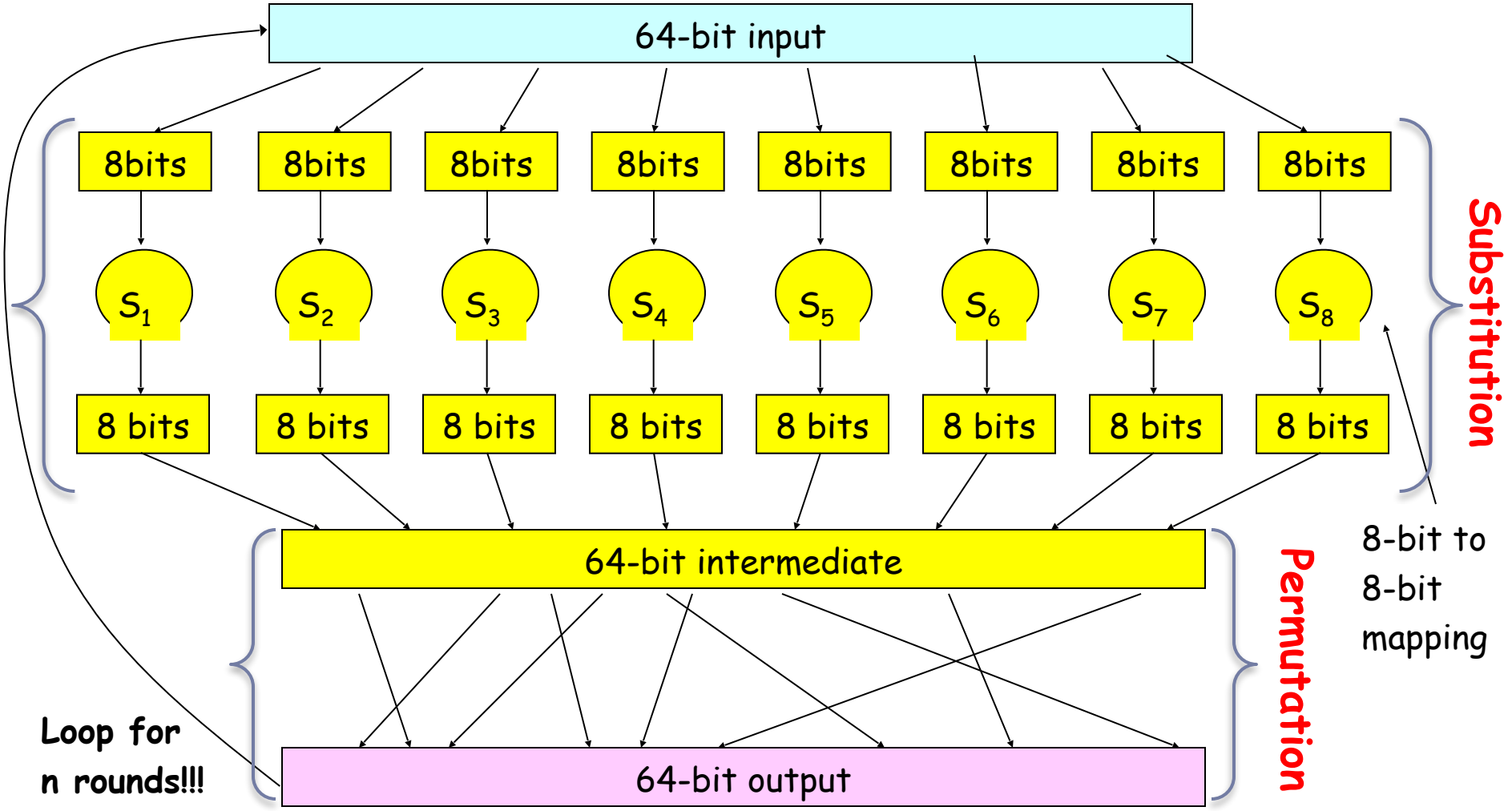
Padding

- Block ciphers require the length n of the plaintext to be a multiple of the block size b. So pad the last block of the plaintext to size b.
- Padding must be reversible/unambiguous (i.e., what's the original plaintext vs padding)
- Example for b = 128 (16 bytes)
  - Plaintext: “Thinker” (7 bytes)
  - Idea #1: "Thinker000000000". Is it good or bad, and why?
  - Idea #2: “Thinker999999999” (16 bytes), where 9 denotes the number and not the character
    - This is essentially what's done for two similar real-world schemes, PKCS#5 and PKCS#7.
- We need to always pad the last block, which may consist only of padding

What do block functions look like?

From Kaufman et al

What do block functions look like?



Why rounds?

- More rounds = more randomness (sort of like shuffling cards)
  - If only a single round, then one bit of input affects at most 8 bits of output.
  - In 2nd round, the 8 affected bits get scattered and inputted into multiple substitution boxes.

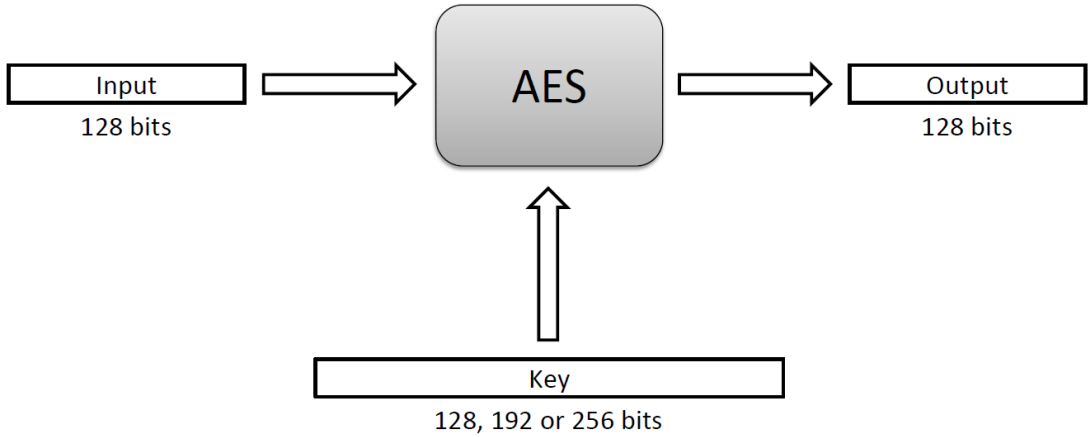
- How many rounds?
  - More rounds = more security but higher encryption/ decryption runtime
  - In practice, usually 10-14 rounds

AES: Advanced Encryption Standard

- In 1997, the U.S. National Institute for Standards and Technology (NIST) put out a public call for a replacement to DES, an older (now insecure) algorithm.
- It ultimately chose an algorithm that is now known as the **Advanced Encryption Standard** (**AES**).
- New (Nov. 2001) symmetric-key NIST standard, replacing DES
  - **Nice mathematical justification for design choices**
- Processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- Brute force decryption (try each key) takes ~1 sec on DES, takes 149 trillion years for AES

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.

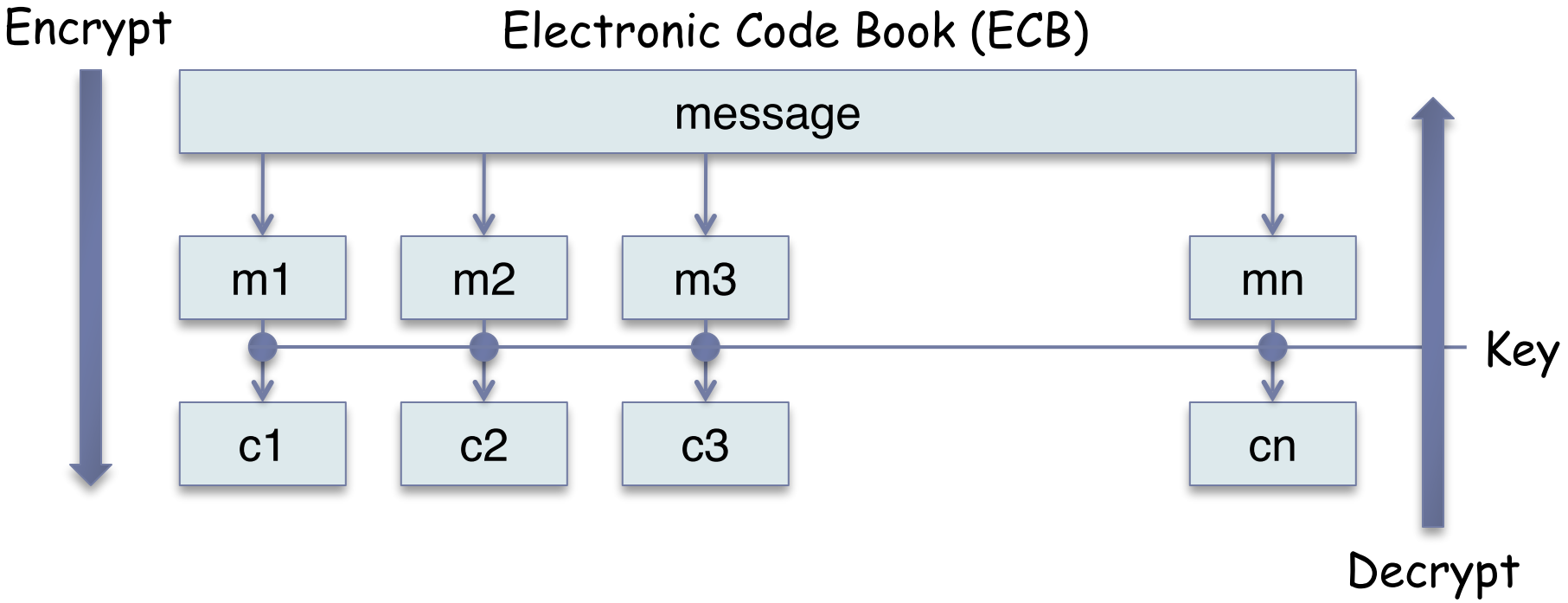


Encrypting Large Messages

- Block ciphers only operate on messages of a fixed size (e.g., 64-bit, 128-bit, 192-bit, 256-bit, etc)
- How do you encrypt messages larger than the block size?
  - Use one of the modes of operation defined in the cryptographic standards.
- Padding Modes of Operation
  - Electronic Codebook (ECB)
  - Cipher Block Chaining (CBC)
- Stream Cipher Modes of Operation
  - Output Feedback Mode (OFB)
  - Cipher Feedback Mode (CFB)
  - Counter Mode (CTR)

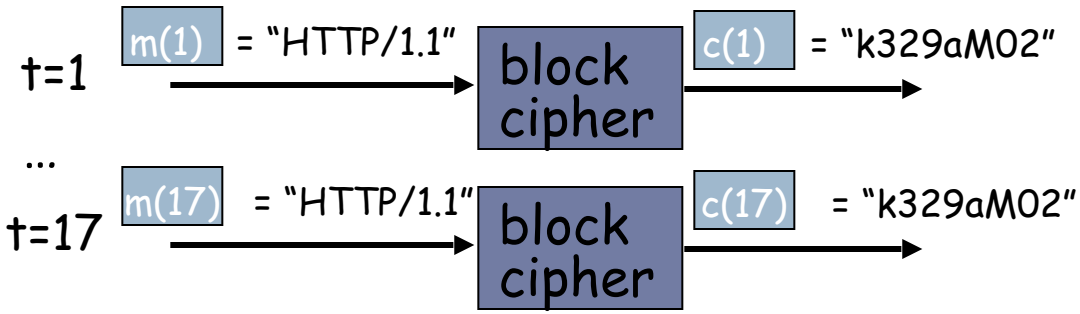
Electronic Codebook (ECB)

- Why not just break message in 64-bit blocks, encrypt each block separately?



Electronic Codebook (ECB)

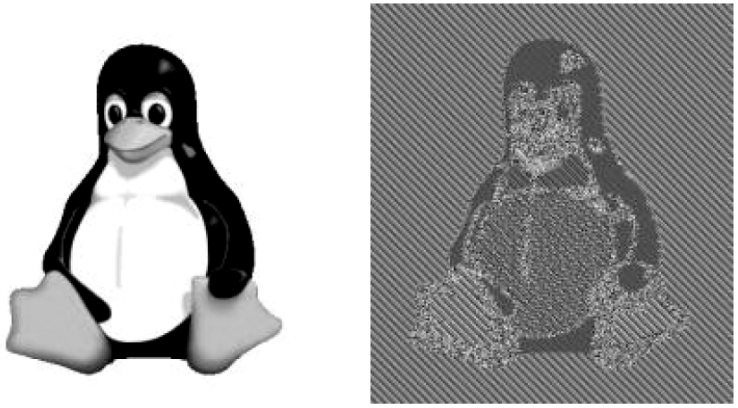
- Why not just break message in 64-bit blocks, encrypt each block separately?
  - If same block of plaintext appears twice, will give same ciphertext
  - May facilitate cryptanalysis
  - we could swap things (e.g., swap salaries)



Strengths and Weaknesses of ECB

- Strengths:
  - Is very simple
  - Allows for parallel encryptions of the blocks of a plaintext
  - Can tolerate the loss or damage of a block

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

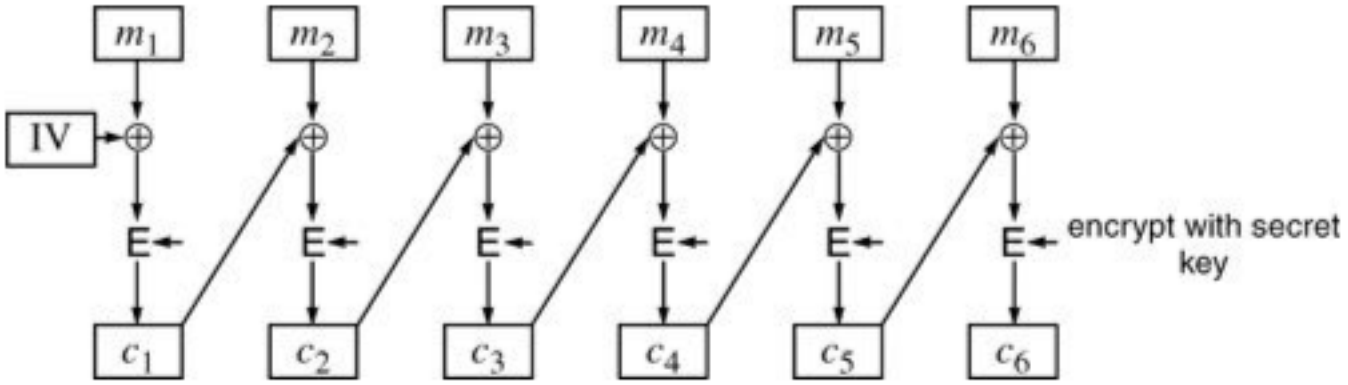


(a) (b)

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.)

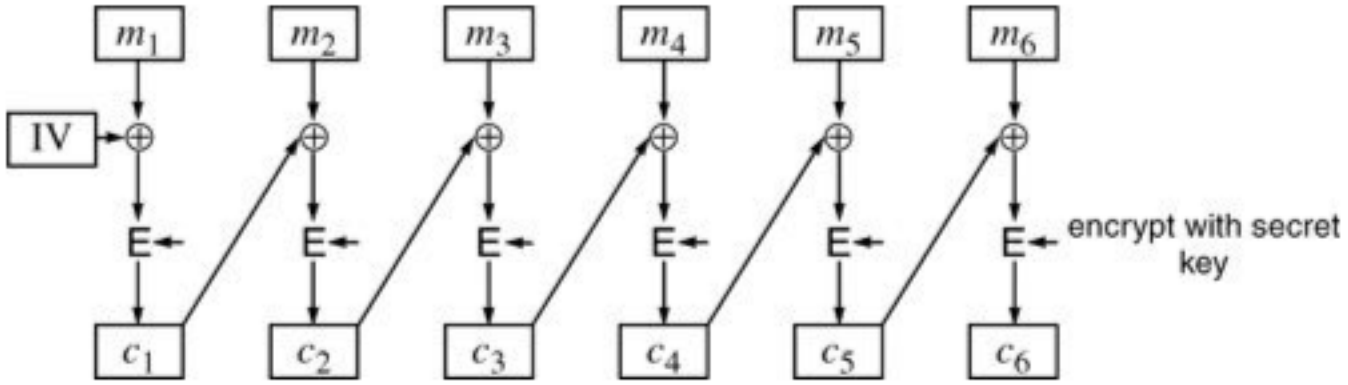
Cipher Block Chaining (CBC)

- 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
- 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

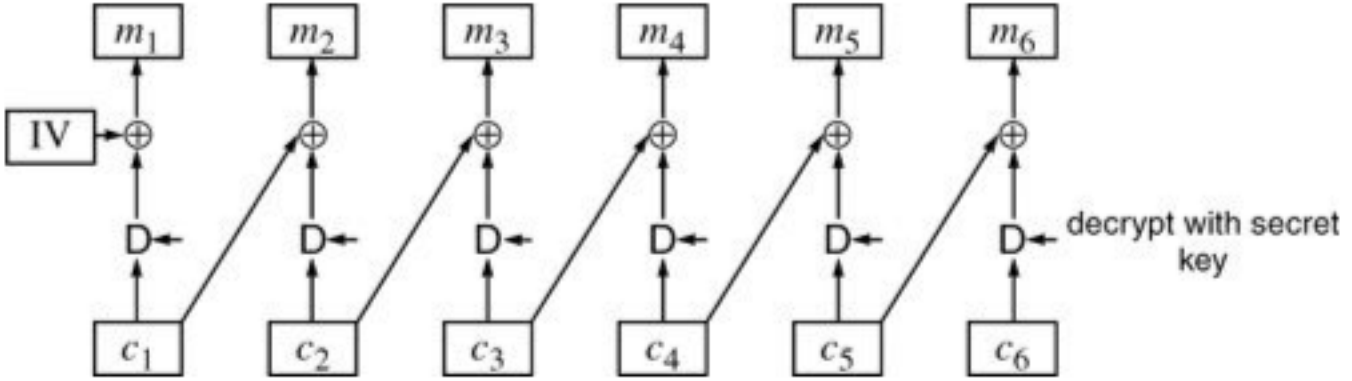


CBC Encryption and Decryption

CBC Encryption



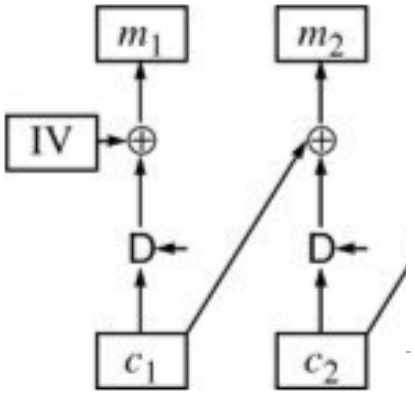
CBC Decryption



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) ⊕ c_(i-1) ⊕ c_(i-1) ⊕ c'_(i-1)
    - = m_i ⊕ [c_(i-1) ⊕ (c'_(i-1)]
  - Known plaintext attack: if attacker knows m_i and observes c_(i-1), they can control the decrypted value by changing it to c'_(i-1).

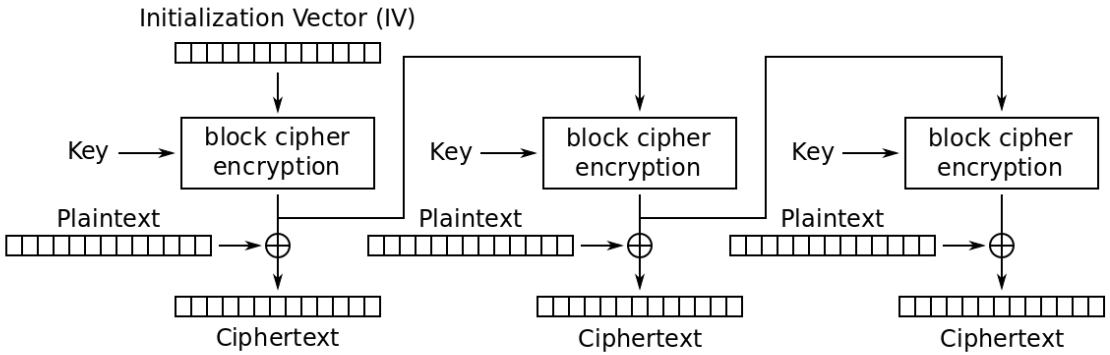
- Defenses?
  - Attach checksum block to the plaintext before encrypting (assumes attacker doesn't know entire plaintext message)



Strengths and Weaknesses of CBC

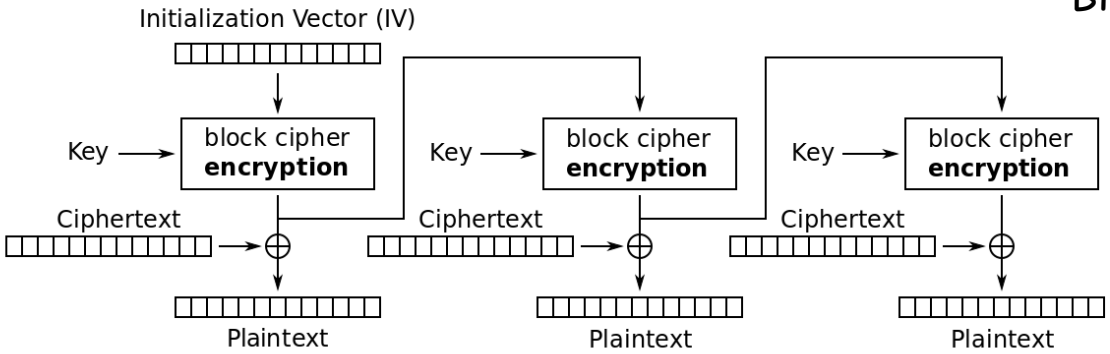
- Strengths:
  - Doesn't show patterns in the plaintext
  - Is the most common mode
  - Is fast and relatively simple
  - Parallel decryption
- Weaknesses:
  - CBC requires the reliable transmission of all the blocks sequentially (may not be suitable for high packet-loss traffic, like video/music streaming)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

Output Feedback Mode (OFB)



Output Feedback (OFB) mode encryption

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



Output Feedback (OFB) mode decryption

- Weakness: If attacker knows plaintext block P, can replace it with desired P'. (Similar to CBC malleability)

Counter Mode (CTR)

- Similar to OFB
- 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