Lecture 20: Confidentiality (Via Encryption)

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Goals for Today

- <u>Learning Objectives</u>:
  - Understand different methods for message confidentiality
- <u>Announcements, etc</u>:
  - Midterm grades released on Compass2G.
    - Refer to last class' slides for reliable grade distribution
    - Midterm Stdev: 14.61 (16%)
  - MP3 Checkpoint #2: Wednesday Oct 23 at 6pm
  - **Also Oct 23:** *<u>Special Guest Lecture on Human Factors!</u>*

Reminder: Please put away devices at the start of class

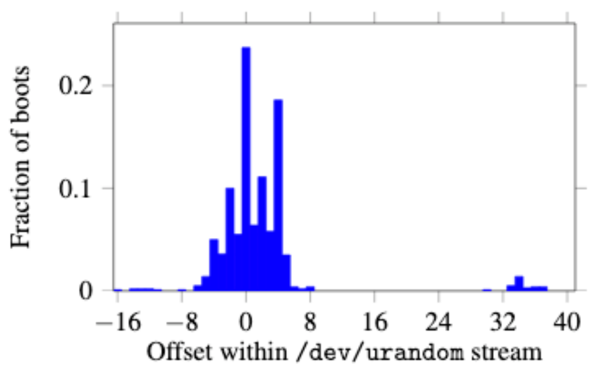
Goals for Today

- Announcements, etc:
  - u-pick-em lectures!!
    - Discussion thread on piazza
    - poll coming soon

Reminder: Please put away devices at the start of class

Fact Check: /dev/random v. /dev/urandom

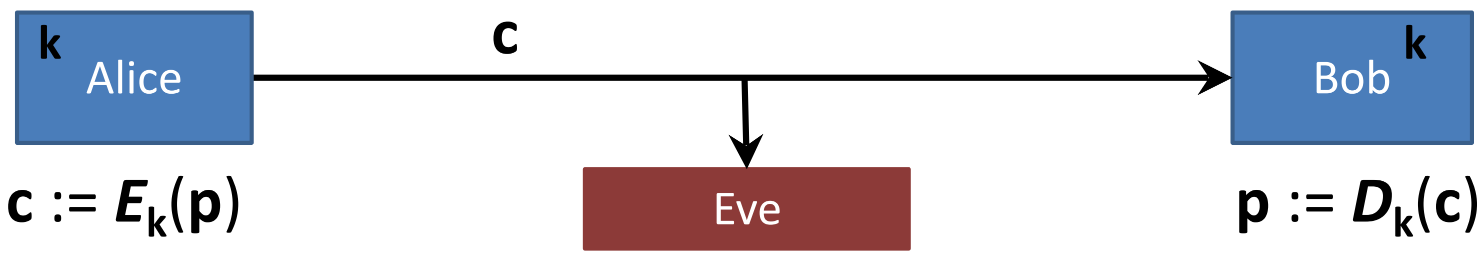
- Same cryptographically-secure PRG is being used under the hood for both entropy sources; very few differences:
  - `/dev/random` — blocks if there is less entropy available than request (specifically, only returns random bytes within the estimated number of bits of noise in the entropy pool).
  - /dev/urandom — does not block while waiting for more entropy
- So is `/dev/random` more secure?
  - In the normal case, **no**. So little entropy is needed that they are effectively identical in their steady state.
  - In the worst case, **YES**.



[Heninger et al., USENIX Security '12]

Confidentiality

Goal: Keep contents of message **p** secret from an *eavesdropper*



Terminology

p	plaintext
c	ciphertext
k	secret key
E	encryption function
D	decryption function

Classic Cryptography: Caesar Cipher

First recorded use: Julius Caesar (100-44 BC)

Replaces each plaintext letter with one a fixed number of places down the alphabet

Encryption: $\mathbf{c_i} := (\mathbf{p_i} + \mathbf{k}) \bmod 26$

Decryption: $\mathbf{p_i} := (\mathbf{c_i} - \mathbf{k}) \bmod 26$

e.g. (**k**=3):

```
           Plain:   ABCDEFGHIJKLMNOPQRSTUVWXYZ
          +Shift:   33333333333333333333333333
=Cipher:            DEFGHIJKLMNOPQRSTUVWXYZABC

           Plain:   attack at dawn
            +Key:   333333 33 3333
=Cipher:            dwwdfn dw gdzq
```

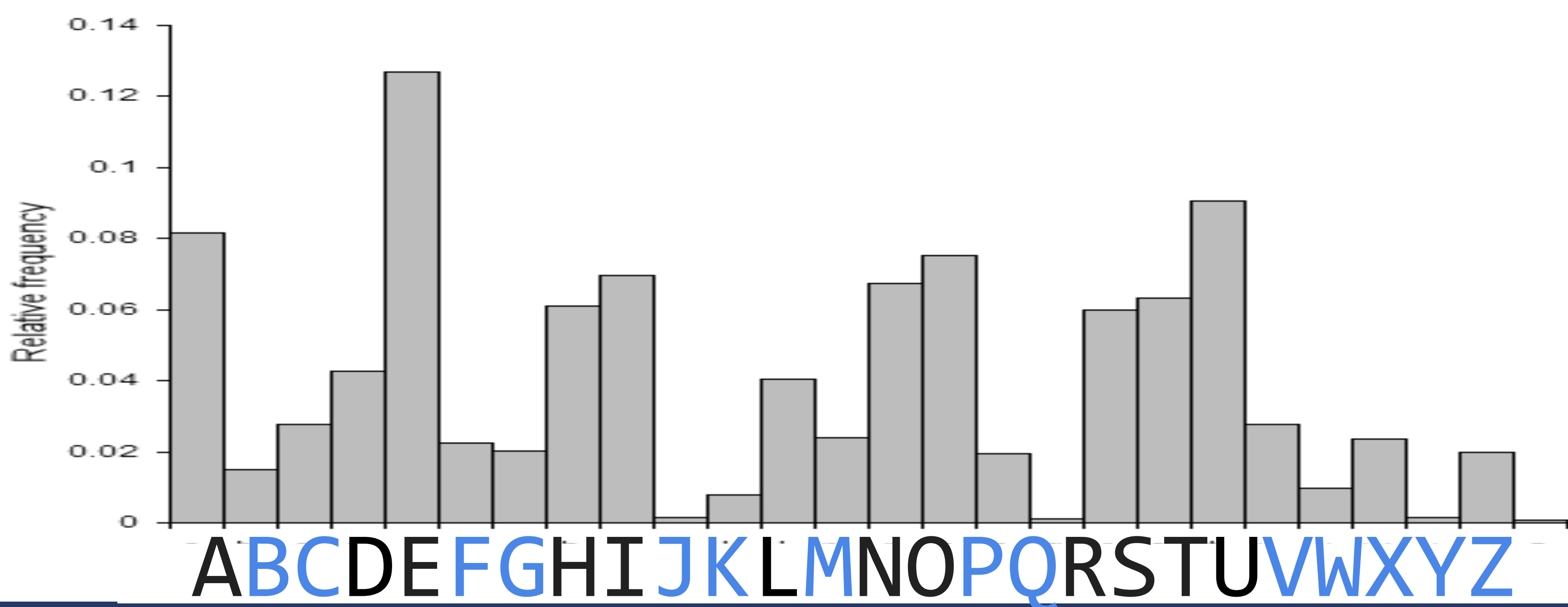
Cryptanalysis of Caesar Cipher

Only 26 possible keys:

Try every possible **k** by *"brute force"*

Can a computer recognize the right one?

Use *frequency analysis*: English text has distinctive letter frequency distribution



Classic Cryptography: Vigènere Cipher

First described by Bellaso in 1553,
later misattributed to Vigenère

Called « le chiffre indéchiffrable »
("the indecipherable cipher")

Encrypts successive letters using a sequence of Caesar ciphers determined by the letters of a keyword

For an **n**-letter keyword **k**,

Encryption: $\mathbf{c_i} := (\mathbf{p_i} + \mathbf{k}_{\mathbf{i} \bmod \mathbf{n}}) \bmod 26$

Decryption: $\mathbf{p_i} := (\mathbf{c_i} - \mathbf{k}_{\mathbf{i} \bmod \mathbf{n}}) \bmod 26$

Example: **k**=ABC (i.e. $\mathbf{k}_0=0$, $\mathbf{k}_1=1$, $\mathbf{k}_2=2$)

Plain:	`bbbbbb`	`amazon`
+Key:	`012012`	`012012`
=Cipher:	`bcdbcd`	`anczpp`

Cryptanalysis of Vigènere

Frequency analysis over entire cipher text doesn't work; distribution of English letters will be distorted by rotating key. However…

Simple, if we know the keyword length, **n**:

1. Break ciphertext into **n** slices
2. Solve each slice as a Caesar cipher

How to find **n**? One way: Kasiski method (1863)

Published 1863 by Kasiski (earlier known to Babbage?)

Repeated strings in long plaintext
will sometimes, by coincidence,
be encrypted with same key letters

Plain:	**CRYPTO**ISSHORTFOR**CRYPTO**GRAPHY
+Key:	ABCDABCDABCDABCDABCDABCD
=Cipher:	**CSASTP**KVSIQUTGQU**CSASTP**IUAQJB

Distance between repeated strings in ciphertext is likely a multiple of key length e.g., distance 16 implies **n** is 16, 8, 4, 2, 1

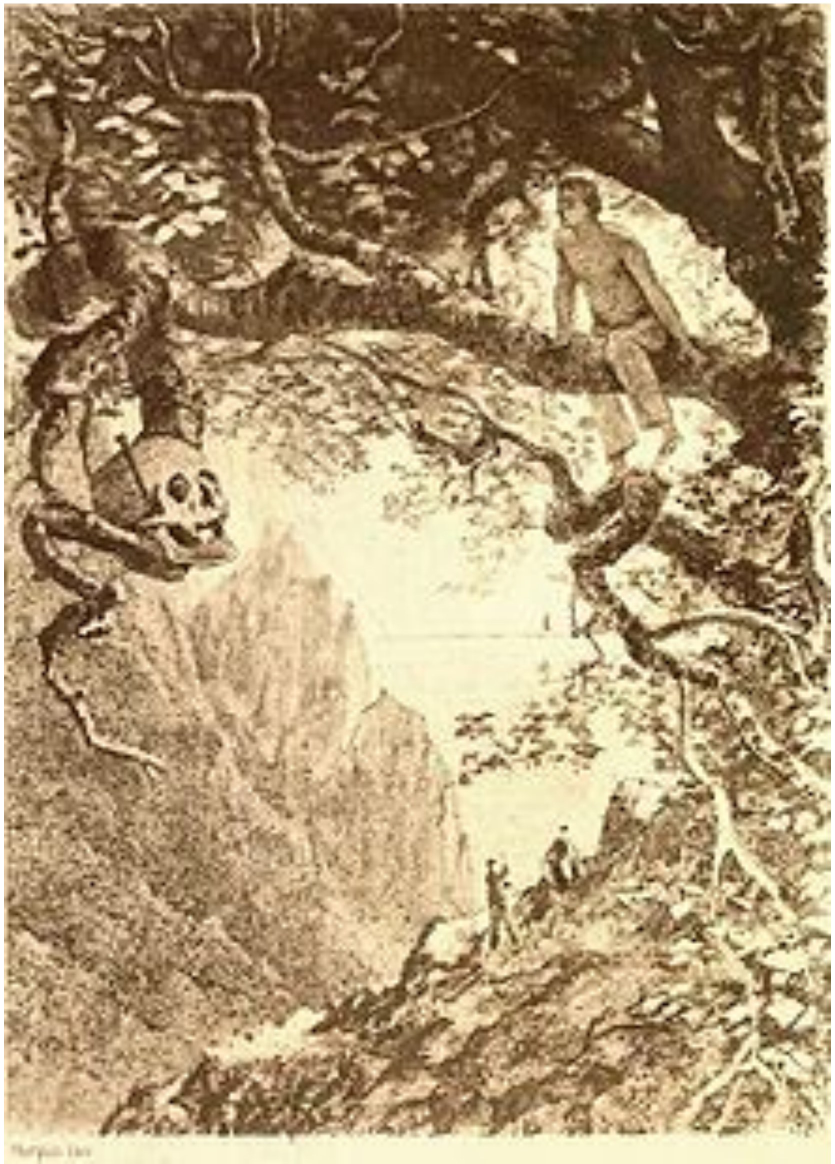
[What if key is as long as the plaintext?]

Classic Cryptography: "The Gold Bug" Substitution Cipher

"The Gold Bug"

By Edgar Allen Poe, 1843

```
53‡‡†305))6*;4826)4‡.)4‡);80
6*;48†8¶60))85;1‡(;:‡*8†83(88)
5*†;46(;88*96*?;8)*‡(;485);5*†
2:*‡(;4956*2(5*–4)8¶8*;40692
85);)6†8)4‡‡;1(‡9;48081;8:8‡1
;48†85;4)485†528806*81(‡9;48
;(88;4(‡?34;48)4‡;161;:188;‡?;
```



Kerckhoffs's Principles

1st: The system must be practically, if not mathematically, indecipherable;

2nd: The system must not require secrecy and must not cause inconvenience should it fall into the hands of the enemy;

3rd: The key must be able to be used in communiques and retained without the help of written notes, and be changed or modified at the discretion of the correspondents;

4th: The system must be compatible with telegraphic communication;

5th: The system must be portable, and remain functional without the help of multiple people;

6th: Finally, it's necessary, given the circumstances in which the system will be applied, that it's easy to use, is undemanding, not overly stressful, and doesn't require the knowledge and observation of a long series of rules

Kerckhoffs's Principles

1st: The system must be practically, if not mathematically, indecipherable;

2nd: The system must not require secrecy and must not cause inconvenience should it fall into the hands of the enemy;

3rd: The key must be able to be used in communiques and retained without the help of written notes, and be changed or modified at the discretion of the correspondents;

4th: The system must be compatible with telegraphic communication;

5th: The system must be portable, and remain functional without the help of multiple people;

6th: Finally, it's necessary, given the circumstances in which the system will be applied, that it's easy to use, is undemanding, not overly stressful, and doesn't require the knowledge and observation of a long series of rules

One-Time Pads

Alice and Bob jointly generate a secret,
very long, string of random bits
(the one-time pad, **k**)

To encrypt: $\mathbf{c_i} = \mathbf{p_i}$ xor $\mathbf{k_i}$

To decrypt: $\mathbf{p_i} = \mathbf{c_i}$ xor $\mathbf{k_i}$

a	**b**	**a** xor **b**
0	0	0
0	1	1
1	0	1
1	1	0

a xor **b** xor **b** = **a**

a xor **b** xor **a** = **b**

"one-time" means you should never reuse any part of the pad.

If you do:

Let $\mathbf{k_i}$ be pad bit

Adversary learns (**a** xor $\mathbf{k_i}$) and (**b** xor $\mathbf{k_i}$)

Adversary xors those to get (**a** xor **b**), which is useful to him [How?]

Provably secure [Why?]

Usually impractical [Why? Exceptions?]

Examples of OTP's being used?

Numbers Stations

Mysterious shortwave radio stations that appeared intermittently throughout the 1900’s.

First known use was in World War I.

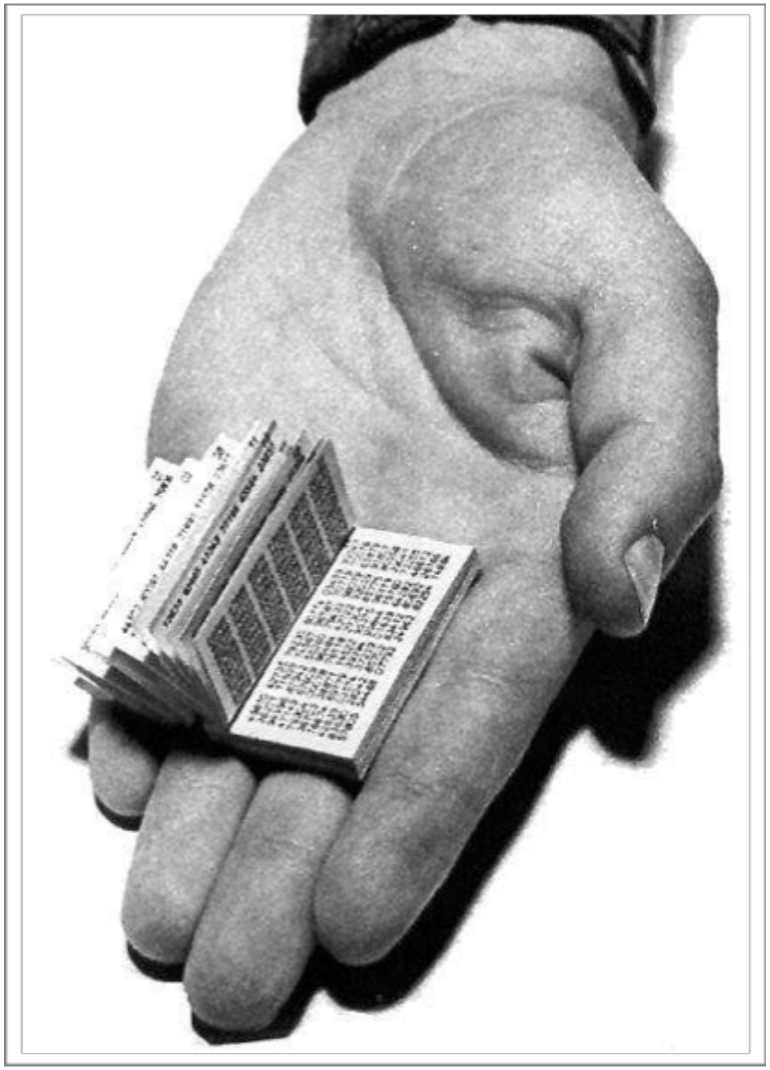
Commonly used in Cold War.

Still in use today! (e.g., Walter Kendall Myers, 2009)

Used as a one-way voice link to transmit messages with a one-time pad; only recipient could decrypt.

One-Time Pads

- Used when perfect secrecy is necessary
- Serves as ideal of perfect symmetric encryption scheme
- Does OTP provide integrity?



Stream Ciphers

Obvious idea: Use a **pseudorandom generator** instead of a truly random pad

(Recall: Secure **PRG** inputs a seed **k**, outputs a stream that is practically indistinguishable from true randomness unless you know **k**)

Called a **stream cipher**:

1. Start with shared secret key **k**
2. Alice & Bob each use **k** to seed the PRG
3. To encrypt, Alice XORs next bit of her generator's output with next bit of plaintext
4. To decrypt, Bob XORs next bit of his generator's output with next bit of ciphertext

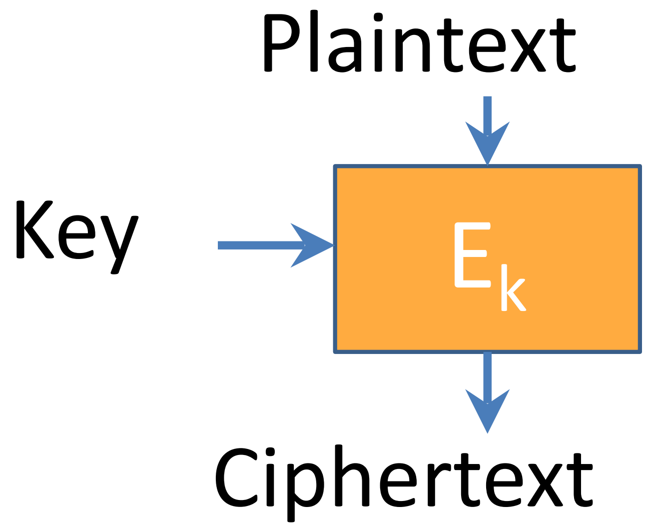
Works nicely, but: don't <u>*ever*</u> reuse the key, or the generator output bits

Block Ciphers

Functions that encrypts fixed-size blocks with a reusable key.

Inverse function decrypts when used with same key.

The most commonly used approach to encrypting for confidentiality.



A block cipher is <u>not</u> a pseudorandom function [Why?]

Block Ciphers

What we want instead:

pseudorandom permutation (PRP)

function from **n**-bit input to **n**-bit output

distinct inputs yield distinct outputs (one-to-one)

Defined similarly to **PRF**:

practically indistinguishable from a

random permutation without secret **k**

Basic challenge: Design a hairy function

that is invertible, but only if you have the key

Minimal properties of a good block cipher:

- Highly nonlinear ("confusion")
- Mixes input bits together ("diffusion")
- Depends on the key

Definition: a cipher is "Semantically Secure"

Similar game to PRF/PRG/PRP definition:

1. We flip a coin secretly to get a bit **b**, random secret **k**
2. Mallory chooses arbitrary $\mathbf{m_i}$ in **M**, gets to see $\boldsymbol{Enc_k}(\mathbf{m_i})$
3. Mallory chooses two messages $\mathbf{m'_0}$ and $\mathbf{m'_1}$ not in **M**
4. If **b**=0, let **c** be $\boldsymbol{Enc_k}(\mathbf{m'_0})$

   If **b**=1, let **c** be $\boldsymbol{Enc_k}(\mathbf{m'_1})$
5. Mallory can see **c**
6. Mallory guesses **b**, wins if guesses correctly

Also known as: IND-CPA "Chosen plaintext attack"

Advanced Encryption Standard (AES)

Today's most common block cipher:

AES (**Advanced Encryption Standard**)

- Designed by NIST competition, long public comment/discussion period
- Widely believed to be secure, but we don't know how to prove it
- Variable **key size** and **block size**
- We'll use 128-bit key, 128-bit block (are also 192-bit and 256-bit versions)
- Ten **rounds**: Split **k** into ten **subkeys**, performs set of operations ten times, each with diff. subkey

Padding

Remaining problem:

How to encrypt longer messages?

Padding:

- Can only encrypt in units of cipher blocksize, but message might not be multiples of blocksize
- *Solution:* Add padding to end of message

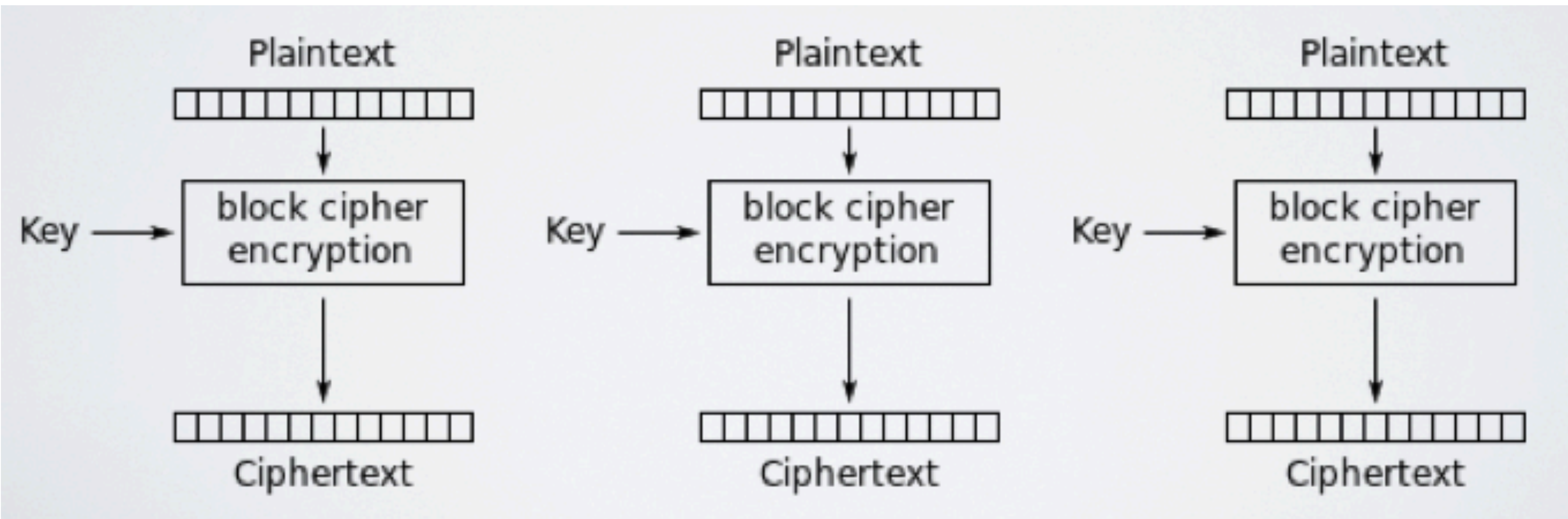
Must be able to recognize and remove padding afterward

- Common approach: Add **n** bytes that have value **n**

[Caution: What if message ends at a block boundary?]

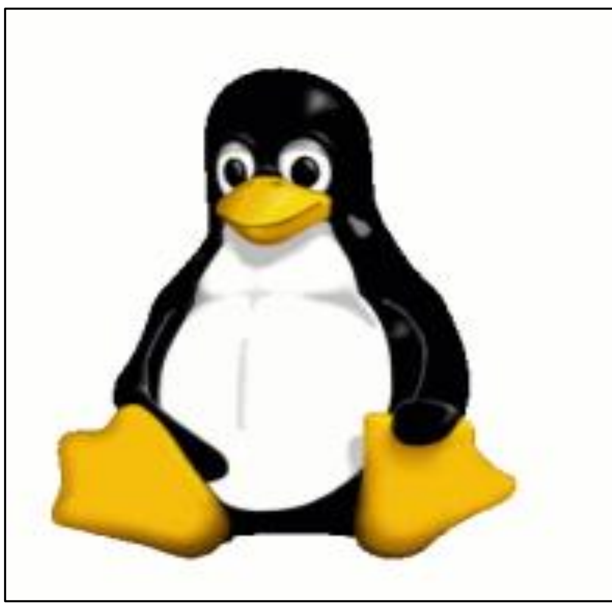
Block Ciphers

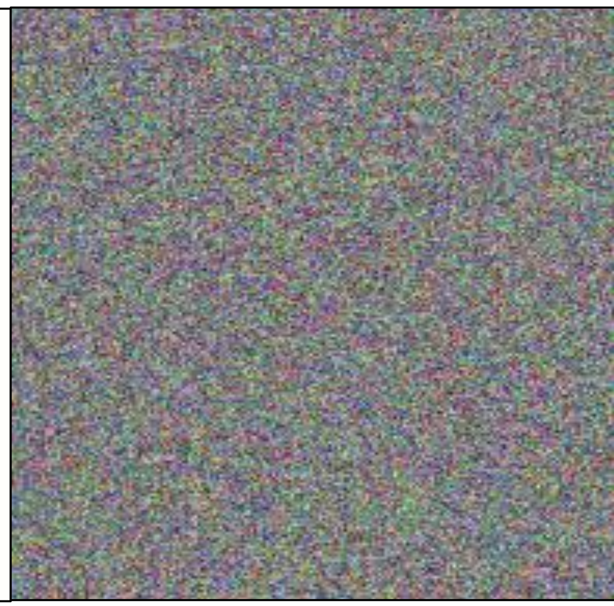
- Several modes of operation for longer messages
- **Electronic Code Book (ECB) Mode:** basically, encrypts each block separately.
- Avoid — why?

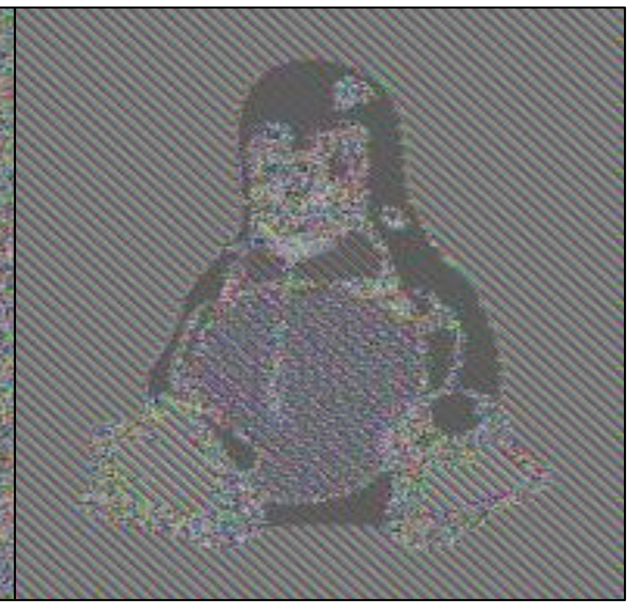


Block Ciphers

- Several modes of operation for longer messages
- **Electronic Code Book (ECB) Mode:** basically, encrypts each block separately.
- Avoid — why?

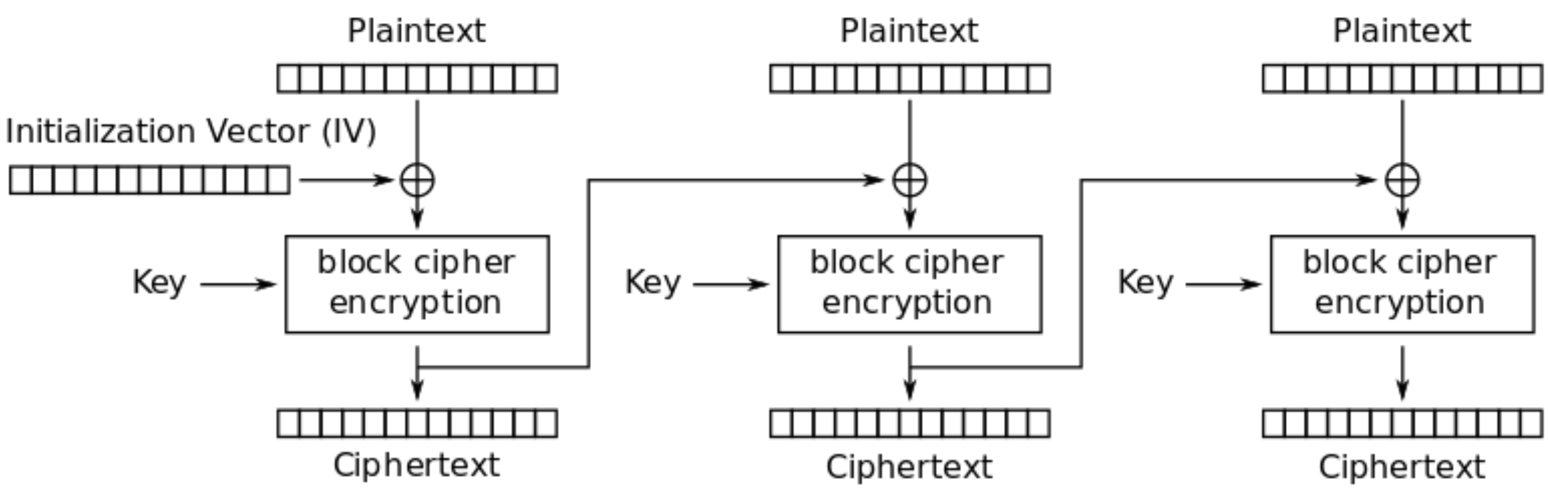

Plaintext


Pseudorandom


ECB mode

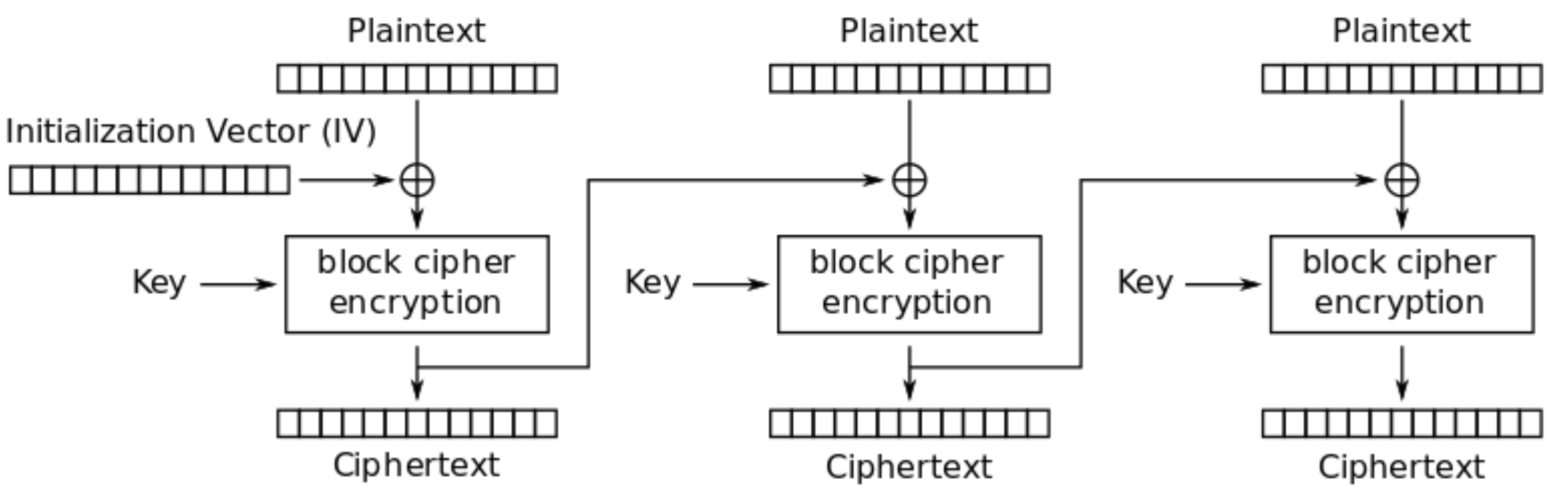
Block Ciphers

- **Cipher Block Chaining (CBC) Mode**: XOR the last cipher block into the next plaintext.
- Use random **Initialization Vector (IV)** for the first block in lieu of a previous cipher block.
- Subtle attack possible if attacker knows IV, controls plain text. [Other Disadvantages?]



Block Ciphers

- **IV Reuse Attack**
  - Early implementations of CBC reused the last cipher text block as the IV for the next message
  - What if attacker gets to see previous encryption before? Is CBC secure against Chosen-Plaintext Attack?

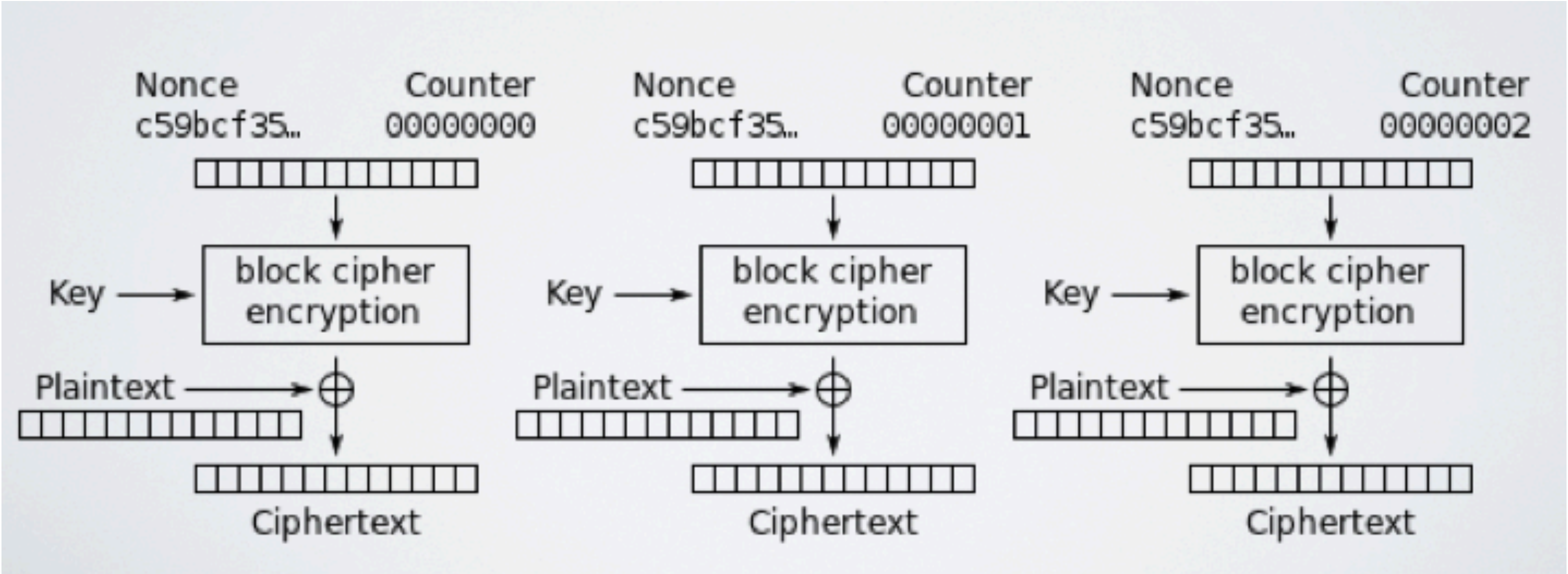


Block Ciphers

- **Padding Oracle Attack on CBC**
  - CBC Decryption: Decrypt all blocks, validate the padding, remove padding, return message.
  - Padding incorrect? <u>Return “Decryption Failed,” not “Invalid Padding!”</u>
  - Otherwise, he attacker can use the server as a padding oracle to decrypt (and sometimes encrypt) messages.

Block Ciphers

- **Counter Mode (CTR)**: Generate next key stream block by encrypting successive values of a counter
- Block cipher becomes stream cipher
- Why is nonce (i.e., Message ID) needed?
- Advantages?



From the command line…

```
$ KEY=$(openssl rand -hex 16)
```

Generates a random string

```
$ openssl aes-256-cbc -in mymsg.txt -out mymsg.enc
-p -K ${KEY} -iv $(openssl rand -hex 16)
  key=8582D9E1A36DA4DB065394FB1F401DB3
  iv =DBB272FE6486C4D9B09DBE464E080468
```

Prints the key and IV

```
$ openssl aes-256-cbc -d -in mymsg.enc -out mymsg.txt   -K
${KEY} -iv <iv from above>
```

- By default, uses the standard padding described earlier
- Unfortunately, you have to handle prepending/extracting the IV on your own

Definition: a cipher is "Semantically Secure"

Similar game to PRF/PRG/PRP definition:

1. We flip a coin secretly to get a bit **b**, random secret **k**
2. Mallory chooses arbitrary $\mathbf{m_i}$ in **M**, gets to see $\boldsymbol{Enc}_k(\mathbf{m_i})$
3. Mallory chooses two messages $\mathbf{m'_0}$ and $\mathbf{m'_1}$ not in **M**
4. If **b**=0, let **c** be $\boldsymbol{Enc}_k(\mathbf{m'_0})$

   If **b**=1, let **c** be $\boldsymbol{Enc}_k(\mathbf{m'_1})$
5. Mallory can see **c**
6. Mallory guesses **b**, wins if guesses correctly

Also known as: IND-CPA "Chosen plaintext attack"

What is not covered by "Semantic Security?"

- "Malleability" attacks
Given just some ciphertexts, can the attacker create new ciphertexts that Bob decrypts to the wrong value?

- Encryption does NOT IMPLY integrity!
Often you really want both ("authenticated encryption")

- Chosen Ciphertext attacks
The "semantic security" definition does not allow the adversary to see decryptions of (potentially garbage) ciphertexts chosen by the adversary

- Solution: Encrypt-then-MAC

Next class…

Assumption we've been making so far:

Alice and Bob ***shared a secret key*** in advance

Amazing fact:

Alice and Bob can have a **public** conversation to derive a shared key!

Next class…

So Far

Message Integrity

Randomness /Pseudorandomness

Confidentiality: Stream Ciphers, Block Ciphers

Friday…

Key Exchange, Key Management, Public Key Crypto