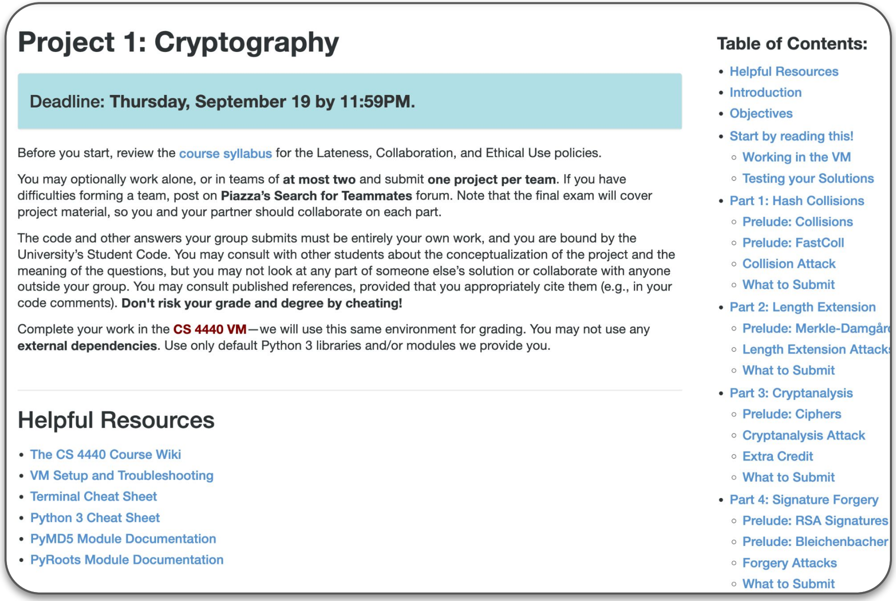
Week 3: Lecture B
Block Ciphers

Thursday, September 5, 2024

Announcements

- **Project 1: Crypto** released (see **Assignments** page on course website)
  - **Deadline:** Thursday, September 19th by 11:59 PM



Project 1: Cryptography

Deadline: **Thursday, September 19 by 11:59PM.**

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies.

You may optionally work alone, or in teams of **at most two** and submit **one project per team**. If you have difficulties forming a team, post on **Piazza's Search for Teammates** forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don't risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM**—we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.

Helpful Resources

- The CS 4440 Course Wiki
- VM Setup and Troubleshooting
- Terminal Cheat Sheet
- Python 3 Cheat Sheet
- PyMD5 Module Documentation
- PyRoots Module Documentation

Table of Contents:

Announcements

utahsec

See Discord for meeting info!

utahsec.cs.utah.edu

Announcements

Questions?

Last time on CS 4440…

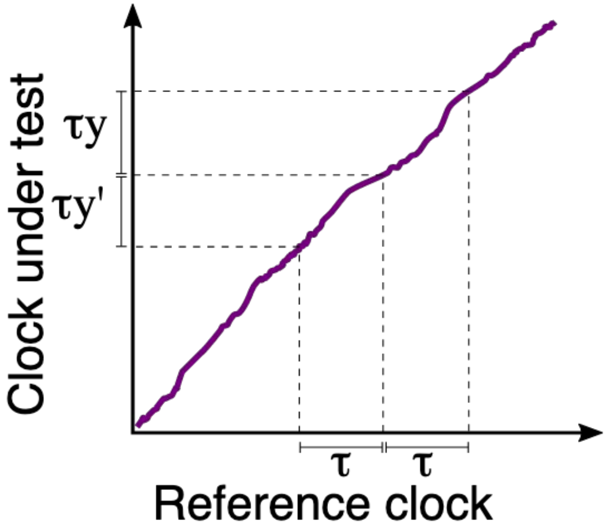
Pseudo-random Keys
One-time Pads
Transposition Ciphers
Cipher Metrics

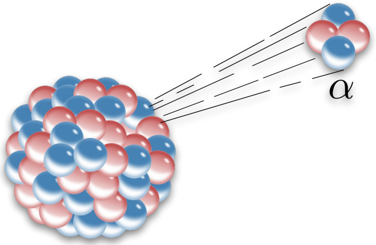
Generating Random Keys

- **Physical randomness:**
  - ???

Generating Random Keys

- **Physical randomness:**
  - Coin flips
  - Atomic decay
  - Thermal noise
  - Electromagnetic noise
  - Physical variation
    - Clock drift
    - DRAM decay
    - Image sensor errors
    - SRAM startup-state
  - Lava Lamps



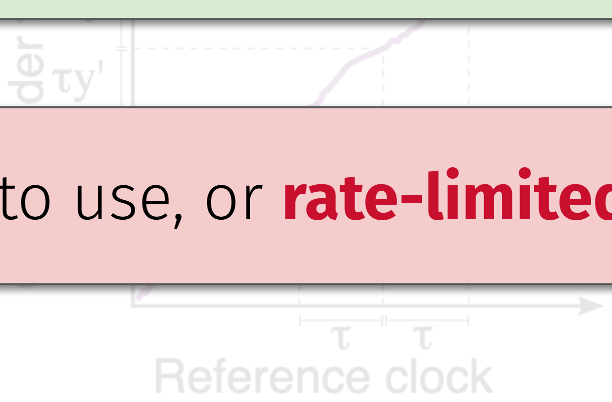


Generating Random Keys

- **Physical randomness:**
  - Coin flips
  - Atomic decay
  - Thermal noise
  - Electromagneti
  - Physical variati
    - Clock drift
    - DRAM dec
    - Image sen
    - SRAM star
  - Lava Lamps

Generating Random Keys

- **Physical randomness:**
  - Coin flips
  - Atomic dec
  - Thermal no
  - Electromag
  - Physical variation
    - Clock drift
    - DRAM
    - Image
    - SRAM
  - Lava Lamps



Highest guarantees of **security**

Difficult to use, or **rate-limited**

Pseudo-random Key Generators

- What is **true randomness**?
  - **???**

Pseudo-random Key Generators

- What is **true randomness**?
  - **Physical** process that's inherently random
  - **Secure** yet **impractical**
    - Scarce, hard to use
    - Rate-limited
- Pseudo-random generator (PRG)
  - **Input:** ???
  - **Output:** ???

Pseudo-random Key Generators

- What is **true randomness**?
  - **Physical** process that's inherently random
  - **Secure** yet **impractical**
    - Scarce, hard to use
    - Rate-limited

- Pseudo-random generator (PRG)
  - **Input:** a small **seed** that is **truly random**
  - **Output:** long sequence that **appears random**

Pseudo-random Generators (PRGs)

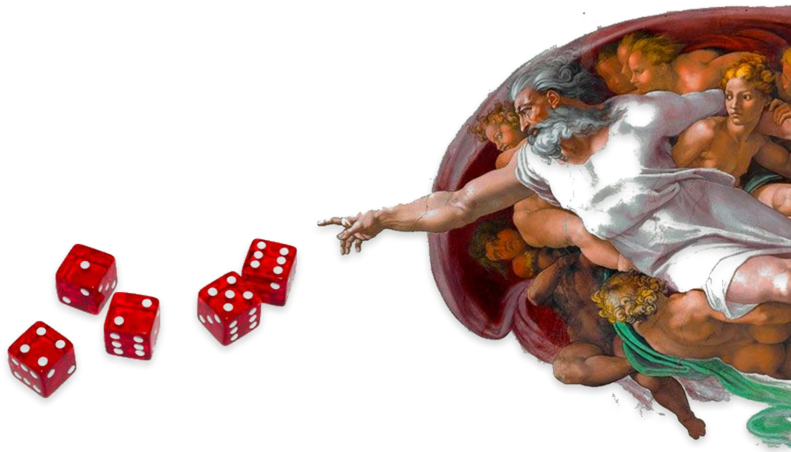
- We say a **PRG** is **secure** if Mallory can't do better than random guessing
- **Problem:** How much **true randomness** is **enough**?
  - **Example:** **one coin flip** = Mallory needs **very few tries** to guess
- **Problem:** Is our "true randomness" **truly random**?
  - **Example:** coin flip output = **one in two**. Lava lamps have way more!
- **Solutions:**
  - **???**

Pseudo-random Generators (PRGs)

- We say a **PRG** is **secure** if Mallory can't do better than random guessing
- **Problem:** How much **true randomness** is **enough**?
  - **Example:** **one coin flip** = Mallory needs **very few tries** to guess
- **Problem:** Is our "true randomness" **truly random**?
  - **Example:** coin flip output = **one in two**. Lava lamps have way more!
- **Solutions:**
  - Generate a bunch of true randomness **over a long time** from a **high entropy source**
  - Run through a **PRF** to get an easy-to-work-with, **fixed-length** randomness (e.g., 256 bits)

Practical Randomness

- Where do you get **true** randomness?
- Modern OSes typically collect randomness
- They give you API calls to capture it
- e.g., Linux:
  - `/dev/random` is a device that gives random bits; it blocks until available
  - `/dev/urandom` gives output of a PRG; nonblocking; seeded from `/dev/random` eventually



One-time Pads

- Alice and Bob generate **???**

One-time Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad **k**
  - Encryption: $c_i := p_i \text{ XOR } k_i$
  - Decryption: $p_i := c_i \text{ XOR } k_i$
- Are they practical?
  - **???**
- Are they secure?
  - **???**

A	B	Q
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 Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad k
  - Encryption:
  - Decryption:
- Are they practical?
  - ???
- Are they secure?
  - ???

Provably **Secure**
(if key is **random** + not **reused**)

Highly **Impractical**

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

a XOR b XOR b = a

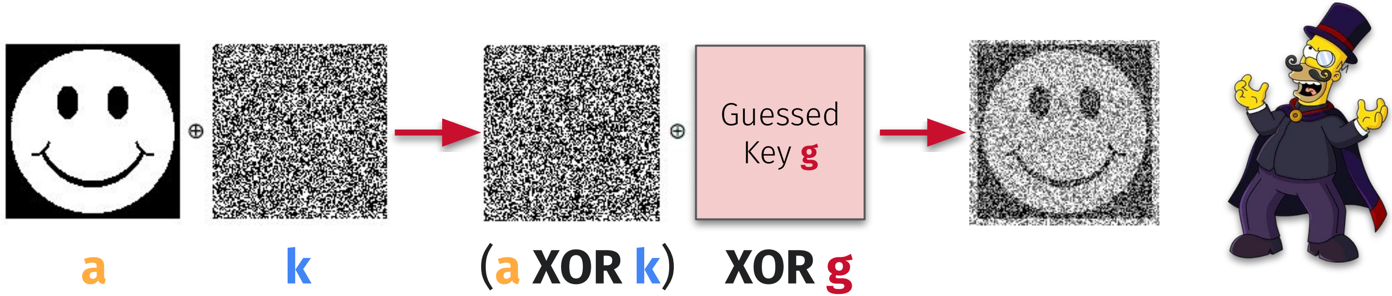
a XOR b XOR a = b

Attacking OTPs

- What happens if the key **isn't** truly random?

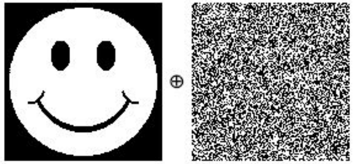
Attacking OTPs

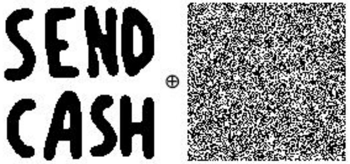
- What happens if the key **isn't** truly random?
  - If Mallory correctly guesses some key bits, she can **recover** parts of the plaintext



Attacking OTPs

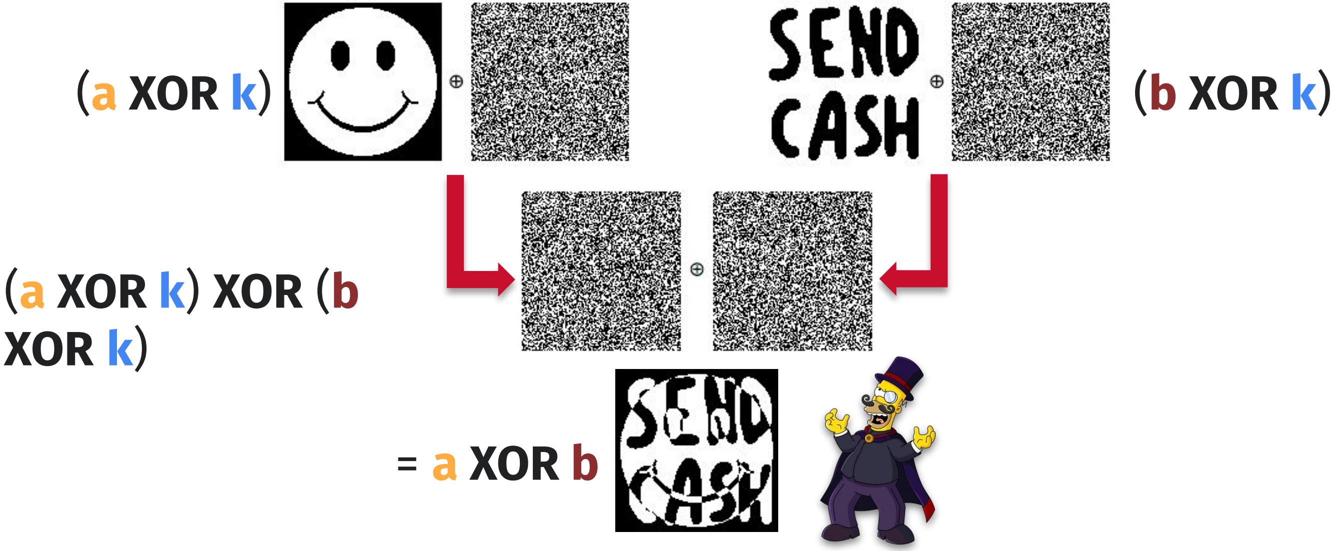
- What if Mallory intercepts multiple messages that **reuse** the same key?
  - Mallory can **XOR them together** to recover partial plaintext information!

(a XOR k) 

 (b XOR k)

Attacking OTPs

- What if Mallory intercepts multiple messages that **reuse** the same key?
  - Mallory can **XOR them together** to recover partial plaintext information!



Stream Cipher

- **Idea:** Use a **Pseudo-random Generator** instead of a truly random pad

- **Recall:** a secure PRG inputs a **true-random seed**, outputs a stream that's **indistinguishable** from true randomness (unless attacker **knows seed**)

  1. Start with a shared secret **truly random seed** (from a lava lamp, mouse clicks, etc.)
  2. Alice & Bob each use this seed to seed their PRG and generate **k bits of PRG output**
  3. To encrypt and decrypt, perform the same operations as the One-time Pad:
     - Encryption: c_i := p_i **XOR** k_i
     - Decryption: p_i := c_i **XOR** k_i

Stream Cipher

- **Idea:** Use a pse … ndom pad
- **Recall:** Secure … tically indistinguishab … knows **k**)

What is the tradeoff between an OTP and Stream Cipher?

1. Start with shared secret key truly random number **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**

Stream Cipher

- **Idea:** Use a pse… …ndom pad
- **Recall:** Secure … …tically indistinguishab… …knows **k**)

1. Start with sh…
2. Alice & Bob e…
3. To encrypt, **A**… …**it of plaintext**
4. To decrypt, **Bob XORs next bit** of his generator's output with **next bit of ciphertext**

What is the tradeoff between an **OTP** and **Stream Cipher**?

Provably **Secure**
(if key is **random** + not **reused**)

Much more **practical**

Stream Cipher

- **Idea:** Use a pseudo… …y random pad

- **Recall:** Secure PR… …ractically indistinguishable … …ker knows **k**)

Are stream ciphers vulnerable to **attack**?

1. Start with shared secret key truly random number **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**

Stream Cipher

- **Idea:** Use a pseudo… …y random pad

- **Recall:** Secure PR… …ractically indistinguishable … …ker knows **k**)

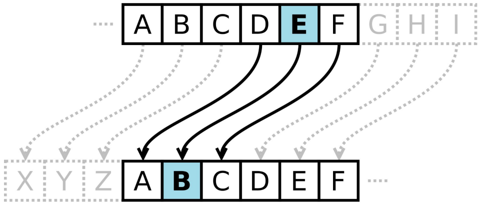
1. Start with shared…
2. Alice & Bob each…
3. To encrypt, **Alice** … **…xt bit of plaintext**
4. To decrypt, **Bob X…** … **…t bit of ciphertext**

Are stream ciphers vulnerable to **attack**?

Seed or key reuse helps Mallory recover plaintext!

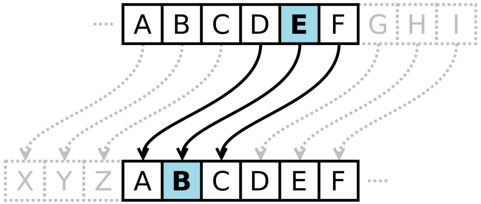
Substitution vs Transposition Ciphers

- **Substitution:** replace plaintext symbols with others
  - **Examples: ???**



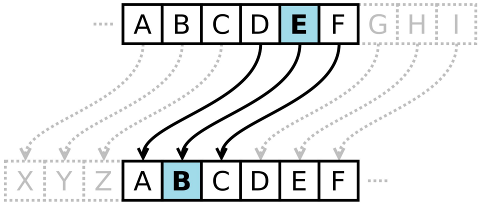
Substitution vs Transposition Ciphers

- **Substitution:** replace plaintext symbols with others
  - **Examples:** simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness: ???**

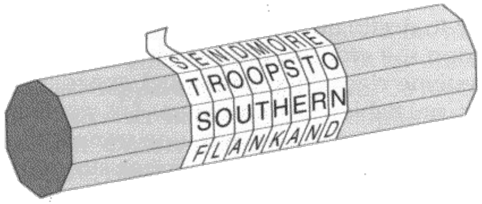


Substitution vs Transposition Ciphers

- **Substitution:** replace plaintext symbols with others
  - **Examples:** simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness:** although letters changed, **frequencies upheld**

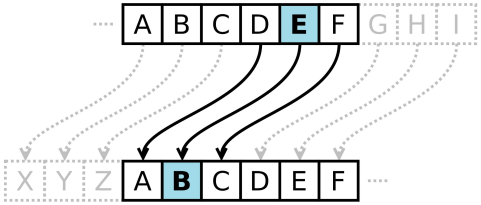


- **Transposition:** plaintext symbols are rearranged
  - **Examples: ???**

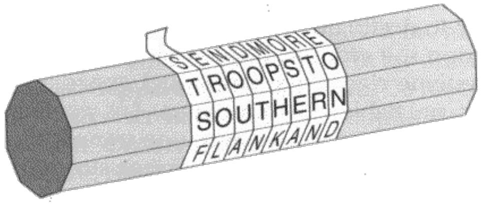


Substitution vs Transposition Ciphers

- **Substitution:** replace plaintext symbols with others
  - **Examples:** simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness:** although letters changed, **frequencies upheld**

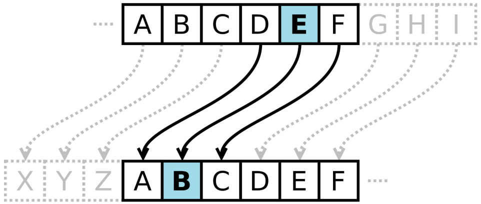


- **Transposition:** plaintext symbols are rearranged
  - **Examples:** columnar, rail fence / zig zag / scytale, grids
  - **Key weakness:** **???**

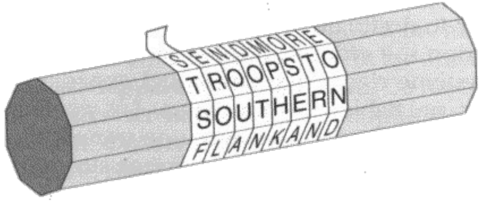


Substitution vs Transposition Ciphers

- **Substitution:** replace plaintext symbols with others
  - **Examples:** simple shifts (Caesar, Vigènere), XORs (OTP, stream)
  - **Key weakness:** although letters changed, **frequencies upheld**



- **Transposition:** plaintext symbols are rearranged
  - **Examples:** columnar, rail fence / zig zag / scytale, grids
  - **Key weakness:** plaintext letters in ciphertext; **anagram attacks**



Columnar Transposition

- Rearrange plaintext symbols to create ciphertext
  - Create a table with |**k**| columns and **|p|/|k|** rows (**k** is the keyword)
  - Place plaintext symbols in columns (left to right), cycling around to next row of the first column when current row of last column is filled
  - Create the ciphertext by writing entire columns (as a serial stream) to the output, where the keyword determines the column order

- Example:
  - **k** = "ZEBRAS" (632415)
  - **p** = "We are discovered flee at once"
  - **c** = EVLN**X** ACDT**Q** ESEA**M**
ROFO**P** DEEC**D** WIREE
  - Replace **null** with nonsense symbol

6	3	2	4	1	5
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	F	L	E
E	A	T	O	N	C
E	null	null	null	null	null

Columnar Transposition

- **How does Bob decrypt** Alice's columnar-transposition-encrypted message?

k = "ZEBRAS" (632415)

p = "We are discovered flee at once"

c = EVLN**X** ACDT**Q** ESEA**M**
ROFO**P** DEEC**D** WIREE

Columnar Transposition

- **How does Bob decrypt** Alice's columnar-transposition-encrypted message?

k = "ZEBRAS" (632415)

p = "We are discovered flee at once"

c = EVLN**X** ACDT**Q** ESEA**M**
ROFO**P** DEEC**D** WIREE

E	A	E	R	D	W
V	C	S	O	E	I
L	D	E	F	E	R
N	T	A	O	C	E
X	Q	M	P	D	E

Columnar Transposition

- **How does Bob decrypt** Alice's columnar-transposition-encrypted message?

k = "ZEBRAS" (632415)

p = "We are discovered flee at once"

c = EVLN**X** ACDT**Q** ESEA**M**
ROFO**P** DEEC**D** WIREE

1	2	3	4	5	6
E	A	E	R	D	W
V	C	S	O	E	I
L	D	E	F	E	R
N	T	A	O	C	E
X	Q	M	P	D	E

E	A	E	R	D	W
V	C	S	O	E	I
L	D	E	F	E	R
N	T	A	O	C	E
X	Q	M	P	D	E

Columnar Transposition

- **How does Bob decrypt** Alice's columnar-transposition-encrypted message?

k = "ZEBRAS" (632415)

p = "We are discovered flee at once"

c = EVLNX ACDTQ ESEAM ROFOP DEECD WIREE

E	A	E	R	D	W
V	C	S	O	E	I
L	D	E	F	E	R
N	T	A	O	C	E
X	Q	M	P	D	E

1	2	3	4	5	6
E	A	E	R	D	W
V	C	S	O	E	I
L	D	E	F	E	R
N	T	A	O	C	E
X	Q	M	P	D	E

Z	E	B	R	A	S
6	3	2	4	1	5
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	F	L	E
E	A	T	O	N	C
E	M	Q	P	X	D

Columnar Transposition

- **Can you decrypt the ciphertext?**

c = SAKSECROYNSBOWOLYUOL

k = "TEAMS"

Columnar Transposition

- **Can you decrypt the ciphertext?**

c = SAKSECROYNSBOWOLYUOL

k = “TEAMS” (**52134**)

1	2	3	4	5
S	E	Y	O	Y
A	C	N	W	U
K	R	S	O	O
S	O	B	L	L

Columnar Transposition

- **Can you decrypt the ciphertext?**

c = SAKSECROYNSBOWOLYUOL

k = "TEAMS" (**52134**)

T	E	A	M	S
5	**2**	**1**	**3**	**4**
Y	E	S	Y	O
U	C	A	N	W
O	R	K	S	O
L	O	S	B	L

Columnar Transposition

- **Can you decrypt the ciphertext?**

c = SAKSECROYNSBOWOLYUOL

k = “TEAMS” (**52134**)

T	E	A	M	S
5	2	1	3	4
Y	E	S	Y	O
U	C	A	N	W
O	R	K	S	O
L	O	null	null	null

Columnar Transposition

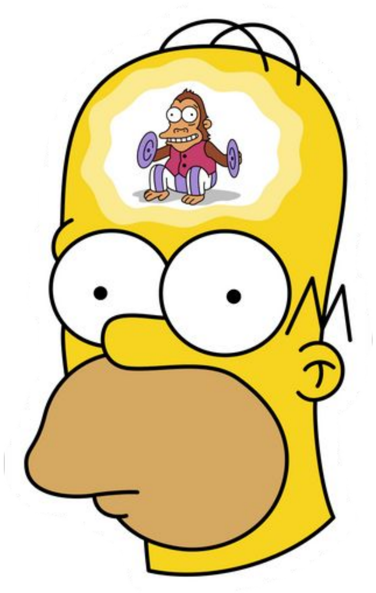
- **Can you decrypt the ciphertext?**

  c = SAKSECROYNSBOWOLYUOL

  k = "TEAMS" (**52134**)

- **"Yes, you can work solo"** (on projects)
  - Though we don't recommend it! 😃

Can we make transposition stronger?



Can we make transposition stronger?

- **More Transposition:**
  - Increase entropy!

k_1 = “ZEBRAS” (632415)
c_1 = EVLN**X** ACDT**Q** ESEA**M** ROFO**P** DEEC**D** WIRE**E**

k_2 = “STRIPE” (632415)
c_2 = CAEI**X** NSOI**N** AEDR**X** LEFW**S** EDRE**E** VTOC**G**

5	6	4	2	3	1
E	V	L	N	A	C
D	T	E	S	E	A
R	O	F	O	D	E
E	C	W	I	R	I
E	null	null	null	null	null

Can we make transposition stronger?

- **More Transposition:**
  - Increase entropy!

k_1 = “ZEBRAS” (632415)
c_1 = EVLNX ACDTQ ESEAM ROFOP DEECD WIREE

k_2 = “STRIPE” (632415)
c_2 = CAEIX NSOIN AEDRX LEFWS EDREE VTOCG

5	6	4	2	3	1
E	V	L	N	A	C
D	T	E	S	E	A
R	O	F	O	D	E
E	C	W	I	R	I
E	null	null	null	null	null

- **Apply Fractionation:**
  - Eliminate anagrams!

“We’re discovered flee at once!”

o–– o o– o–o o …

0101011101100101…

Can we make transposition stronger?

- **More Transposition:**
  - Increase entropy!

k_1 = “ZEBRAS” (632415)
c_1 = EVLNX ACDTQ ESEAM
ROFOP DEECD WIREE

5	6	4	2	3	1
E	V	L	N	A	C
D	T	E	S	E	A
R	O	F	O	D	E
E	C	W	I	R	I
E	null	null	null	null	null

k_2 = “STRIPE” (632415)
c_2 = CAEIX NSOIN AEDRX
LEFWS EDREE VTOCG

- **Apply Fractionation:**
  - Eliminate anagrams!

“We’re discovered flee at once!”

o–– o o– o–o o ...

010101110110 0101...

- **Apply Substitution:**
  - Increase entropy
    + eliminate anagrams!

c = EVLNB ACDTA ESEAR
ROFOX DEECB WIREE

k = ABCAB CABCA BCABC

c_2 = EWNNC CCEVA FUEBT
RPHOY FEFEB XKRFG

Cipher Metrics

- How we "weigh" a cipher's resilience to **cryptanalysis**

- **"Confusion"**
  - **???**

- **"Diffusion"**
  - **???**

Cipher Metrics

- How we "weigh" a cipher's resilience to **cryptanalysis**

- **"Confusion"**
  - Every bit of the ciphertext should depend on **several parts** of the plaintext
  - Maintains that the ciphertext is statistically independent of **the plaintext**

- **"Diffusion"**
  - A change to one plaintext bit should change **50%** of the ciphertext bits
  - A change to one ciphertext should change **50%** of the plaintext bits
  - Plaintext features **spread** throughout the entire ciphertext

Exercise: Cipher Metrics

Cipher	Relies on?	Strength?	Why?
Caesar	**?**	**?**	**?**

Exercise: Cipher Metrics

Cipher	Relies on?	Strength?	Why?
Caesar	**Confusion**	**Weak**	Frequencies unchanged
Vigenere	**?**	**?**	**?**
One-time Pad, Stream Cipher	**?**	**?**	**?**
Transposition	**?**	**?**	**?**
Fractionation	**?**	**?**	**?**

Exercise: Cipher Metrics

Cipher	Relies on?	Strength?	Why?
Caesar	**Confusion**	**Weak**	Frequencies unchanged
Vigenere	**Confusion**	**Weak**	Frequencies unchanged
One-time Pad, Stream Cipher	**Confusion**	**Strong**	Key change = relationship cannot be determined
Transposition	**Diffusion**	**Weak**	Symbols unchanged
Fractionation	**Both!**	**Strong**	Symbols changed, spread

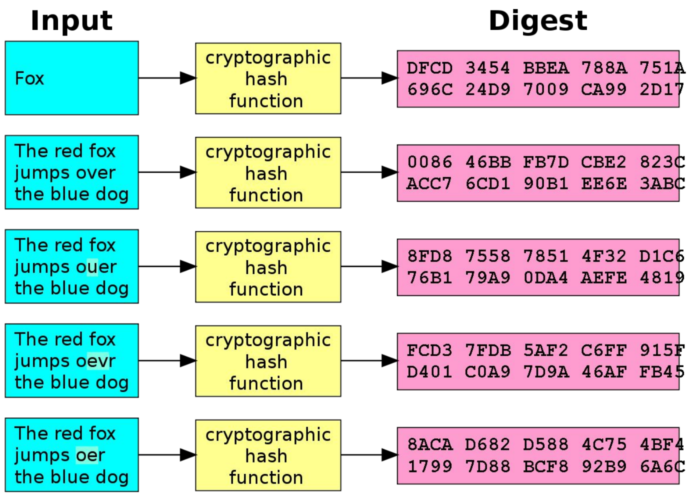
Food for thought...

- **Question:** do we care about confusion and diffusion in **cryptographic hashes**?

Food for thought...

- **Question:** do we care about confusion and diffusion in **cryptographic hashes**?
  - **Absolutely we do!**

- Implications of **low** confusion/diffusion:
  - Tampering, forgery, collisions
  - Pre-image attacks



Questions?

This time on CS 4440...

Block Ciphers
DES and AES
Block Cipher Modes
Building a Secure Channel

Message Confidentiality

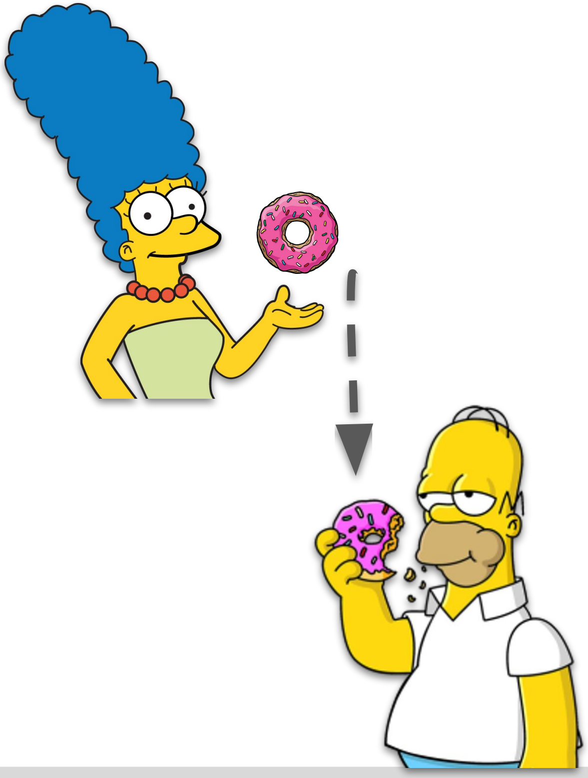
- **Confidentiality:** ensure that only **trusted parties** can read the message
- Terminology:
  - **p** plaintext: original, readable message
  - **c** ciphertext: transmitted, unreadable message
  - **k** secret key: known only to Alice and Bob; facilitates **p → c** and **c → p**
  - ***E*** encryption function: ***E*** **(p, k) → c**
  - ***D*** decryption function: ***D*** **(c, k) → p**

Message Confidentiality

- **Confidentiality:** ensure that only **trusted parties** can read the message
- Terminology:
  - p plaintext: original, readable message
  - c ciphertext: transmitted, unreadable message
  - k secret key: known only to Alice and Bob; facilitates **p** → **c** and **c** → **p**
  - *E* encryption function: *E* (p, k) → c
  - *D* decryption function: *D* (c, k) → p

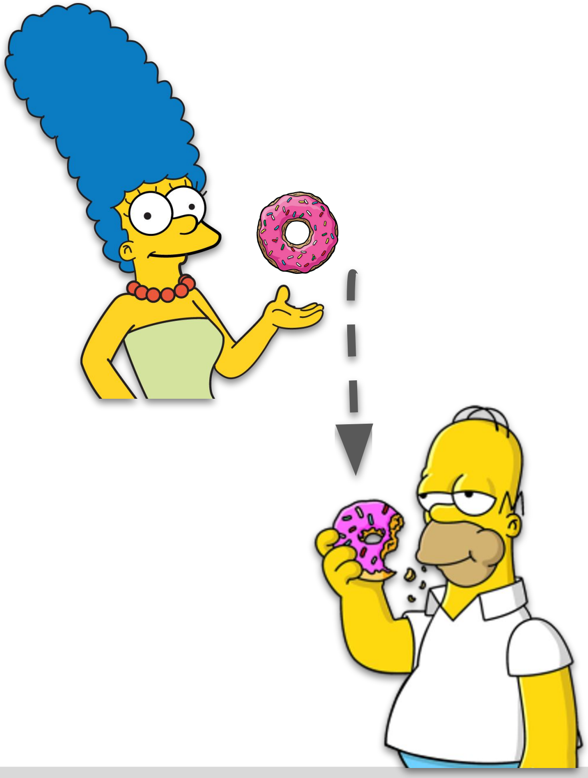
Key-based Encryption Schemes

- **"Symmetric" Key**
  - Encryption and decryption relies on **the same key**
  - Communicating parties must share key **in advance**
  - **Examples: ???**



Key-based Encryption Schemes

- **“Symmetric” Key**
  - Encryption and decryption relies on **the same key**
  - Communicating parties must share key **in advance**
  - **Examples:**
    - Caesar, Vigènere
    - One-time Pad, Stream
    - Transposition ciphers



SKE via Stream Ciphers

- **Stream cipher:** operates on **individual bits** (or bytes); **one at a time**
  - Generates pseudo-random key bits that are **XOR'd** to plaintext bits

Encryption

plaintext =	H	E	L	L	O
	⊕	⊕	⊕	⊕	⊕
key =	A	X	H	J	B
	↓	↓	↓	↓	↓
ciphertext =	K	M	I	V	E

Decryption

ciphertext =	K	M	I	V	E
	⊕	⊕	⊕	⊕	⊕
key =	A	X	H	J	B
	↓	↓	↓	↓	↓
plaintext =	H	E	L	L	O

SKE via Stream Ciphers

- **Stream cipher:** operates on **individual bits** (or bytes); **one at a time**
  - Generates pseudo-random key bits that are **XOR'd** to plaintext bits

Confusion and diffusion?

No diffusion—symbols are not spread around!

Block Ciphers

Block CIpher

- Functions that **encrypts** **fixed-size blocks** with a reusable key
- **Inverse function** **decrypts** when used with **same key**
- The most commonly used encryption approach for confidentiality.

Block Ciphers vs. Hashes

- Hash functions:
  - **???**

Block Ciphers vs. Hashes

- Hash functions:
  - **Must not** have collisions
  - **Must not** be reversible
  - **Goal: integrity**
    - Detect message tampering

Block Ciphers vs. Hashes

- Hash functions:
  - **Must not** have collisions
  - **Must not** be reversible
  - **Goal: integrity**
    - Detect message tampering

- Block Ciphers:
  - **Must not** have collisions
  - **Must be** reversible
  - **Goal: confidentiality**
    - Keep secret message secret

Block Ciphers vs. Hashes

- Hash functions:
  - **Must** **not** have collisions
  - **Must** **not** be reversible
  - **Goal:** **integrity**
    - Detect message tampering
- Block Ciphers:
  - **Must** **not** have collisions
  - **Must** be reversible
  - **Goal:** **confidentiality**
    - Keep secret message secret

A block cipher is not a pseudo-random function

Block Ciphers vs. Hashes

- Hash functions:
  - **Must not** have collisions
  - **Must not** be reversible
  - **Goal: integrity**
    - Detect message tampering
- Block Ciphers:
  - **Must not** have collisions
  - **Must** be reversible
  - **Goal: confidentiality**
    - Keep secret message secret

A block cipher is not a pseudo-random function

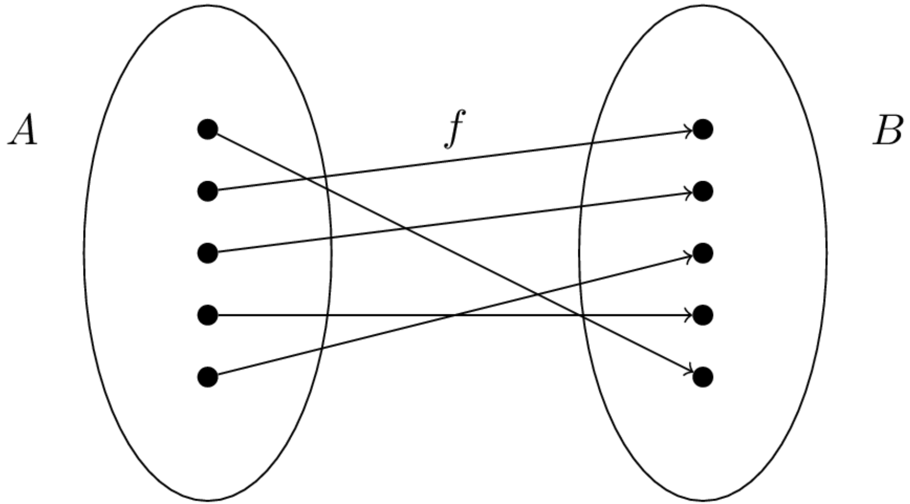
A block cipher is a pseudo-random **permutation**

Pseudo-random Permutation (PRP)

- Defined similarly to a PRF:
  - Practically indistinguishable from a random permutation without secret **k**
- **Main challenge:** design a function that's **invertible**… but only with the **key**
- **Minimal properties of a good block cipher:**
  - Highly nonlinear ("confusion")
  - Mixes input bits together ("diffusion")
  - Dependent on the key

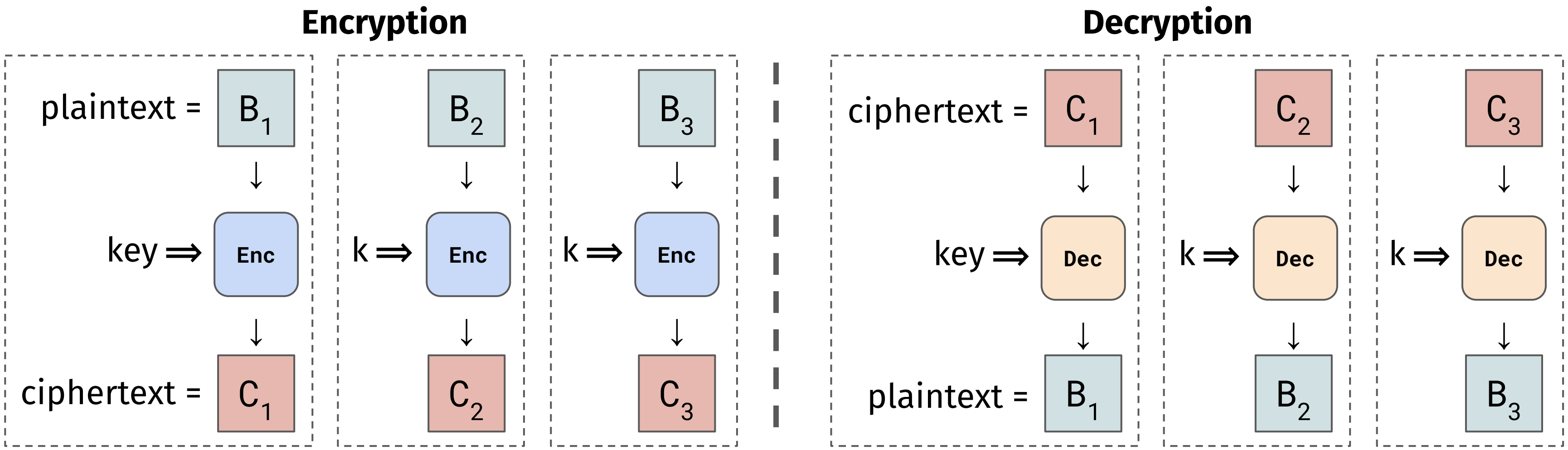
Pseudo-random Permutation (PRP)

- **What we want at a high-level:**
  - Function from **n**-bit input to **n**-bit output
  - Ideally, one bit flip of the input results in **50%** of output bits flipping
  - Distinct inputs yield distinct outputs
  - Thus, an **invertible bijection**



SKE via Block Ciphers

- **Block cipher:** operates on **fixed-length groups** of bits called **blocks**
  - Processes blocks using a **reversible, non-colliding** function



Block vs. Stream Ciphers

- Major categories of SKE
  - **Stream cipher:** operates on **individual bits** (or bytes); **one at a time**
  - **Block cipher:** operates on **fixed-length groups** of bits called **blocks**

- Only a few symmetric methods are used today

Methods	**Year approved**	**Comments**
Data Encryption Standard (**DES**)	1977	1998: EFF's Deep Crack breaks a DES key in 56 hrs
DES-Cipher Block Chaining (**DES-CBC**)		
Triple DES – (**TDES** or **3DES**)	1999	
Advanced Encryption Standard (**AES**)	2001	Among the most used today
Other symmetric encryption methods		
IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish		

Questions?

Data Encryption Standard (DES)

Breaking up long messages into "blocks"

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size…
  - But message might not be **multiples** of block size

Breaking up long messages into "blocks"

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size...
  - But message might not be **multiples** of block size

- **Solution:** Append **padding** to end of message
  - Must be able to recognize and remove padding afterward
  - Common approach: add **n** bytes that have value **n**

Breaking up long messages into "blocks"

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size…
  - But message might not be **multiples** of block size

- **Solution:** Append **padding** to end of message
  - Must be able to recognize and remove padding afterward
  - Common approach: add **n** bytes that have value **n**

- **Challenge:** What if message **terminates** a block?
  - End of message might be misread as padding!

Breaking up long messages into "blocks"

- **Challenge:** How to encrypt longer messages?
  - Can only encrypt in units of cipher block size…
  - But message might not be **multiples** of block size

- **Solution:** Append **padding** to end of message
  - Must be able to recognize and remove padding afterward
  - Common approach: add **n** bytes that have value **n**

- **Challenge:** What if message **terminates** a block?
    - End of message might be misread as padding!

- **Solution:** Append an entire new block of padding

Data Encryption Standard (DES)

- DES is a **block, symmetric encryption** scheme
  - Uses a **64-bit** key
  - Plaintext divided and encrypted as fixed-size, 64-bit blocks
  - Different **modes** of encryption—each with different security implications

Methods	**Year approved**	**Comments**
Data Encryption Standard (**DES**)	1977	1998: EFF's Deep Crack breaks a DES key in 56 hrs
DES-Cipher Block Chaining (**DES-CBC**)		
Triple DES – (**TDES** or **3DES**)	1999	
Advanced Encryption Standard (**AES**)	2001	Among the most used today
Other symmetric encryption methods		
IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish		

Data Encryption Standard (DES)

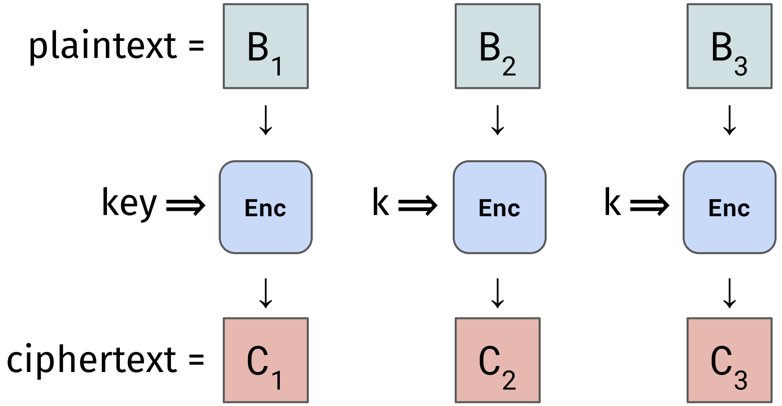
- A variety of **"block cipher modes"** exist today
  - As time went on, researchers found issues with them and proposed better ones
  - We'll talk about a few of these: **Electronic Codebook** and **Cipher Block Chaining**

Methods	Year approved	Comments
Data Encryption Standard (**DES**)	1977	1998: EFF's Deep Crack breaks a DES key in 56 hrs
DES-Cipher Block Chaining (**DES-CBC**)		
Triple DES – (**TDES** or **3DES**)	1999	
Advanced Encryption Standard (**AES**)	2001	Among the most used today
Other symmetric encryption methods		
IDEA (International Data Encryption Algorithm), RC5 (Rivest Cipher 5), CAST (Carlisle Adams Stafford Tavares), Blowfish		

DES Modes: Electronic Codebook

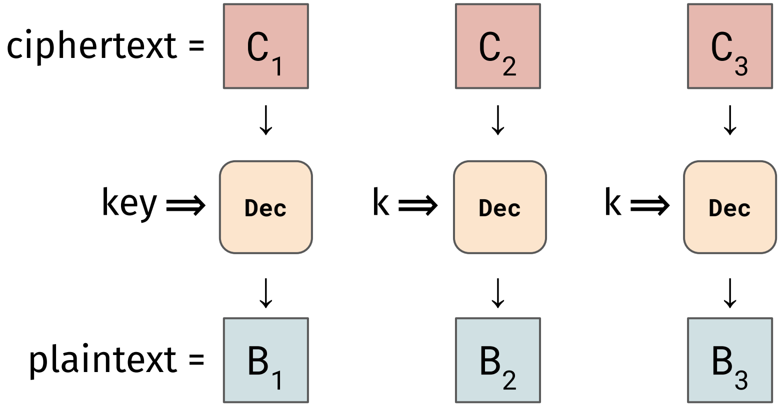
Mode #1: Electronic Codebook (ECB)

- Electronic Codebook (**ECB**)
  - Message divided into code blocks
  - Each block encrypted **separately**



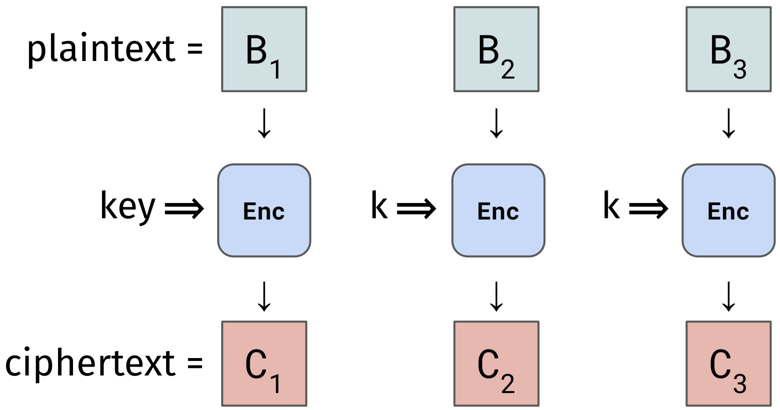
Mode #1: Electronic Codebook (ECB)

- Electronic Codebook (**ECB**)
  - Message divided into code blocks
  - Each block encrypted **separately**; decrypted separately too



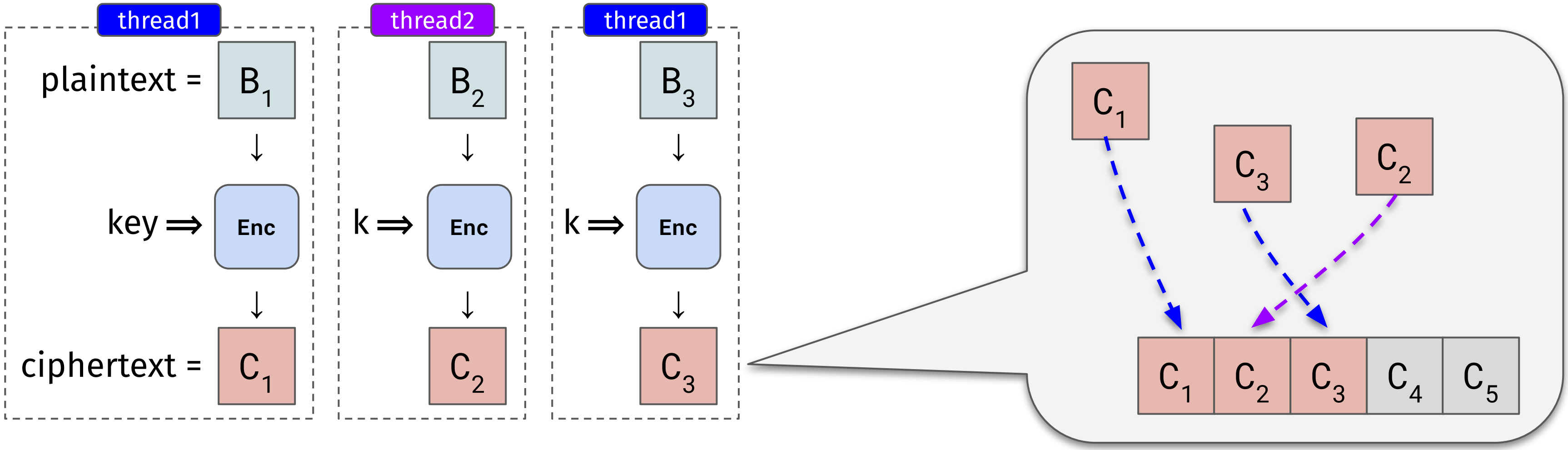
Mode #1: Electronic Codebook (ECB)

- **ECB Strengths:**
  - Construction is **un-chained**
    - Message can be **???**



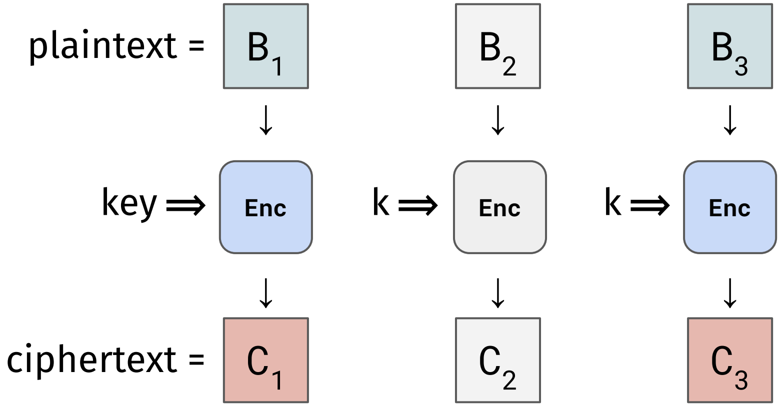
Mode #1: Electronic Codebook (ECB)

- **ECB Strengths:**
  - Construction is **un-chained**
    - Message can be split up and processed in parallel—**fast!**
    - No need to wait on previous block's encryption



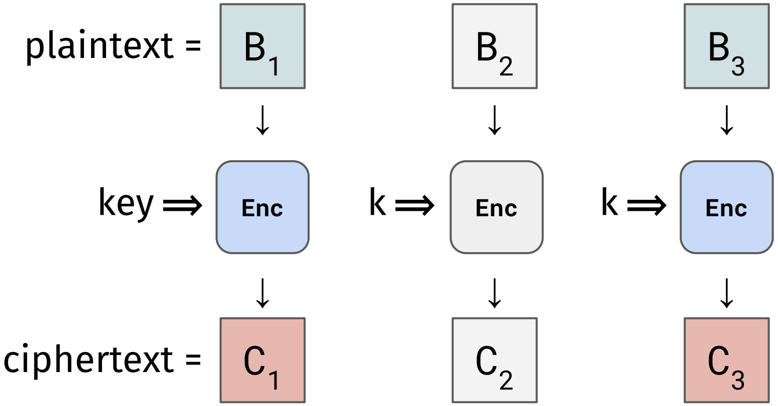
Mode #1: Electronic Codebook (ECB)

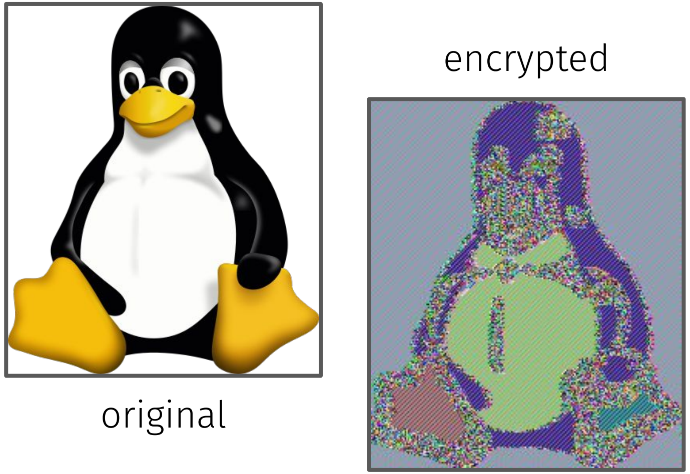
- **ECB Drawbacks:**
  - Identical plaintext blocks produce same ciphertext
    - This results in **low ???**



Mode #1: Electronic Codebook (ECB)

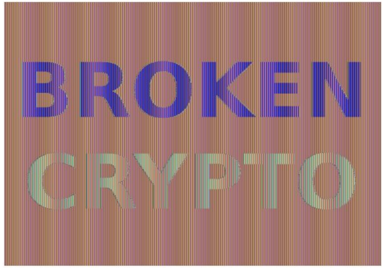
- **ECB Drawbacks:**
  - Identical plaintext blocks produce same ciphertext
    - This results in **low diffusion**



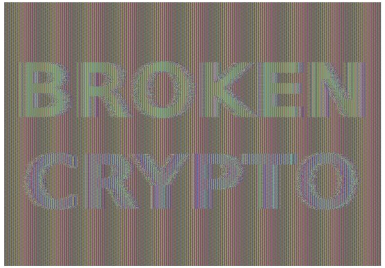


Mode #1: Electronic Codebook (ECB)

- **ECB Drawbacks:**
  - Do **larger block sizes** increase diffusion?
    - Yes—but **at what cost ???**



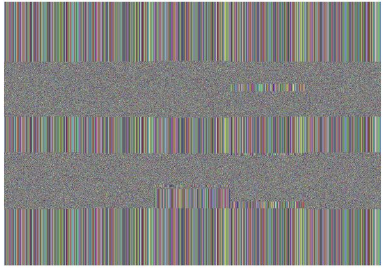
(b) ECB mode ciphertext, 5 pixel (120 bit) block size.



(c) ECB mode ciphertext, 30 pixel (720 bit) block size.

(a) Plaintext image, 2000 by 1400 pixels, 24 bit color depth.

(d) ECB mode ciphertext, 100 pixel (2400 bit) block size.

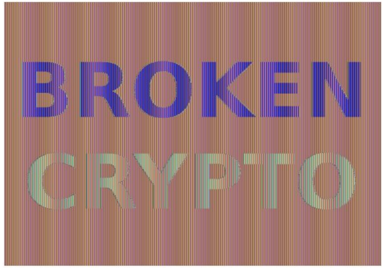


(e) ECB mode ciphertext, 400 pixel (9600 bit) block size.

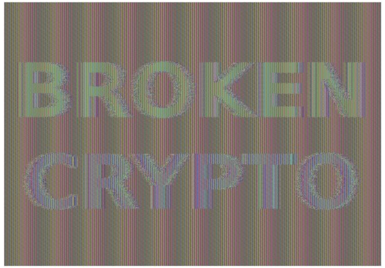
Mode #1: Electronic Codebook (ECB)

- **ECB Drawbacks:**
  - Do **larger block sizes** increase diffusion?
    - Yes—but **at what cost**
      - **Much more impractical**
      - E.g., higher memory footprint

(a) Plaintext image, 2000 by 1400 pixels, 24 bit color depth.

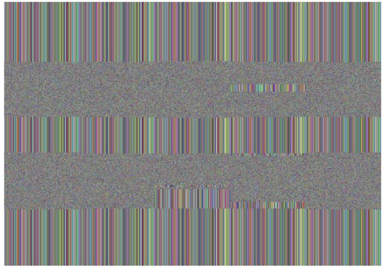


(b) ECB mode ciphertext, 5 pixel (120 bit) block size.



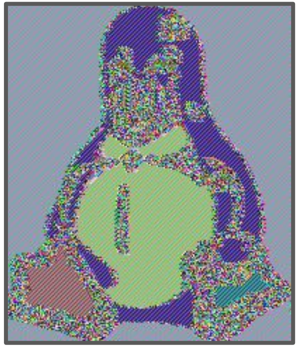
(c) ECB mode ciphertext, 30 pixel (720 bit) block size.

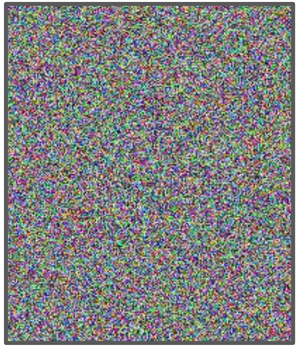
(d) ECB mode ciphertext, 100 pixel (2400 bit) block size.



(e) ECB mode ciphertext, 400 pixel (9600 bit) block size.

How can we increase diffusion?

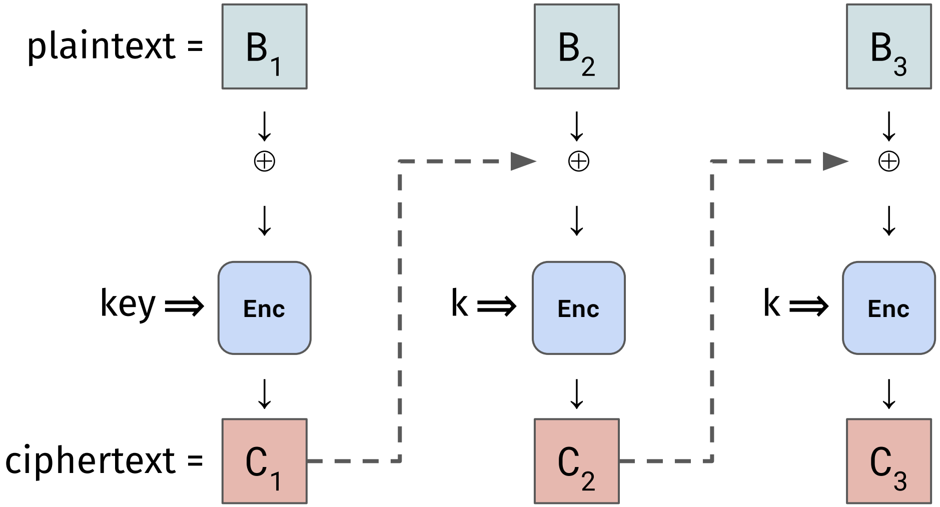




DES Modes: Cipher Block Chaining

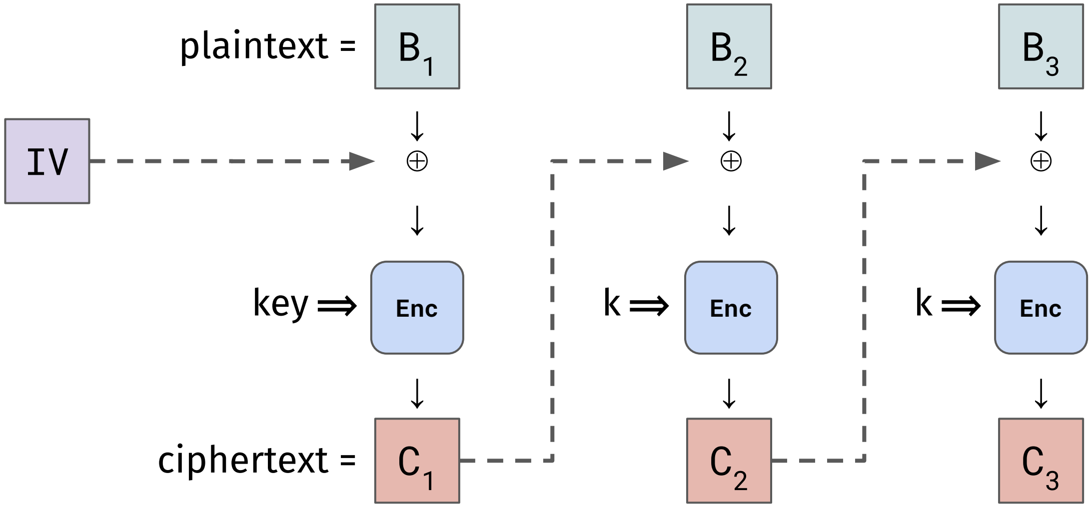
Mode #2: Cipher Block Chaining (CBC)

- **Key idea:** seed current block with **ciphertext from the previous block**



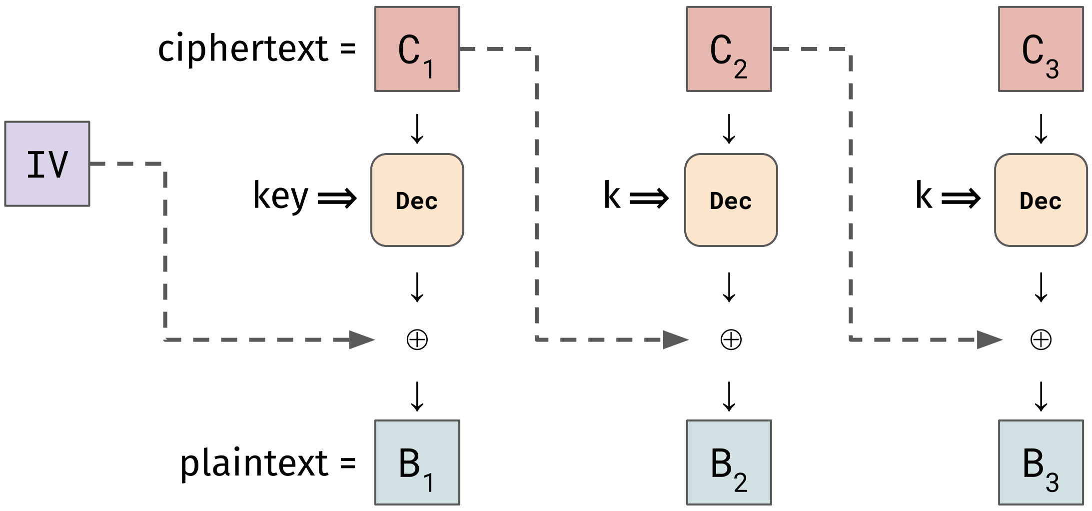
Mode #2: Cipher Block Chaining (CBC)

- **Key idea:** seed current block with **ciphertext from the previous block**
  - Since first block has no "previous" cipher, seed it with a 64-bit initialization vector (I.V.)
    - A **random or pseudo-random** block that's unpredictable



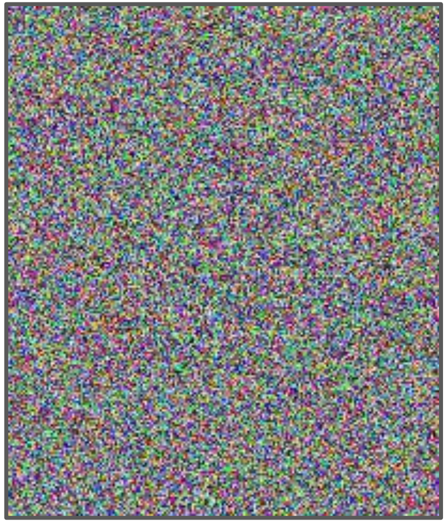
Mode #2: Cipher Block Chaining (CBC)

- **Decryption** operates similarly:



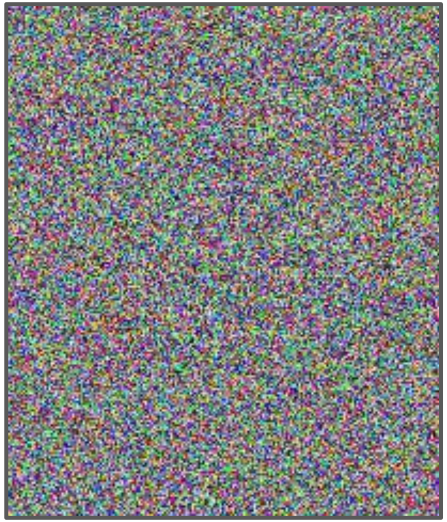
Mode #2: Cipher Block Chaining (CBC)

- **CBC Strengths:**
  - Chained construction far stronger than ECB
    - **More diffusion!**
    - Negates ECB's need for super-large blocks



Mode #2: Cipher Block Chaining (CBC)

- **CBC Strengths:**
  - Chained construction far stronger than ECB
    - **More diffusion!**
    - Negates ECB's need for super-large blocks

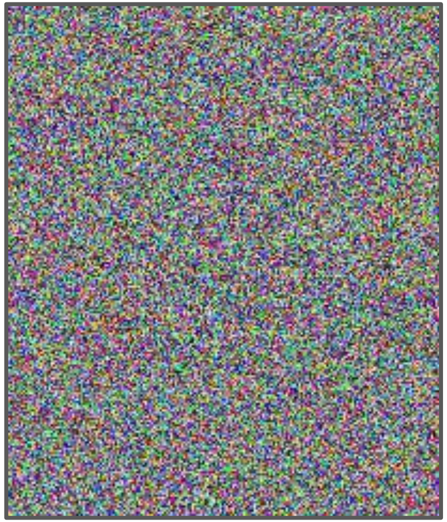


- **CBC Drawbacks:**
  - Completely sequential
    - ???

Mode #2: Cipher Block Chaining (CBC)

- **CBC Strengths:**
  - Chained construction far stronger than ECB
    - **More diffusion!**
    - Negates ECB's need for super-large blocks

- **CBC Drawbacks:**
  - Completely sequential
    - **Cannot be parallelized!**
    - No leveraging advances in multi-threading etc.



Questions?

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES)

- Today's most common block cipher
  - Designed by NIST competition, with a very long public discussion
  - Widely believed to be secure… but we don't know how to prove it
- Variable **key size**:
  - 128-bit fairly common; also 192-bit and 256-bit versions
- Input message is split into **128-bit blocks**
- Ten **rounds**:
  - Split **k** into ten subkeys (key scheduling)
  - Performs set of identical operations ten times (each with different subkey)

AES Cliff Notes

- Systematically designed through a read/blue team competition by NIST
  - Layered design to remove flaws of individual components
  - Prevent statistical leakage
    - Letter frequency of substitution ciphers
    - Anagrams of transposition ciphers

- **Many fancier "modes" with ordering counters, etc.**
  - Efficient software and hardware implementations

- Exposes security performance tradeoff to user
  - 128-bit key: 10 rounds
  - 192-bit key: 12 rounds
  - 256-bit key: 14 rounds

Disclaimer: details are hairy—don't worry about them.

Secure Channels

Building a Secure Channel

- What if you want **confidentiality** and **integrity** at **the same time**?
  - Which would you perform **first**: encrypting or hashing? And why?

Which would you perform first?

Encrypt (Confidentiality) first

0%

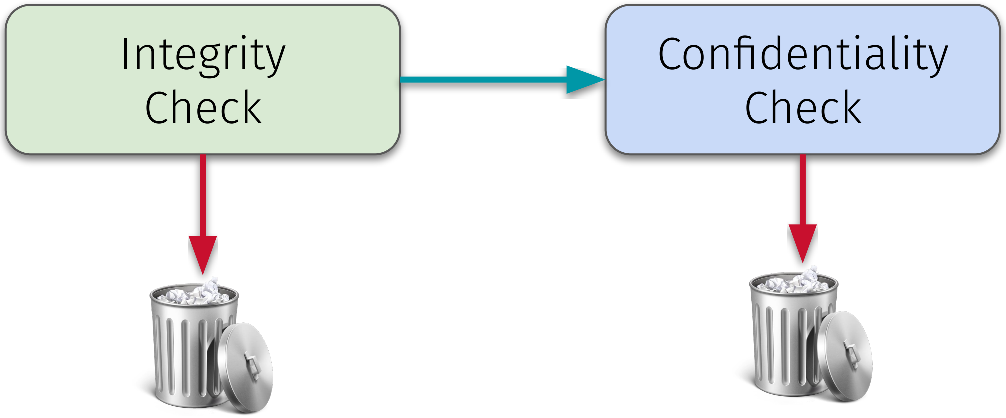
Hash (Integrity) first

0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Building a Secure Channel

- What if you want **confidentiality** and **integrity** at **the same time**?
  - Which would you perform **first**: encrypting or hashing? And why?



Limitations of Symmetric Crypto

- Complex mathematics
  - Hardware and software efficiency is key
  - A huge study of modern cryptography research

- Requires pre-shared keys
  - The keys need to **stay secret always**

Limitations of Symmetric Crypto

- Complex mathematics
  - Hardware and software efficiency is key
  - A huge study of modern cryptography research
- Requires pr
  - The keys

Amazing fact: Alice and Bob can have a **public** conversation to derive a shared **secret** key

Next time on CS 4440...

Public-key Encryption, Signatures