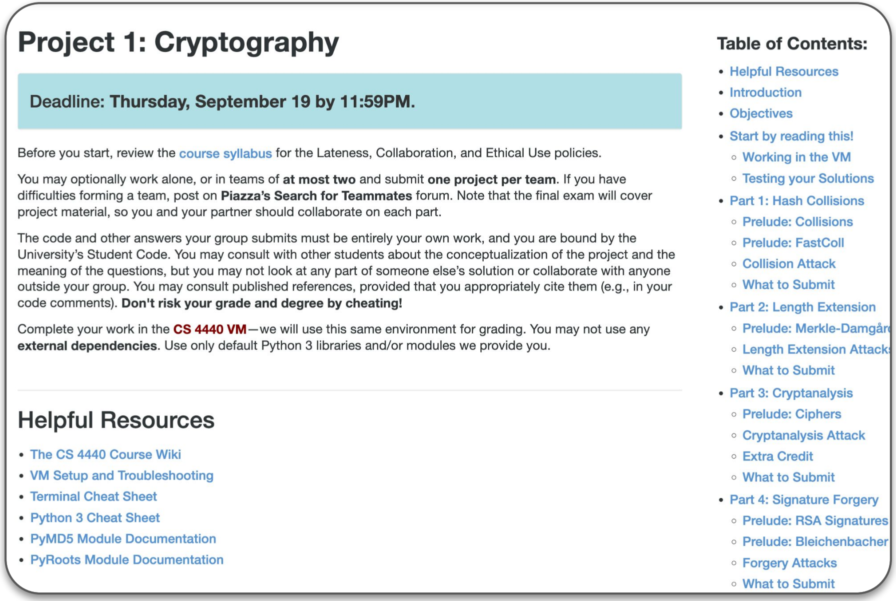
Week 3: Lecture A

Improved Cipher Designs

Tuesday, September 3, 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!**

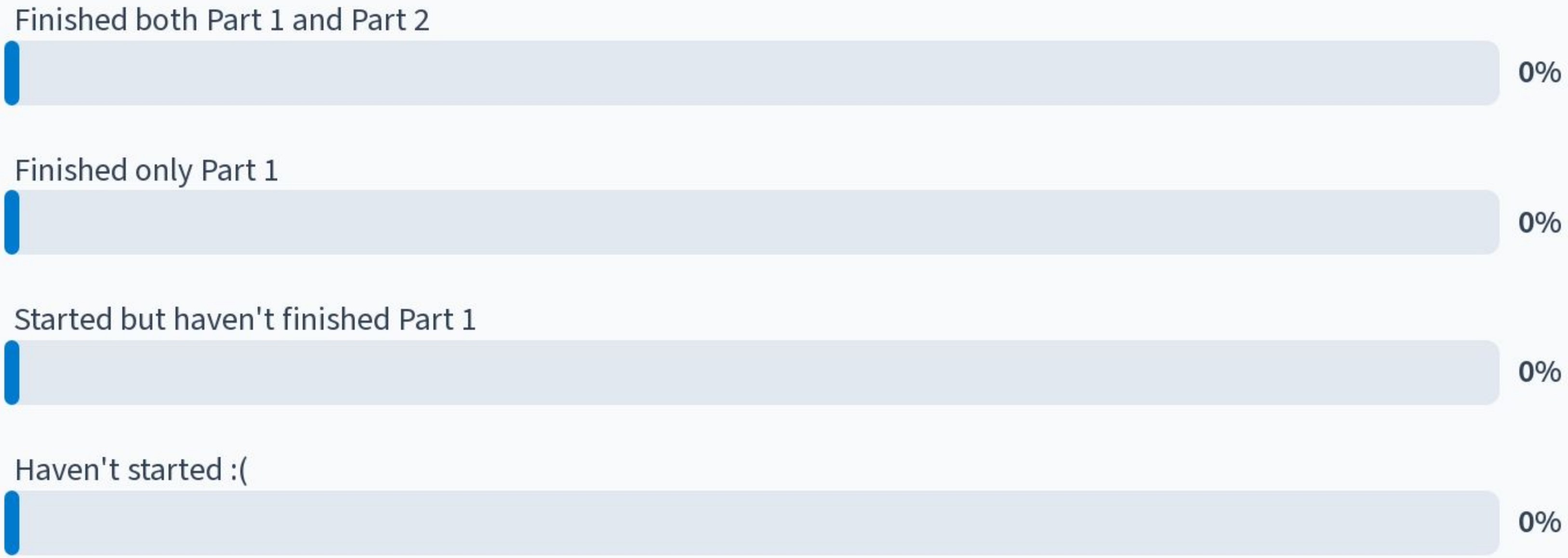
Complete your work in the **CS 4440 VM**—we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.

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:

Progress on Project 1



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

Project Tips

- Projects are challenging—**you're performing real-world attacks!**
  - Build off of lecture concepts
  - Make sure you understand the lectures
  - Prepare you to defend **in the real world**

- **Suggested strategy: get high-level idea down, then start implementing**
  1. Go through assignment and start sketching-out your approach
  2. **Come to Office Hours and ask if you're on the right track!**
  3. Then start building your program

- Don't get discouraged—**we are here to help!**
  - Most issues are cleared up in a few minutes of white-boarding

Announcements

See Discord for meeting info!

utahsec.cs.utah.edu

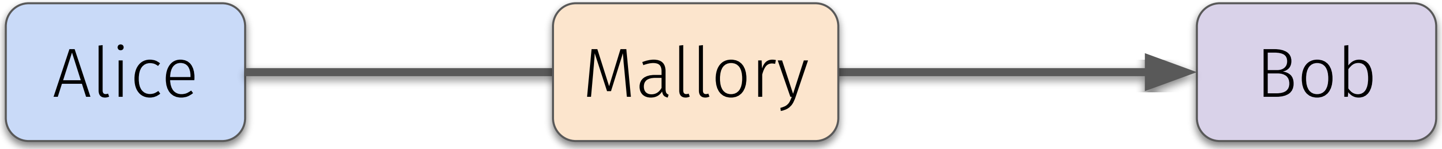
Questions?

Last time on CS 4440…

Message Confidentiality
Substitution Ciphers
Frequency Cryptanalysis

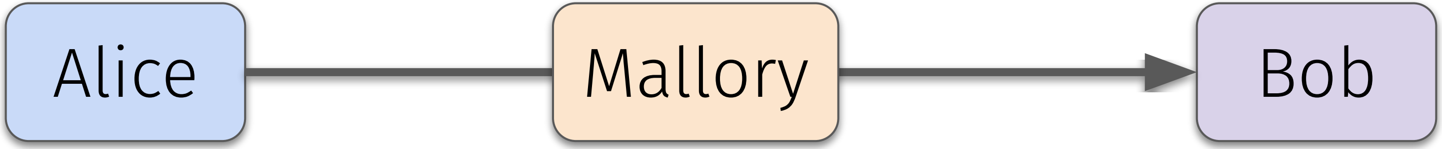
Message Confidentiality

- **Confidentiality: ???**



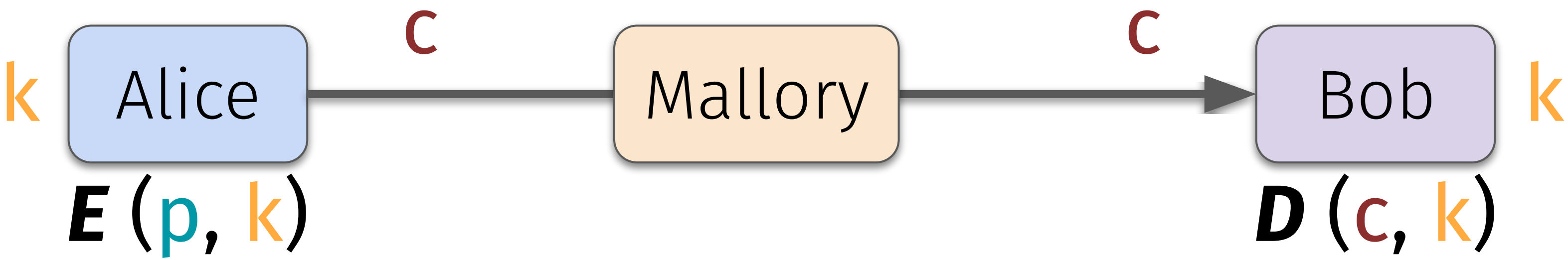
Message Confidentiality

- **Confidentiality:** ensure that only **trusted parties** can read the message
- Terminology: **???**



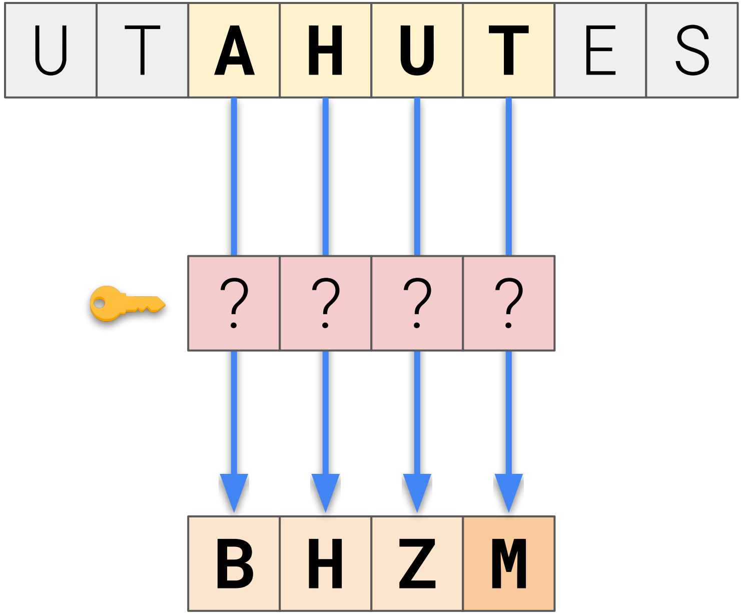
Message Confidentiality

- **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



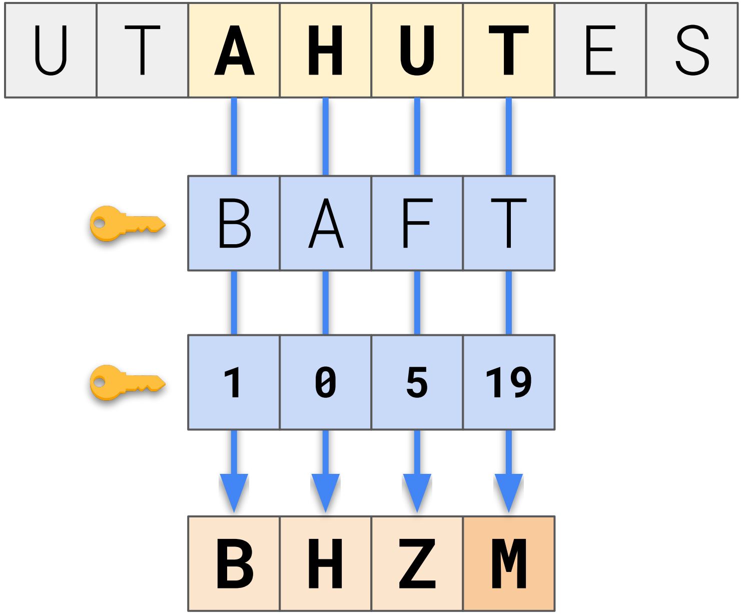
Confidentiality via Ciphers

- We define a key as **???**



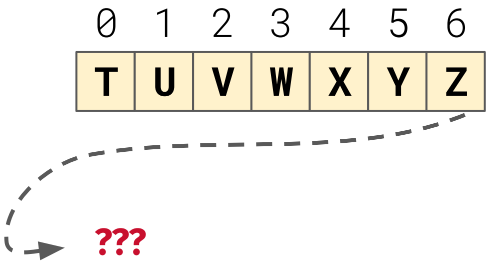
Confidentiality via Ciphers

- We define a key as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet



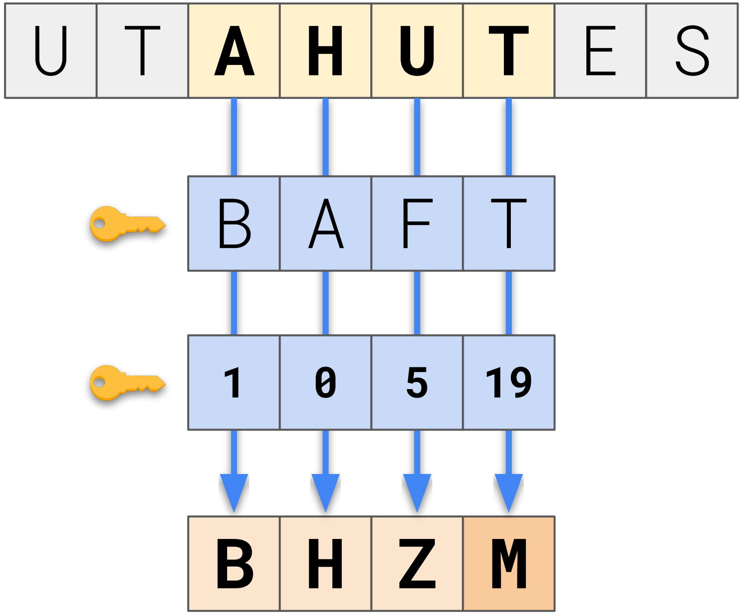
Confidentiality via Ciphers

- We define a key as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet
- Shift goes past end of alphabet?



0	1	2	3	4	5	6
T	U	V	W	X	Y	Z

???

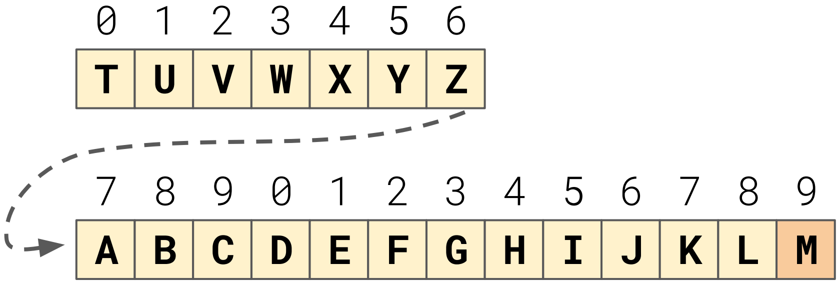


U	T	A	H	U	T	E	S

B	A	F	T
1	0	5	19
B	H	Z	M

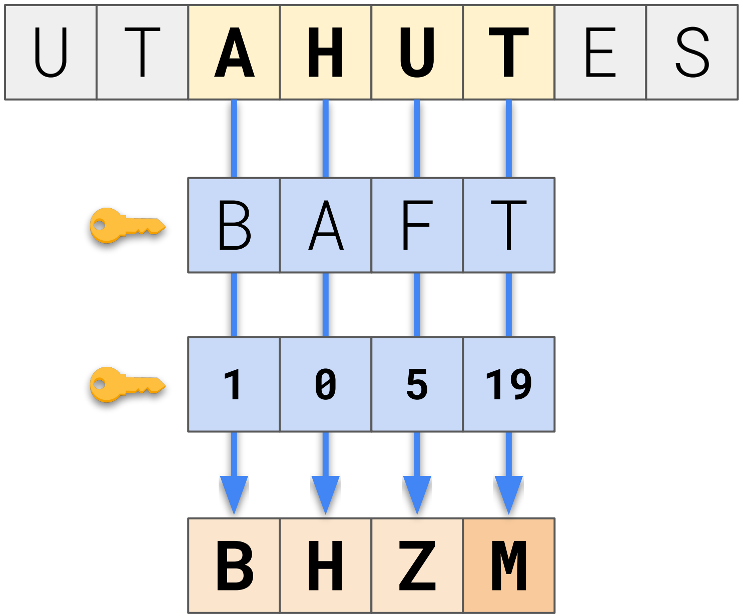
Confidentiality via Ciphers

- We define a key as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet
- Shift goes past end of alphabet?
  - **Wrap around** to beginning!



0	1	2	3	4	5	6
T	U	V	W	X	Y	Z

7	8	9	0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J	K	L	M



Caesar Ciphers

- Really old school cryptography
  - First recorded use: Julius Caesar (100–144 B.C.)
- Replaces each plaintext letter with **???**

Caesar Ciphers

- Really old school cryptography
  - First recorded use: Julius Caesar (100–144 B.C.)

- Replaces each plaintext letter with one a fixed number of places down the alphabet
  - Encryption: $\mathbf{c_i := (p_i + k) \bmod 26}$
  - Decryption: $\mathbf{p_i := (c_i - k) \bmod 26}$

- Example for k = 3:
  - Plain: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
  - +Shift: `33333333333333333333333333`
  - =Cipher: `DEFGHIJKLMNOPQRSTUVWXYZABC`

  - Plain: `go utes beat wash st`
  - +Key: `33 3333 3333 3333 33`
  - =Cipher: `jr xwhv ehdw zdvk vw`

Caesar Cipher Cryptanalysis

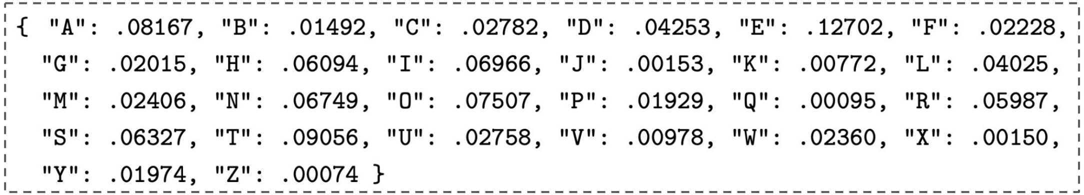
Brute-forcing every possible key

Cryptanalysis

Caesar Cryptanalysis via Chi-Square Test

Example ciphertext string (with a **zero reverse shift**): LJSGUKJYSEKDLJGGAKWOGLHWLJNWFZLVEX

Expected English language letter frequencies:



```
{ "A": .08167, "B": .01492, "C": .02782, "D": .04253, "E": .12702, "F": .02228,
  "G": .02015, "H": .06094, "I": .06966, "J": .00153, "K": .00772, "L": .04025,
  "M": .02406, "N": .06749, "O": .07507, "P": .01929, "Q": .00095, "R": .05987,
  "S": .06327, "T": .09056, "U": .02758, "V": .00978, "W": .02360, "X": .00150,
  "Y": .01974, "Z": .00074 }
```

Caesar Cryptanalysis via Chi-Square Test

Example ciphertext string (with a **zero reverse shift**): LJSGUKJYSEKDLJGGAKWOGLHWLJNWFZLVEX

Expected English language letter frequencies:

```
{ "A": .08167, "B": .01492, "C": .02782, "D": .04253, "E": .12702, "F": .02228,
  "G": .02015, "H": .06094, "I": .06966, "J": .00153, "K": .00772, "L": .04025,
  "M": .02406, "N": .06749, "O": .07507, "P": .01929, "Q": .00095, "R": .05987,
  "S": .06327, "T": .09056, "U": .02758, "V": .00978, "W": .02360, "X": .00150,
  "Y": .01974, "Z": .00074 }
```

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}$$

$\mathbf{O_L}$ = observed count for letter ‘L’ = **5.0**

$\mathbf{E_L}$ = expected count for letter ‘L’

= **EnglishFreq_L** * **StringLength**

= **0.04025** * **34**

= **1.3685**

$$\mathbf{X^2_L} = (\mathbf{5.0} - \mathbf{1.3685})^2 \,/\, \mathbf{1.3685}$$

$$= \mathbf{9.6367}$$

Caesar Cryptanalysis via Chi-Square Test

Example ciphertext string (with a **zero reverse shift**): LJSGUKJYSEKDLJGGAKWOGLHWLJNWFZLVEX

Expected English language letter frequencies:

```
{ "A": .08167, "B": .01492, "C": .02782, "D": .04253, "E": .12702, "F": .02228,
  "G": .02015, "H": .06094, "I": .06966, "J": .00153, "K": .00772, "L": .04025,
  "M": .02406, "N": .06749, "O": .07507, "P": .01929, "Q": .00095, "R": .05987,
  "S": .06327, "T": .09056, "U": .02758, "V": .00978, "W": .02360, "X": .00150,
  "Y": .01974, "Z": .00074 }
```

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}$$

$\mathbf{O_L}$ = observed count for letter ‘L’ = **5.0**

$\mathbf{E_L}$ = expected count for letter ‘L’

= **EnglishFreq$_L$** * **StringLength**

= **0.04025** * **34**

= **1.3685**

$$\mathbf{X^2_L} = (\mathbf{5.0} - \mathbf{1.3685})^2 \,/\, \mathbf{1.3685}$$

$$= \mathbf{9.6367}$$

1. Add X^2 scores for all 26 alphabet letters
2. Final sum = **that reverse shift's X^2 score**
3. Repeat for the 25 other reverse shifts
4. **Lowest score = the correct reverse shift**
5. Mapped as forward shift = **the key letter**

Vigènere Ciphers

- First described by Bellaso in 1553
  - Later misattributed to Vigènere
- Encrypts successive letters via **???**

Vigènere Ciphers

- First described by Bellaso in 1553
  - Later misattributed to Vigènere
- Encrypts successive letters via **sequence of Caesar ciphers** determined by the letters of a keyword
- For an **n**-letter keyword **k** ...
  - Encryption: $\mathbf{c_i := (p_i + k_{i \bmod n}) \bmod 26}$
  - Decryption: $\mathbf{p_i := (c_i - k_{i \bmod n}) \bmod 26}$
- Example for k = ABC (i.e., $\mathbf{k_0 = 0, k_1 = 1, k_2 = 2}$)
  - Plain: `bbbbbb amazon`
  - +Key: `012012 012012`
  - =Cipher: `bcdbcd anczpp`

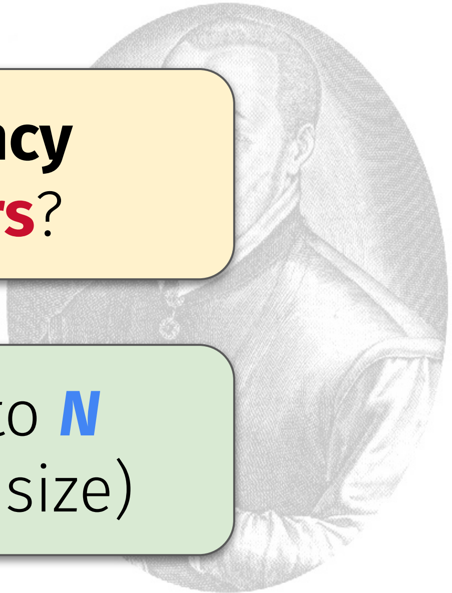
Vigènere Ciphers

- First described by Bellaso in 1553
  - Later misattributed to Vigènere
- Encrypts
  ciphers c
- For an **n**-letter keyword **k** ...
  - Encryption: c_i := (p_i + $k_{i \bmod n}$) mod 26
  - Decryption: p_i := (c_i - $k_{i \bmod n}$) mod 26
- Example for k = ABC (i.e., $k_0 = 0$, $k_1 = 1$, $k_2 = 2$)
  - Plain: bbbbbb amazon
  - +Key: 012012 012012
  - =Cipher: bcdbcd anczpp

Can we still perform **frequency analysis** for **Vigenere ciphers**?

Vigènere Ciphers

- First described by Bellaso in 1553
  - Later misattributed to Vigènere
- Encrypts
  ciphers c
- For an **n**-letter keyword **k** ...
  - Encryption:
  - Decry
- Example
  - Plain
  - +Key: 012012 012012
  - =Cipher: bcdbcd anczpp



Can we still perform **frequency analysis** for **Vigenere ciphers**?

Yes—just partition it down into ***N*** **Caesar ciphers** (where ***N*** = key size)

Finding Key Size via Kasiski Method

- Example:

p	**THERE**	**ARE**TW	O**WAY**S	OFCON	STRUC	TIN**GA**	**S**OFTW	**ARE**DE	SIGNO	NE**WAY**
🔑	**SYSTE**	**MSY**ST	E**MSY**S	TEMSY	STEMS	YST**EM**	**S**YSTE	**MSY**ST	EMSYS	TE**MSY**
c	**LFWKI**	**MJC**LP	S**ISW**K	HJOGL	KMVGU	RAG**KM**	**K**MXMA	**MJC**VX	WUYLG	GI**ISW**
p	ISTOM	AKEIT	**SOS**IM	PLETH	AT**THE**	**REARE**	OBVIO	USLYN	ODEFI	CIEN**C**
🔑	STEMS	YSTEM	**SYS**TE	MSYST	EM**SYS**	**TEMSY**	STEMS	YSTEM	SYSTE	MSYS**T**
c	ALXAE	YCXMF	**KMK**BQ	BDCLA	EF**LFW**	**KIMJC**	GUZUG	SKECZ	GBWYM	OACF**V**
p	**IE**SAN	DTH**EO**	**TH**ERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THAT**T**	**HEREA**
🔑	**EM**SYS	TEM**SY**	**ST**EMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEM**S**	**YSTEM**
c	**MQ**KYF	WXT**WM**	**LA**IDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEF**L**	**FWKIM**
p	**RE**NOO	BVIOU	SDEFI	**CIE**NC	IESTH	EFIRS	TM**ETH**	**O**DISF	ARMOR	EDIFF
🔑	**SY**STE	MSYST	EMSYS	**TEM**SY	STEMS	YSTEM	SY**STE**	**M**SYST	EMSYS	TEMSY
c	**JC**FHS	NNGGN	WPWDA	**VMQ**FA	AXWFZ	CXBVE	LK**WML**	**A**VGKY	EDEMJ	XHUXD

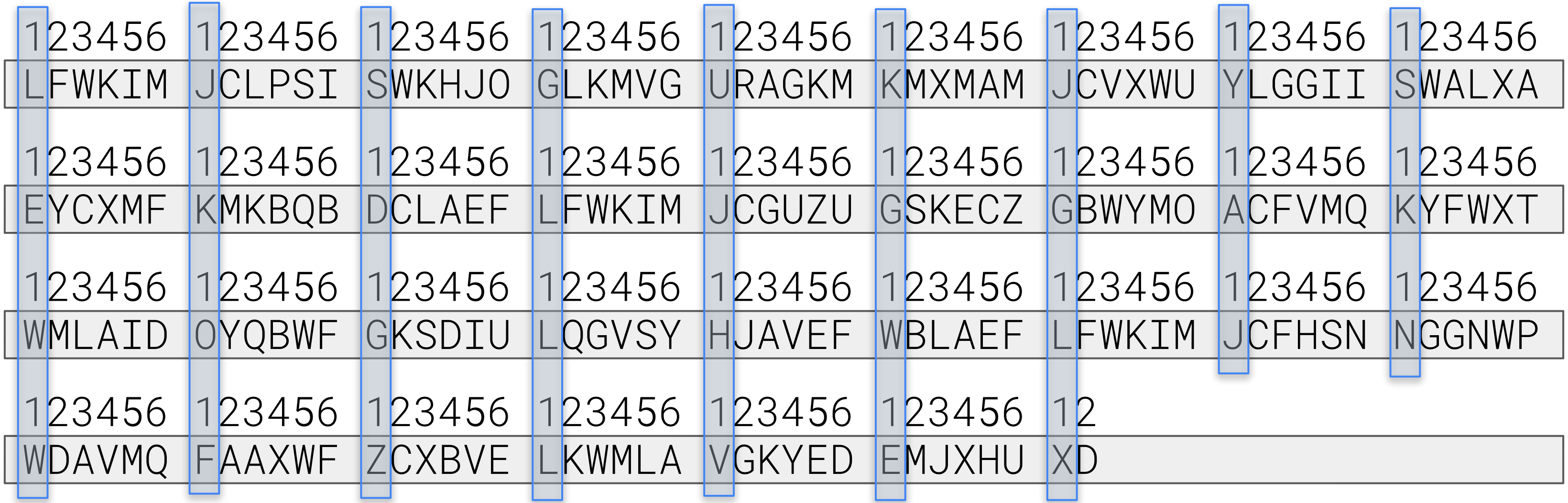
Finding Key Size via Kasiski Method

- Pick **realistic key lengths**; a length of two or three is probably short

Dist.	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
74	x																		
72	x	x	x		x		x	x									x		
66	x	x			x					x									
36	x	x	x		x			x									x		
32	x		x				x								x				
30	x	x		x	x				x					x					

Finding Key Size via Kasiski Method

- Then, **group letters by columns**—they received equal shifts!



```
123456 123456 123456 123456 123456 123456 123456 123456 123456
LFWKIM JCLPSI SWKHJO GLKMVG URAGKM KMXMAM JCVXWU YLGGII SWALXA

123456 123456 123456 123456 123456 123456 123456 123456 123456
EYCXMF KMKBQB DCLAEF LFWKIM JCGUZU GSKECZ GBWYMO ACFVMQ KYFWXT

123456 123456 123456 123456 123456 123456 123456 123456 123456
WMLAID OYQBWF GKSDIU LQGVSY HJAVEF WBLAEF LFWKIM JCFHSN NGGNWP

123456 123456 123456 123456 123456 123456 12
WDAVMQ FAAXWF ZCXBVE LKWMLA VGKYED EMJXHU XD
```

Recap: Breaking Vigènere

1. ???

Recap: Breaking Vigènere

1. Identify the key length:
    - **Project 1:** keys will always be of length **eight**
    - **Extra Credit:** key varies—use **Kasiski method**!

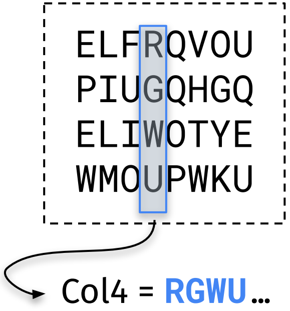
2. **???**

Recap: Breaking Vigènere

1. Identify the key length:
   - **Project 1:** keys will always be of length **eight**
   - **Extra Credit:** key varies—use **Kasiski method**!

2. Divide ciphertext into ***N*** columns:
   - **Why?** Because Vigènere uses a repeating key
   - Vigènere cipher is a set of ***N*** Caesar ciphers

3. **???**



```
ELFRQVOU
PIUGQHGQ
ELIWOTYE
WMOUPWKU
```

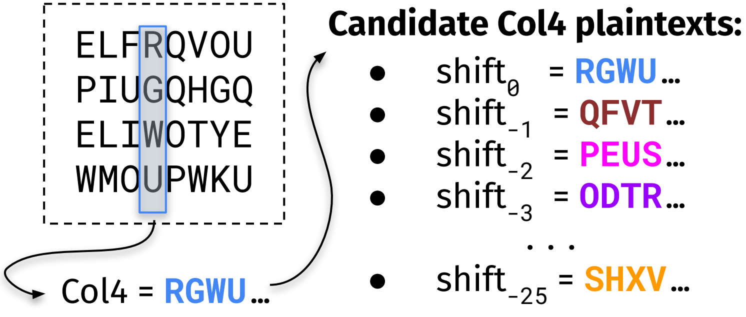
Col4 = **RGWU**…

Recap: Breaking Vigènere

1. Identify the key length:
   - **Project 1:** keys will always be of length **eight**
   - **Extra Credit:** key varies—use **Kasiski method**!

2. Divide ciphertext into ***N*** columns:
   - **Why?** Because Vigènere uses a repeating key
   - Vigènere cipher is a set of ***N*** Caesar ciphers

3. Perform cryptanalysis on each column:
   - Find all candidate **reverse shifts** per column



```
ELFRQVOU
PIUGQHGQ
ELIWOTYE
WMOUPWKU
```

Col4 = **RGWU…**

Candidate Col4 plaintexts:

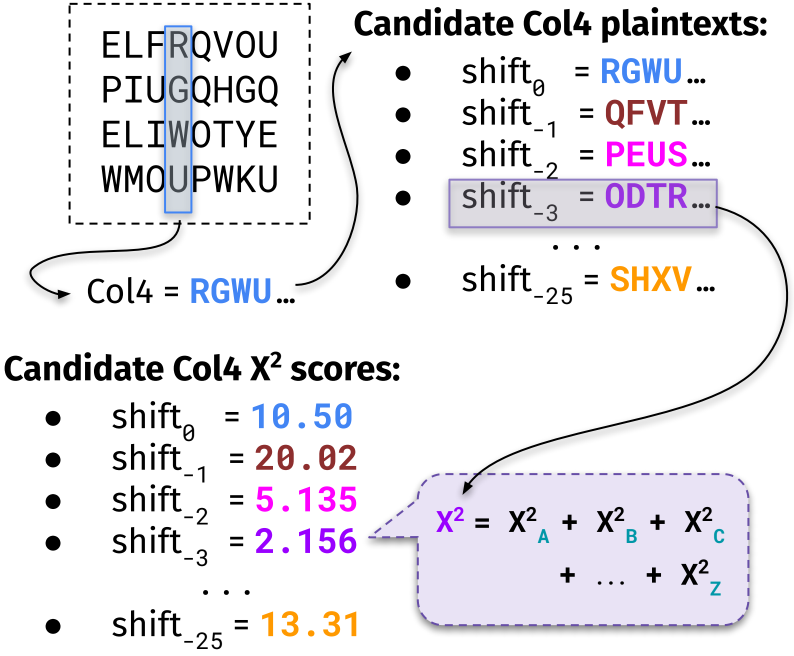
- shift$_0$ = **RGWU…**
- shift$_{-1}$ = **QFVT…**
- shift$_{-2}$ = **PEUS…**
- shift$_{-3}$ = **ODTR…**

. . .

- shift$_{-25}$ = **SHXV…**

Recap: Breaking Vigènere

1. Identify the key length:
   - **Project 1:** keys will always be of length **eight**
   - **Extra Credit:** key varies—use **Kasiski method**!
2. Divide ciphertext into ***N*** columns:
   - **Why?** Because Vigènere uses a repeating key
   - Vigènere cipher is a set of ***N*** Caesar ciphers
3. Perform cryptanalysis on each column:
   - Find all candidate **reverse shifts** per column
   - **Chi-square test:** find **best-fit reverse shift**



```
ELFRQVOU
PIUGQHGQ
ELIWOTYE
WMOUPWKU
```

Col4 = **RGWU…**

Candidate Col4 plaintexts:

- shift_0 = **RGWU…**
- shift_{-1} = **QFVT…**
- shift_{-2} = **PEUS…**
- shift_{-3} = **ODTR…**

. . .

- shift_{-25} = **SHXV…**

Candidate Col4 X^2 scores:

- shift_0 = 10.50
- shift_{-1} = 20.02
- shift_{-2} = 5.135
- shift_{-3} = 2.156

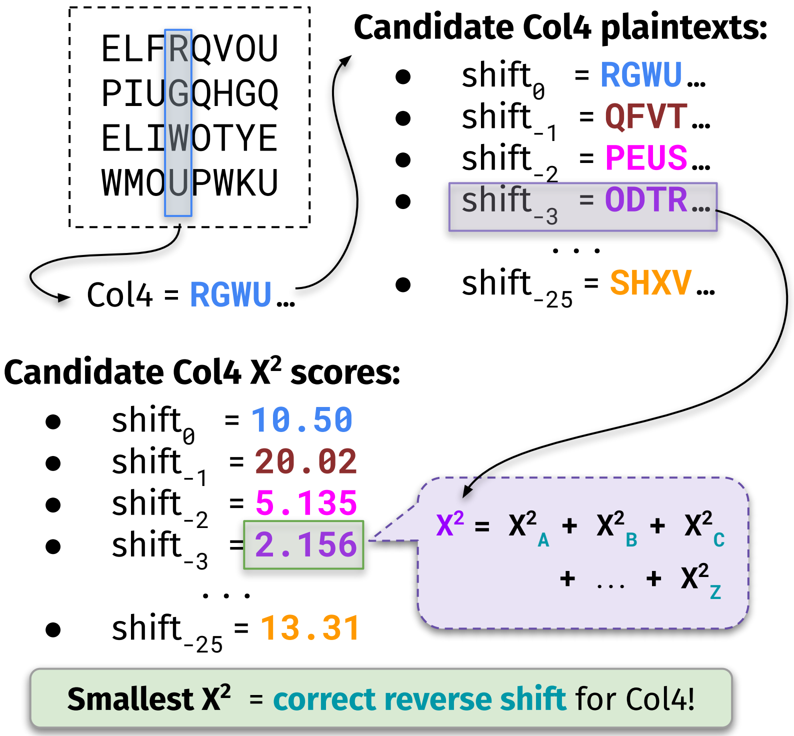
. . .

- shift_{-25} = 13.31

$$X^2 = X^2_A + X^2_B + X^2_C + \ldots + X^2_Z$$

Recap: Breaking Vigènere

1. Identify the key length:
   - **Project 1:** keys will always be of length **eight**
   - **Extra Credit:** key varies—use **Kasiski method**!
2. Divide ciphertext into ***N*** columns:
   - **Why?** Because Vigènere uses a repeating key
   - Vigènere cipher is a set of ***N*** Caesar ciphers
3. Perform cryptanalysis on each column:
   - Find all candidate **reverse shifts** per column
   - **Chi-square test:** find **best-fit reverse shift**
   - Compute forward shift = **column's key letter**



```
ELFRQVOU
PIUGQHGQ
ELIWOTYE
WMOUPWKU
```

Col4 = RGWU…

Candidate Col4 plaintexts:

- shift_0 = RGWU…
- shift_{-1} = QFVT…
- shift_{-2} = PEUS…
- shift_{-3} = ODTR…

. . .

- shift_{-25} = SHXV…

Candidate Col4 X^2 scores:

- shift_0 = 10.50
- shift_{-1} = 20.02
- shift_{-2} = 5.135
- shift_{-3} = 2.156

. . .

- shift_{-25} = 13.31

$$X^2 = X^2_A + X^2_B + X^2_C + \ldots + X^2_Z$$

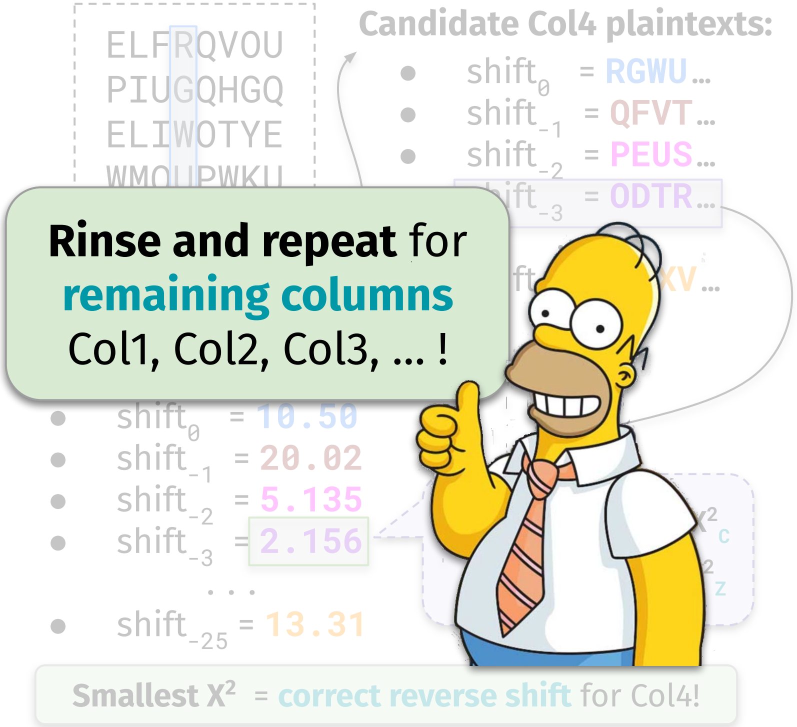
Smallest X^2 = **correct reverse shift** for Col4!

Recap: Breaking Vigènere

1. Identify the key length:
   - **Project 1:** keys will always be of length **eight**
   - **Extra Credit:** key varies—use **Kasiski method**!

2. Divide ciphertext into ***N*** columns:
   - **Why?** Because Vigènere uses a repeating key
   - Vigènere cipher is a set of ***N*** Caesar ciphers

3. Perform cryptanalysis on each column:
   - Find all candidate **reverse shifts** per column
   - **Chi-square test:** find **best-fit reverse shift**
   - Compute forward shift = **column's key letter**
   - Assemble all ***N*** column keys = **the Vigènere key!**



ELFRQVOU
PIUGQHGQ
ELIWOTYE
WMOUPWKU

Candidate Col4 plaintexts:

- shift_0 = RGWU...
- shift_{-1} = QFVT...
- shift_{-2} = PEUS...
- shift_{-3} = ODTR...

...XV...

- shift_0 = 10.50
- shift_{-1} = 20.02
- shift_{-2} = 5.135
- shift_{-3} = 2.156

. . .

- shift_{-25} = 13.31

Smallest X^2 = correct reverse shift for Col4!

Rinse and repeat for **remaining columns** Col1, Col2, Col3, ... !

Questions?

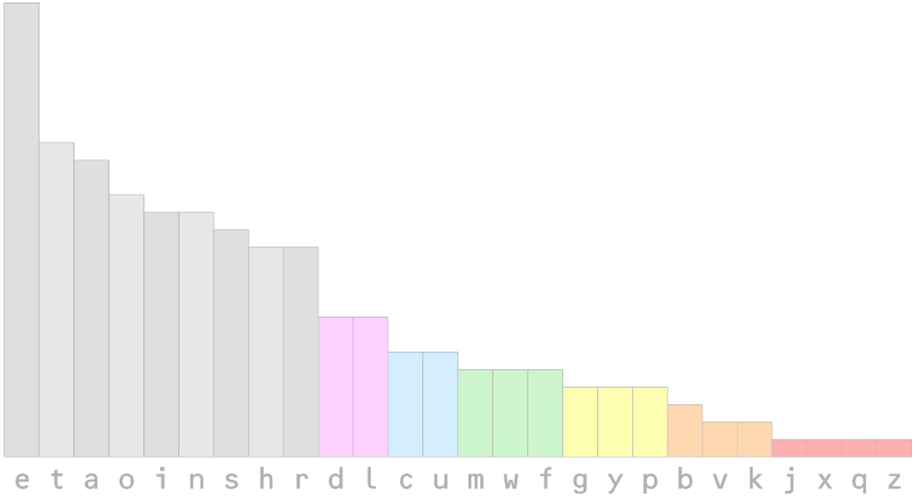
This time on CS 4440…

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

Pseudo-random Keys

Recap: Confidentiality via Substitution Ciphers

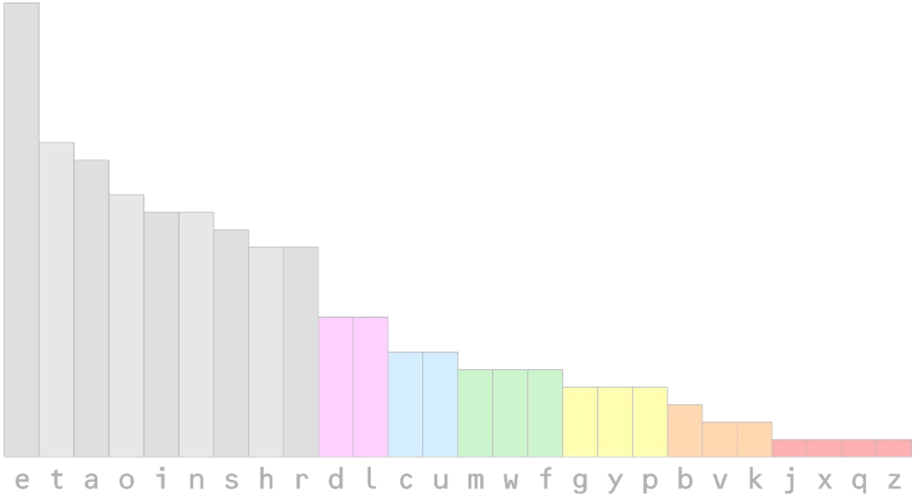
- Clearly, **simple substitution ciphers** are vulnerable to frequency analysis
  - **Root cause: ???**



Ordered by frequency

Recap: Confidentiality via Substitution Ciphers

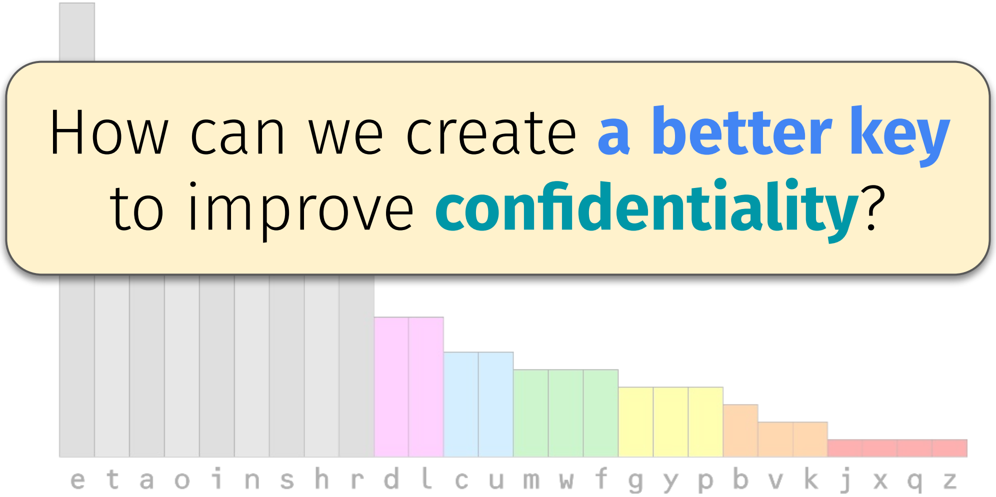
- Clearly, **simple substitution ciphers** are vulnerable to frequency analysis
  - **Root cause:** the key length is **much smaller** than the plaintext length



Ordered by frequency

Recap: Confidentiality via Substitution Ciphers

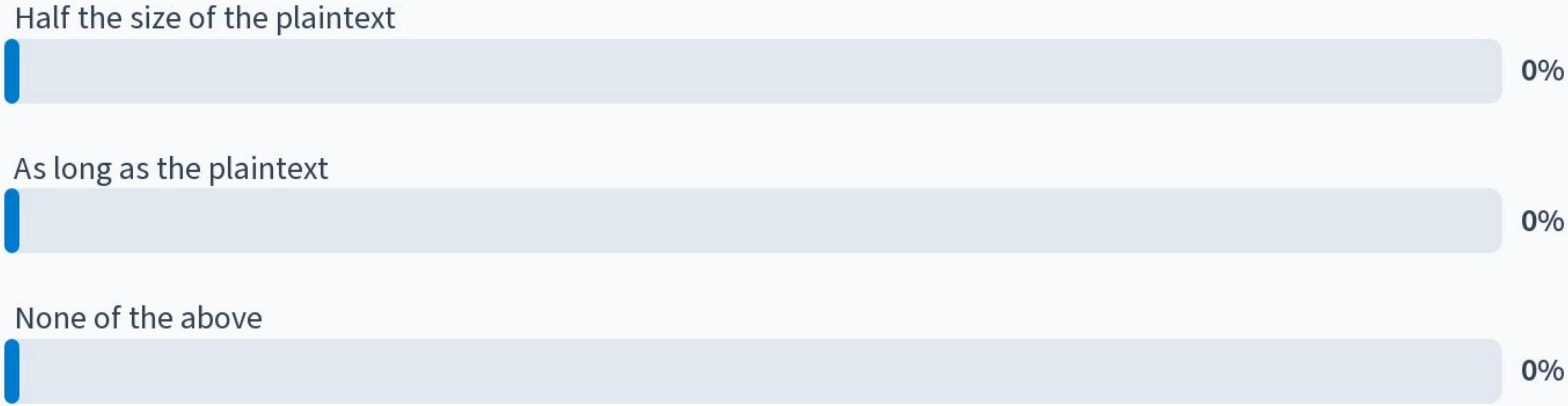
- Clearly, **simple substitution ciphers** are vulnerable to frequency analysis
  - **Root cause:** the key length is **much smaller** than the plaintext length



Ordered by frequency

How can we create **a better key** to improve **confidentiality**?

How long should an ideal cipher key be?

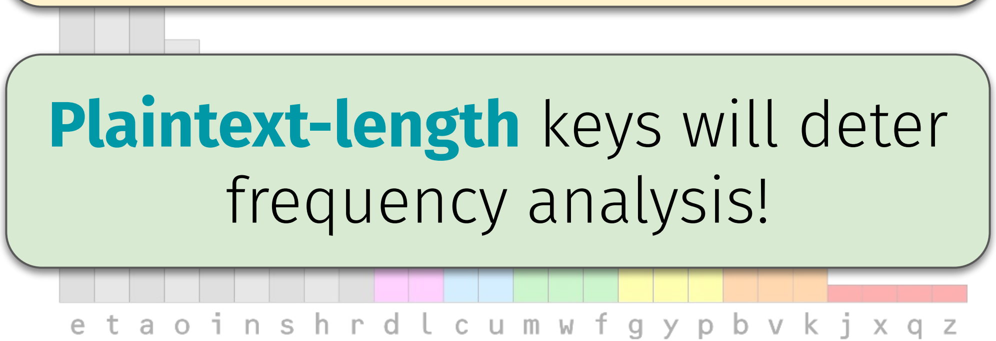


Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Recap: Confidentiality via Substitution Ciphers

- Clearly, **simple substitution ciphers** are vulnerable to frequency analysis
  - **Root cause:** the key length is **much smaller** than the plaintext length

How can we create **a better key** to improve **confidentiality**?



Plaintext-length keys will deter frequency analysis!

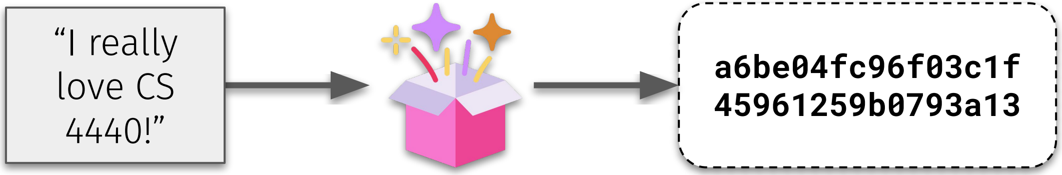
Ordered by frequency

Generating Keys

- **Functions: ???**

Generating Keys

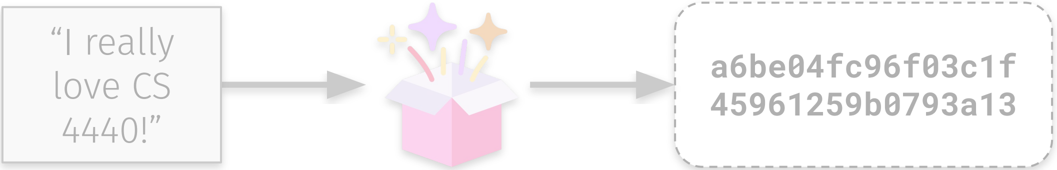
- **Functions:** takes input and generates output
  - E.g., Hash functions
  - E.g., HMAC functions



- **Generators: ???**

Generating Keys

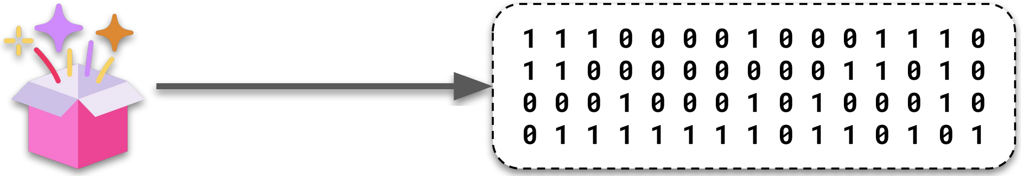
- **Functions:** takes input and generates output
  - E.g., Hash functions
  - E.g., HMAC functions



“I really love CS 4440!”

```
a6be04fc96f03c1f
45961259b0793a13
```

- **Generators:** produces output out of thin air
  - E.g., number generators
  - E.g., HMAC secret keys



```
1 1 1 0 0 0 0 1 0 0 0 1 1 1 0
1 1 0 0 0 0 0 0 0 0 1 1 0 1 0
0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 1 1 1 1 1 1 1 0 1 1 0 1 0 1
```

An ideal key is *random*...

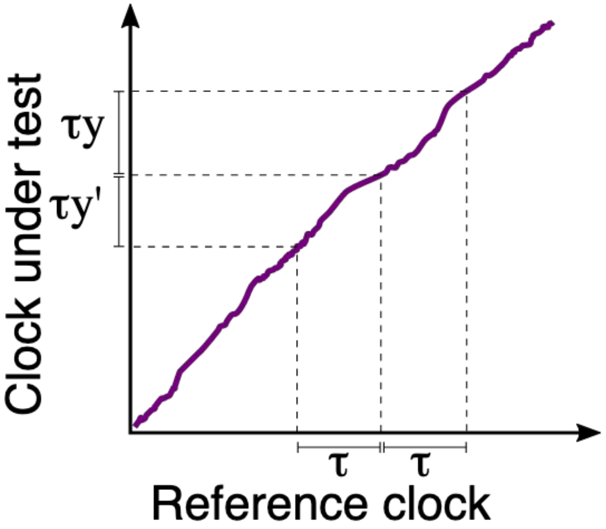
What are some physical sources of randomness?

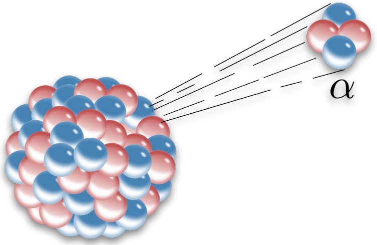
Nobody has responded yet.

Hang tight! Responses are coming in.

Generating Random Keys

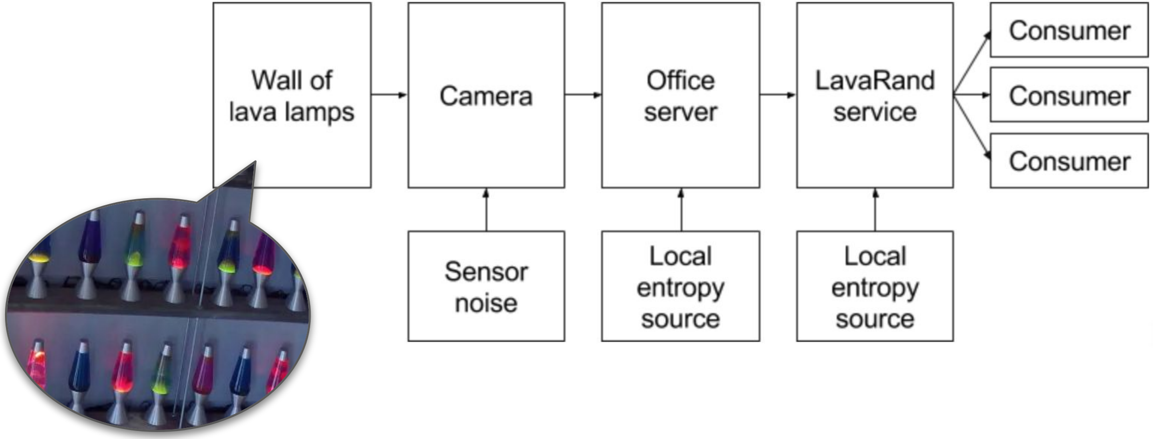
- **Physical randomness:**
  - Coin flips
  - Atomic decay
  - Thermal noise
  - Electromagnetic noise
  - Physical variation
    - Clock drift
    - DRAM decay
    - Image sensor errors
    - SRAM startup-state
  - Lava Lamps





Generating Random Keys

- Harnessing physical randomness: “LavaRand”
  - True randomness from lava lamps
  - Used by CloudFlare today



Wall of lava lamps

Camera

Office server

LavaRand service

Consumer

Consumer

Consumer

Sensor noise

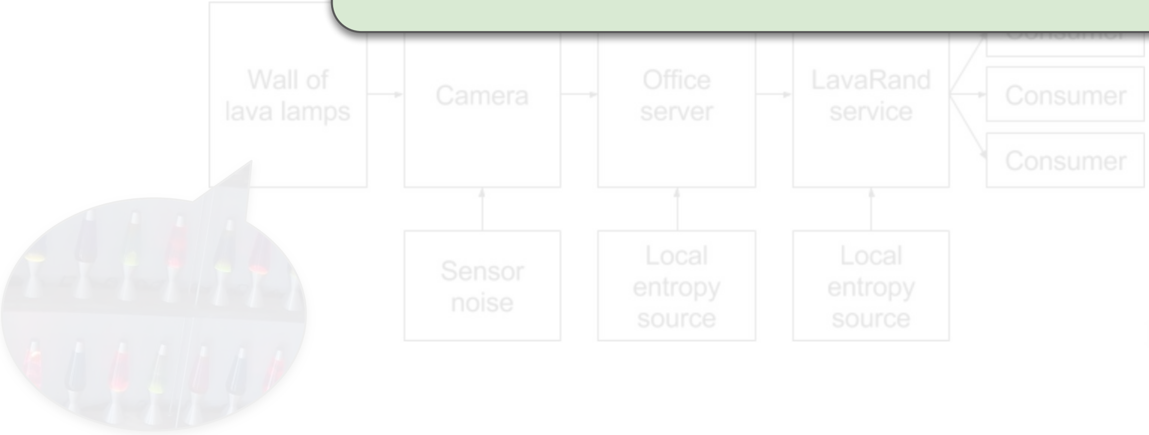
Local entropy source

Local entropy source

Generating Random Keys

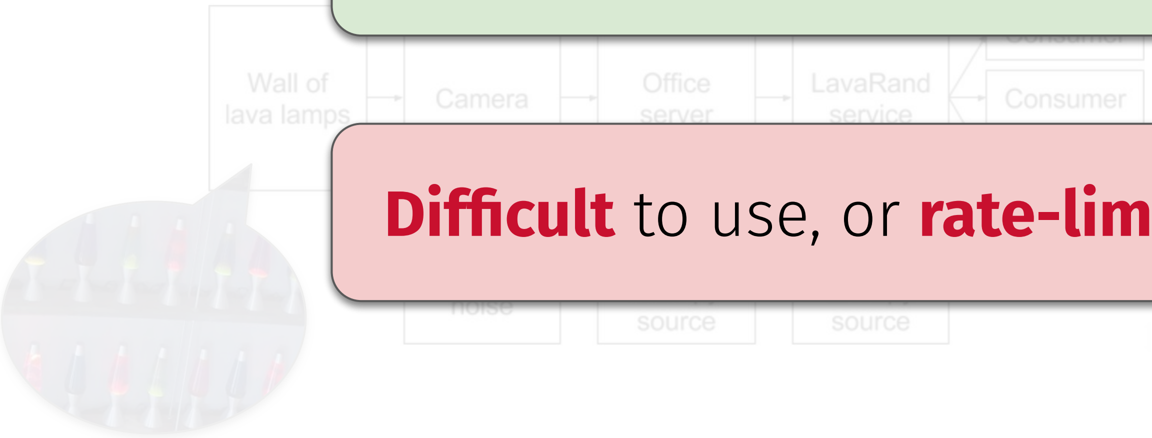
- Harnessing physical randomness: "LavaRand"
  - True randomness from lava lamps
  - Used by Clo

Highest guarantees of **security**



Generating Random Keys

- Harnessing physical randomness: “LavaRand”
  - True randomness from lava lamps
  - Used by Cl



Highest guarantees of **security**

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

“Pseudo” Randomness

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

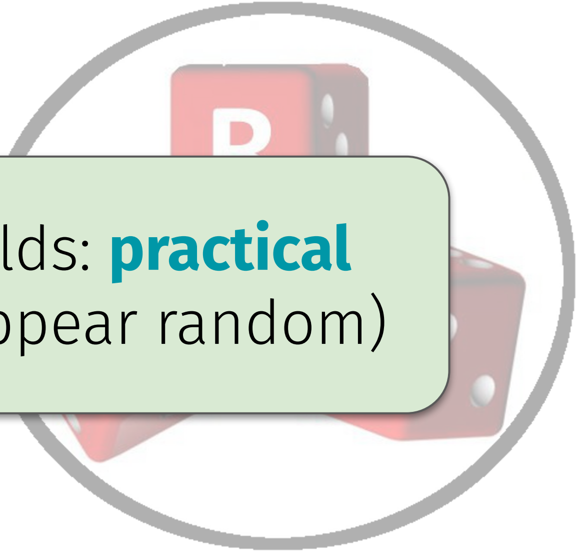
“Pseudo” Randomness

- 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” Randomness

- What is **true randomness**?
  - **Physical** process that’s inherently random
  - **Secure** yet **impractical**
    - Scarce, hard to use
- Pse
  - **Output:** long sequence that **appears random**



PRGs offer the best of both worlds: **practical** (fast, easy-to-use) and **secure** (appear random)

Pseudo-random Generators (PRGs)

- We say a **PRG** is **secure** if Mallory can’t do better than random guessing

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

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!

Pseudo-random Generators (PRGs)

- We say a **PRG** is **secure** if Mallory can't do better than random guessing

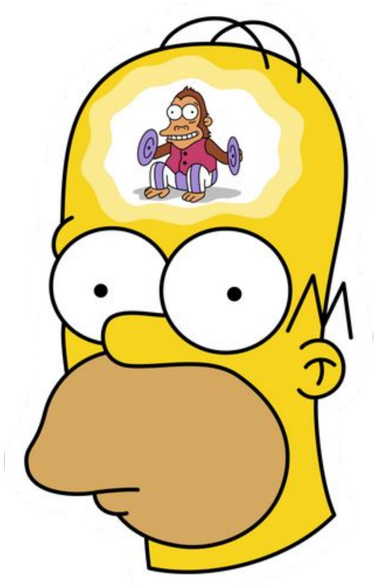
- **Problem:** How much **true randomness** is **enough**?
  - **Example:** **one coin flip** = Mallory needs **very few tries** to guess

- **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)

Constructing a PRG

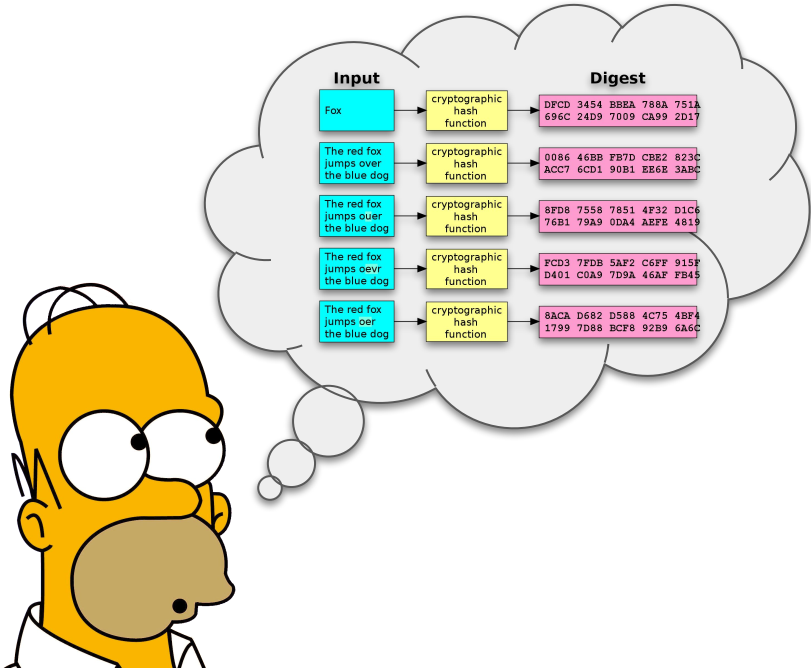
- **Idea:** Build a **PRG** using a **PRF**



Constructing a PRG

- **Idea:** Build a **PRG** using a **PRF**

- **Observation:** **PRF**, given consecutive inputs, produce outputs that are randomly distributed (hopefully)



Input

Digest

Fox → cryptographic hash function → DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17

The red fox jumps over the blue dog → cryptographic hash function → 0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC

The red fox jumps ouer the blue dog → cryptographic hash function → 8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819

The red fox jumps oevr the blue dog → cryptographic hash function → FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45

The red fox jumps oer the blue dog → cryptographic hash function → 8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C

Constructing a PRG

- **Idea:** Build a **PRG** using a **PRF**
- **Observation:** **PRF**, given consecutive inputs, produce outputs that are randomly distributed (hopefully)
- **Result:** For **truly-random s** and **PRF f**:
  - **Pseudo-random generated string** = $f_s(0)$ || $f_s(1)$ || $f_s(2)$ || $f_s(3)$ ...

Proving a PRG is Secure

- **Theorem:** if ***f*** is a **secure PRF**
  - … and ***g*** is seeded from ***f***
  - … then ***g*** must be a **secure PRG**

Proving a PRG is Secure

- **Theorem:** if ***f*** is a **secure PRF**
  - … and ***g*** is seeded from ***f***
  - … then ***g*** must be a **secure PRG**

- **Proof:** if ***f*** is a **secure PRF**, we must show that ***g*** is a **secure PRG**
  1. Assume ***g*** actually is **insecure**… then Mallory can break it
  2. If that were true, Mallory could also break the **PRF** too
  3. This would **contradict** the fact that ***f*** is a **secure PRF**!

Proving a PRG is Secure

- **Theorem:** if *f* is a **secure PRF**
  - ... and ***g*** is seeded from *f*
  - ... then ***g*** r
- **Proof:** if *f* is a PRG
  1. Assume ***g*** actually is **insecure**... then Mallory can break it
  2. If that were true, Mallory could also break the PRF too
  3. This woul

How should we **seed** our PRG?

What happens if we **fail**?

Proving a PRG is Secure

- **Theorem:** if *f* is a **secure PRF**
  - ... and *g* is seeded from *f*
  - ... the

- **Proof:** if *f*
  1. Assu
  2. If tha
  3. This w

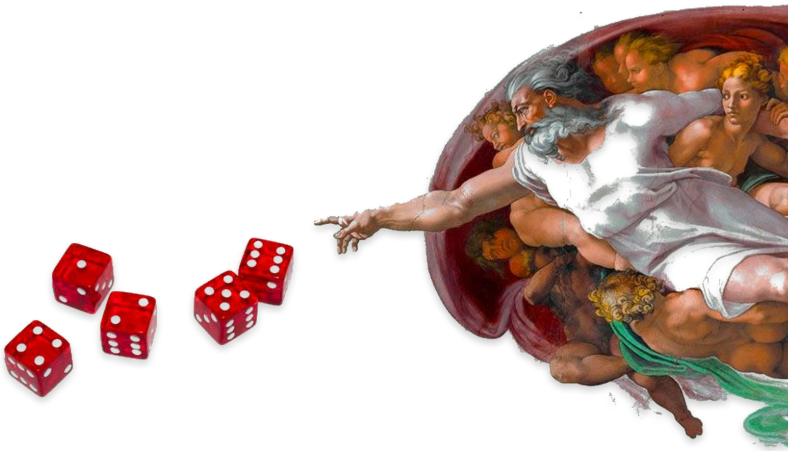
When our assumptions hold, we transform a small amount of **“true” randomness** into a *wealth* of **“apparent” randomness**

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

Questions?

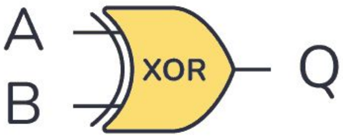
Plaintext-length Keys: One-time Pads

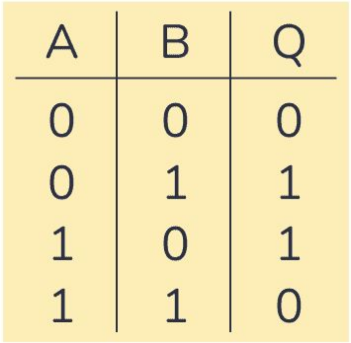
One-time Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad **k**

One-time Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad **k**
  - Encryption: **c_i := p_i XOR k_i**
  - Decryption: **p_i := c_i XOR k_i**





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

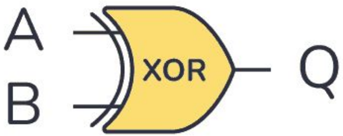
a XOR b XOR b = a

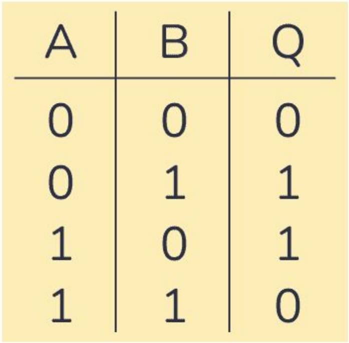
a XOR b XOR a = b

One-time Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad **k**
  - Encryption: c_i := p_i **XOR** k_i
  - Decryption: p_i := c_i **XOR** k_i

- To be secure:
  - Key must be **truly random**
  - Key must never be **reused**





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: Non-random Keys

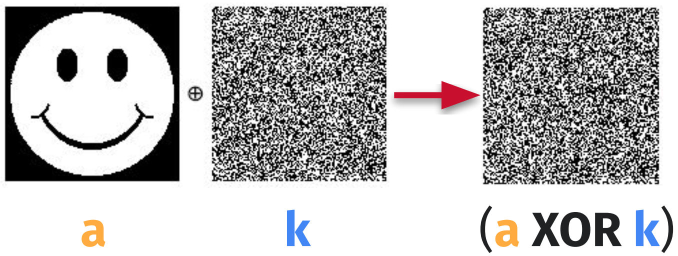
- Suppose the key bits **aren't** truly random
  - E.g., generated by selecting one of three values

- **How would this help Mallory?**

Attacking OTPs: Non-random Keys

- Suppose the key bits **aren't** truly random
  - E.g., generated by selecting one of three values

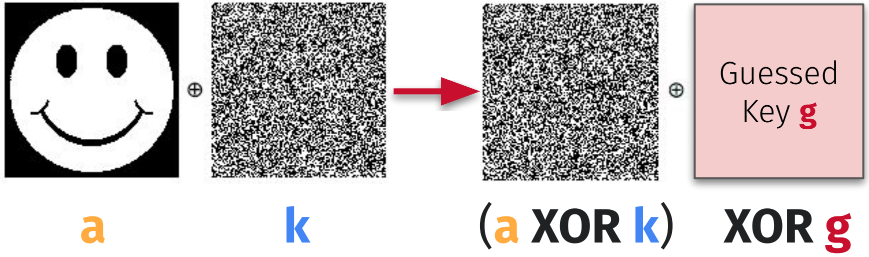
- **How would this help Mallory?**
  1. She intercepts an encrypted message



Attacking OTPs: Non-random Keys

- Suppose the key bits **aren't** truly random
  - E.g., generated by selecting one of three values

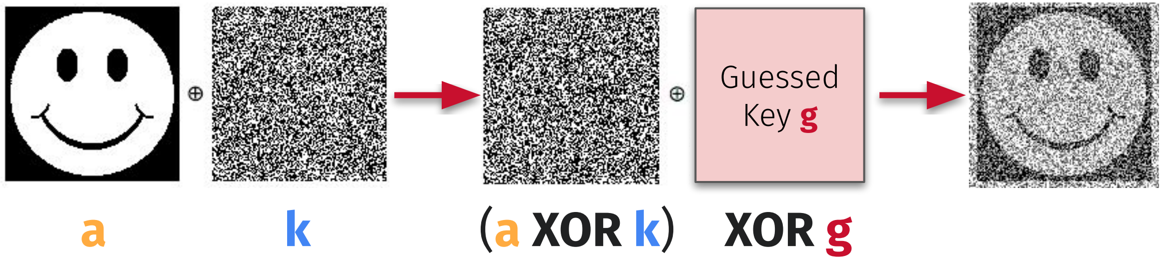
- **How would this help Mallory?**
  1. She intercepts an encrypted message
  2. She **guesses key values** and decrypts



Attacking OTPs: Non-random Keys

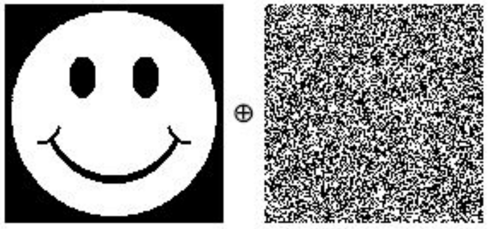
- Suppose the key bits **aren't** truly random
  - E.g., generated by selecting one of three values

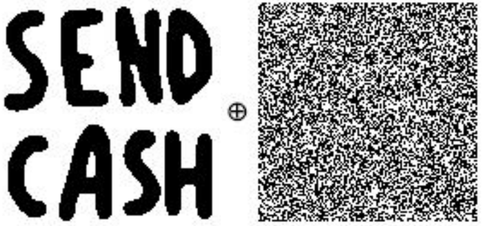
- **How would this help Mallory?**
  1. She intercepts an encrypted message
  2. She **guesses key values** and decrypts
  3. She can **recover** parts of the plaintext!



Attacking OTPs: Key Reuse

(a XOR k)

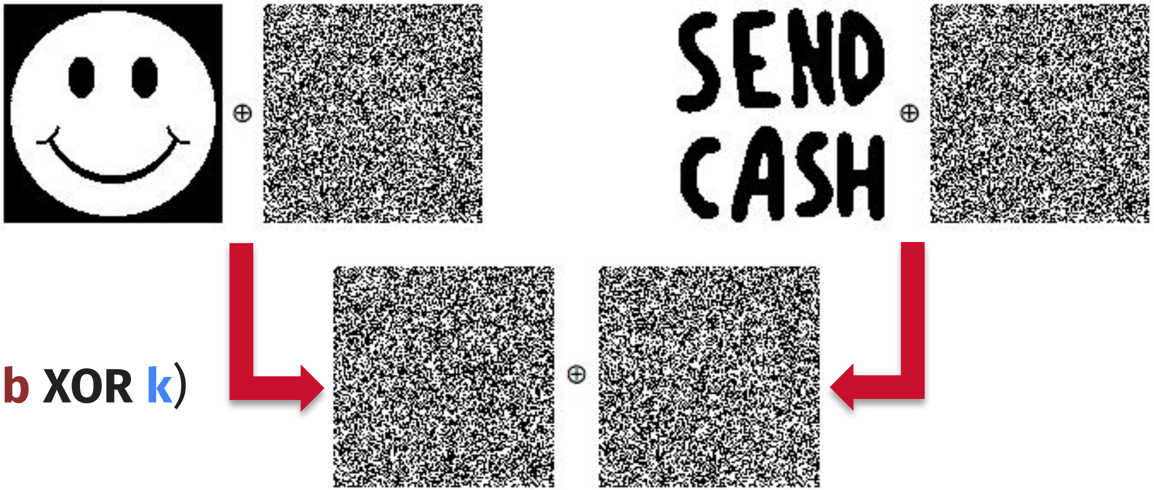




(b XOR k)

Attacking OTPs: Key Reuse

(a XOR k)

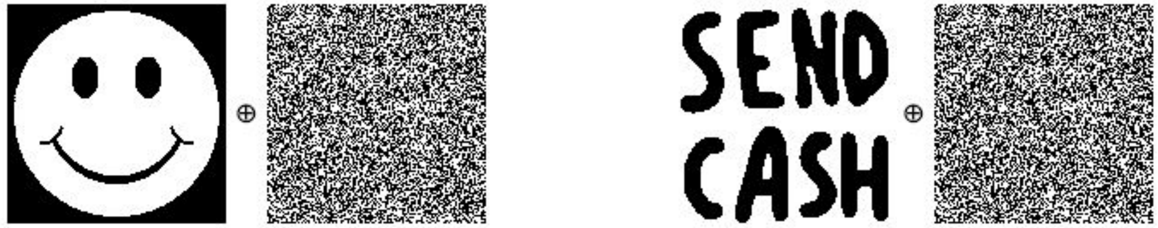


(b XOR k)

(a XOR k) XOR (b XOR k)

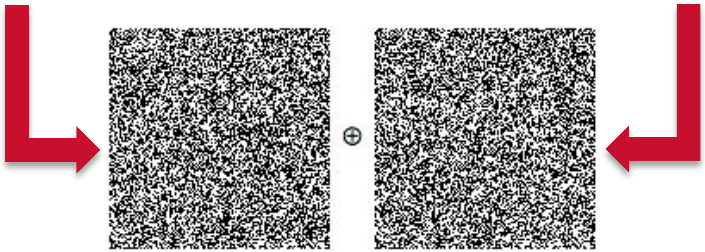
Attacking OTPs: Key Reuse

(a XOR k)

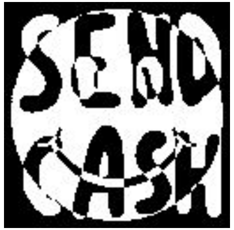


(b XOR k)

(a XOR k) XOR (b XOR k)



= a XOR b



One-time Pads

- Alice and Bob generate a **plaintext-length** string of **random bits**: the one-time pad **k**
  - Encryption:
  - Decryption:
- To be secure:
  - Key must be **truly random**
  - Key must never be **reused**

Provably **Secure**

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

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:
- To be secure:
  - Key must be
  - Key must nev

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

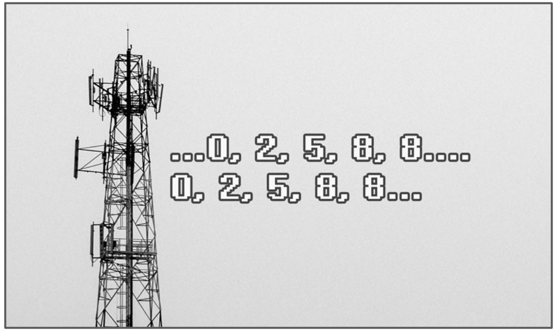
Provably **Secure**

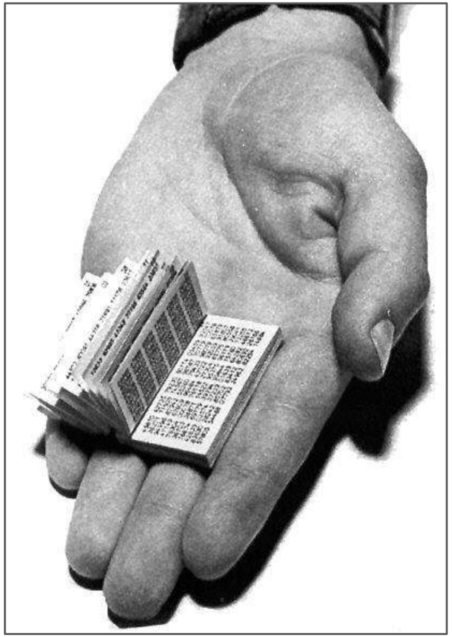
(if key is **random** + not **reused**)

Highly **Impractical**

Impracticality of OTPs

- Generating OTPs
  - Slow and/or rate-limited
    - By hand, LavaRand, etc.
- Deploying OTPs
  - Potentially very long
  - Challenging to conceal
- Cold War numbers stations
  - Encrypted message sent via short-wave radio to agents
  - Agent decrypts with their OTP
    - Throw OTP away after!
  - Many remain in service today!
    - Lincolnshire Poacher





Questions?

Plaintext-length Keys: Stream Ciphers

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

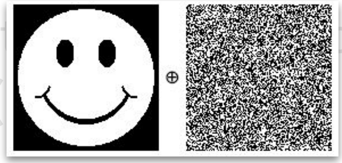
Stream Cipher

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

What if you **reuse** the PRG's **random seed** or its **output**?

tpu tical

ss (know



1. Start with shared secret **truly random** number **k** (e.g., from a lava lamp, mouse clicks, etc.)
2. Alice & Bob each use **k** to seed their 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-random Generator** instead of a truly random pad

tpu tical ess (know

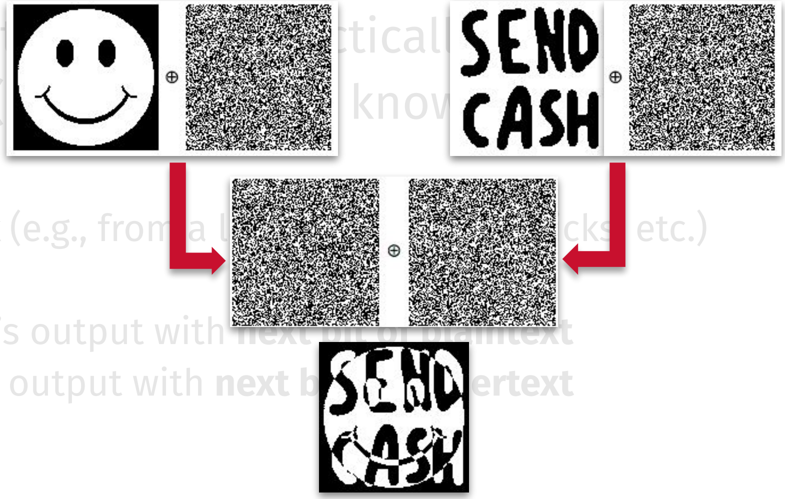
1. Start with shared secret **truly random** number **k** (e.g., from a l cks etc.)

ator's output with **next bit of plaintext**

tor's output with **next b** ertext

What if you **reuse** the PRG's **random seed** or its **output**?

Vulnerable to partial (or full) recovery of the **plaintext**!



Stream Cipher

- **Idea:** Use a pseudorandom generator instead of a truly random pad

- **Recall:** Secure PRG inputs a seed **k**, outputs a stream practically indistinguishal 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**?

Questions?

Transposition Ciphers

Transposition Ciphers

- **Substitution** ciphers swap-out plaintext symbols for others
  - E.g., shifting, XORing, etc.

- We've learned about several substitution ciphers
  - E.g., Caesar, Vigenere, one-time pad, stream cipher

- **Can we come up with an alternative to substitution?**

Transposition Ciphers

- **Substitution** ciphers swap-out plaintext symbols for others
  - E.g., shifting, XORing, etc.
- We've learned about several substitution ciphers
  - E.g., Caesar, Vigenere, one-time pad, stream cipher
- **Can we come up with an alternative to substitution?**
- **Transposition:** rearrange plaintext symbols to create ciphertext

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”

6	3	2	4	1	5

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"

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

- 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 ACDT ESEA
    ROFO DEEC WIREE

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

- 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** = `EVLNX ACDTQ ESEAM`
    `ROFOP DEECD WIREE`
  - Replace `null` with nonsense symbol

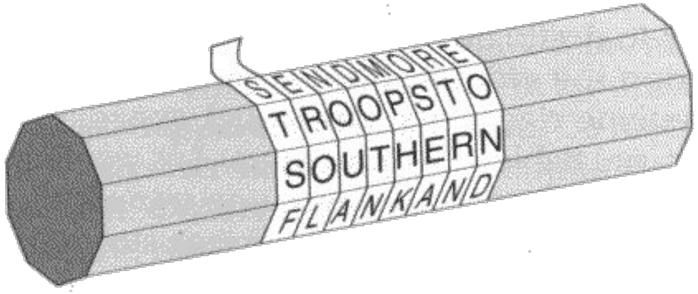
6	3	2	4	1	5
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	F	L	E
E	A	T	O	N	C
E	null	null	null	null	null

Rail Fence (aka Zig Zag or Scytale) Cipher

- Rearrange plaintext on downwards, diagonally successive “rails”

W				E				C				R				L				T				E
	E		R		D		S		O		E		E		F		E		A		O		C	
		A				I				V				D				E				N		

- **c** = `WECRLTE ERDSOEEFEAOC AIVDEN`



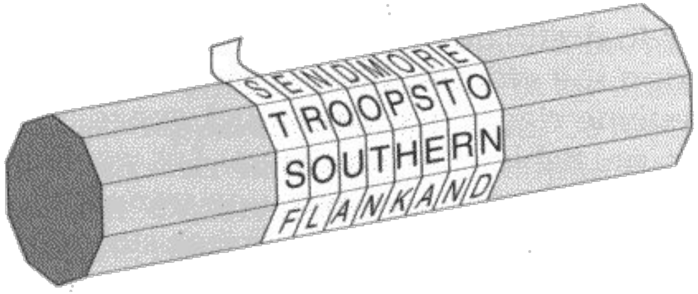
Rail Fence (aka Zig Zag or Scytale) Cipher

- Rearrange plaintext on downwards, diagonally successive “rails”

W				E				C				R				L				T				E
	E		R		D		S		O		E		E		F		E		A		O		C	
		A				I				V				D				E				N		

- **c** = `WECRLTE ERDSOEEFEAOC AIVDEN`

- **Decryption:** use same-diameter cylinder!



Columnar Cipher Cryptanalysis

- What does a **brute force** attack look like?

Columnar Cipher Cryptanalysis

- What does a **brute force** attack look like?
  1. Guess number of columns
  2. Rearrange ciphertext in (probably) wrong order
  3. Look for anagrams to get correct order
     - Harder if null characters are rewritten

- Weakness of a transposition cipher?

Columnar Cipher Cryptanalysis

- What does a **brute force** attack look like?
  1. Guess number of columns
  2. Rearrange ciphertext in (probably) wrong order
  3. Look for anagrams to get correct order
     - Harder if null characters are rewritten

- Weakness of a transposition cipher?
  - **Plaintext** characters end up in the ciphertext

Is it transposition or substitution?

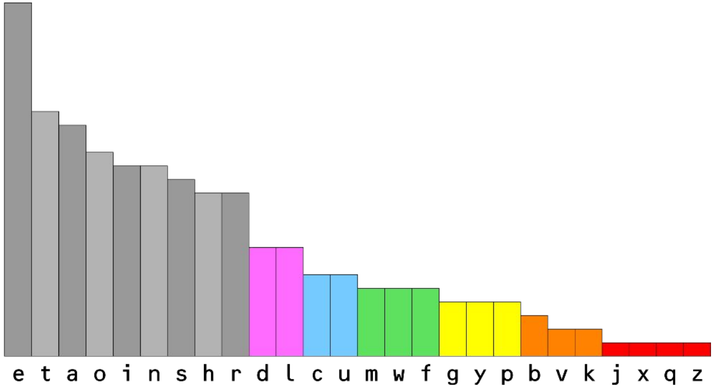
- Given a message ciphertext, how can you determine whether a transposition or a substitution cipher encrypted the plaintext?
  - **Hint:** frequency analysis

Is it transposition or substitution?

- Given a message ciphertext, how can you determine whether a transposition or a substitution cipher encrypted the plaintext?
  - **Hint:** frequency analysis

- **Transposition:**
  - Letters have **expected** letter frequencies

- **Substitution:**
  - Letters have **different** letter frequencies



Ordered by frequency

Stronger Transposition

- How would you build a stronger columnar transposition cipher?
- **Transpose multiple times** with same or different keywords

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

k_1 = “ZEBRAS” (632415)

c_1 = EVLNX ACDTQ ESEAM
ROFOP DEECD WIREE

Stronger Transposition

- How would you build a stronger columnar transposition cipher?
- **Transpose multiple times** with same or different keywords

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

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

Stronger Transposition

- How would you build a stronger columnar transposition cipher?
- **Transpose multiple times** with same or different keywords
  - **Myszkowski Transposition** on recurring letters in key

T	O	M	A	T	O
5	3	2	1	6	4
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

c = ROFOXACDTWESEAZDEECNWIREEEVLNQ

Stronger Transposition

- How would you build a stronger columnar transposition cipher?
- **Transpose multiple times** with same or different keywords
  - **Myszkowski Transposition** on recurring letters in key

T	O	M	A	T	O
5	**3**	**2**	**1**	**6**	**4**
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

c = ROFO**X**ACDT**W**ESEA**Z**DEEC**N**WIREEEVLN**Q**

T	O	M	A	T	O
4	**3**	**2**	**1**	**4**	**3**
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

c = ROFO**X**ACDT**B**EDSEEEAC**TW**WEIVRLENE**Q**

Stronger Transposition

- How would you build a stronger columnar transposition cipher?

- **Fractionation:** convert letters into symbols and transpose those
  - E.g., morse code encoding, bits instead of letters

Stronger Transposition

- How would you build a stronger columnar transposition cipher?

- **Fractionation:** convert letters into symbols and transpose those
  - E.g., morse code encoding, bits instead of letters

- Suppose **p** = “We are discovered…”
  - **Morse:** o—— o o2— o—o o —oo oo ooo —o—o ——— ooo— o o—o o —oo
  - **Binary:** 01010111 01100101 01100001 01110010 01100101 01100100 01101001 01110011 01100011 01101111 01110110 01100101 01110010 01100101 01100100

Stronger Transposition

- How would you build a stronger columnar transposition cipher?

- **Combine with a substitution cipher**
  - Makes anagram discovery more difficult

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**

c_1 = EVLN**B** ACDT**A** ESEA**R**
ROFO**X** DEEC**B** WIREE

Stronger Transposition

- How would you build a stronger columnar transposition cipher?

- **Combine with a substitution cipher**
  - Makes anagram discovery more difficult

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

c_1 = EVLNB ACDTA ESEAR
ROFOX DEECB WIREE

k_s = **ABCAB CABCA BCABC**

c_2 = EWNNC CCEVA FUEBT
RPHOY FEFEB XKRFG

Questions?

Cipher Metrics

Confusion and Diffusion

- “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

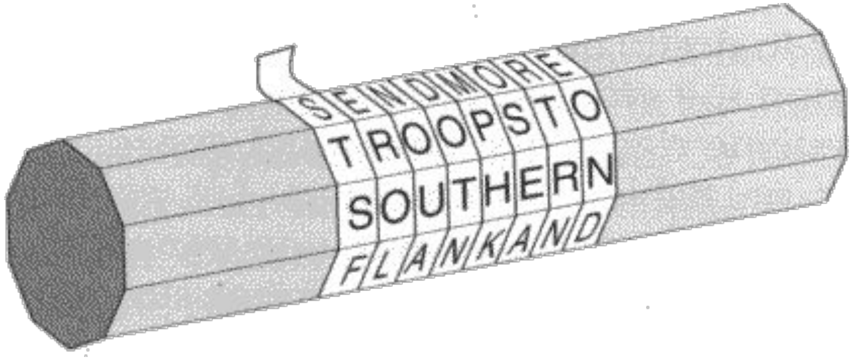
- These are **cipher metrics**—how we “weigh” a cipher’s security

Cipher Metrics: Transposition Ciphers

- Do **transposition ciphers** achieve confusion or diffusion?

Cipher Metrics: Transposition Ciphers

- Do **transposition ciphers** achieve confusion or diffusion?
    - **Diffusion**—they spread the plaintext around!

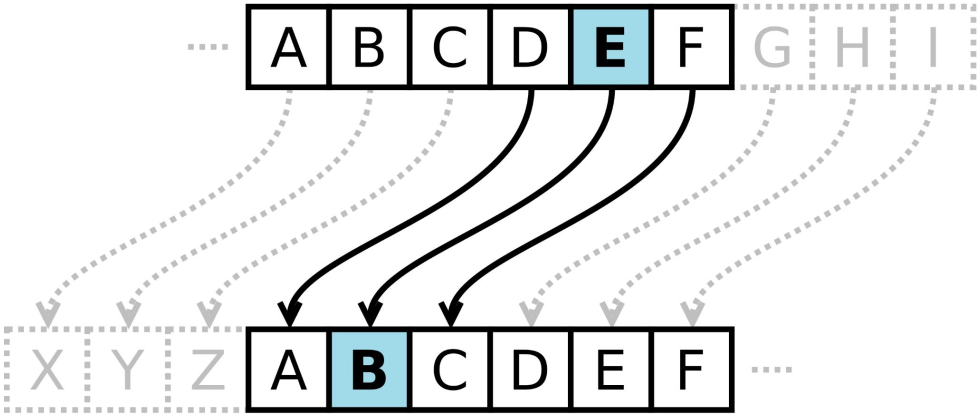


Cipher Metrics: Substitution Ciphers

- What level of confusion & diffusion do ***simple* substitution ciphers** have?

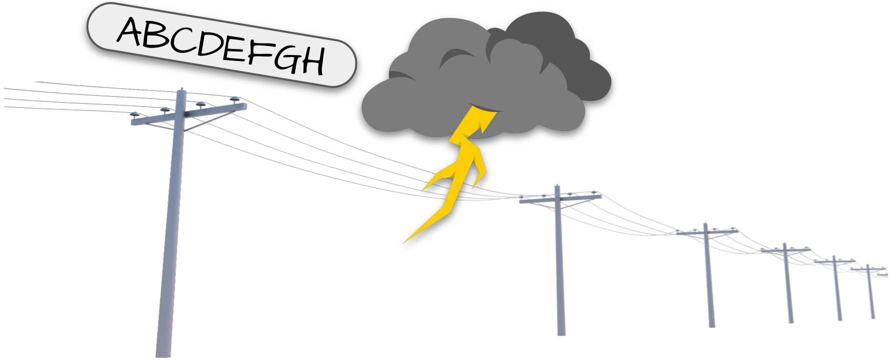
Cipher Metrics: Substitution Ciphers

- What level of confusion & diffusion do ***simple* substitution ciphers** have?
  - **None**—hence why frequency analysis is useful
  - Changing <u>one</u> plaintext or key symbol changes <u>one</u> ciphertext symbol



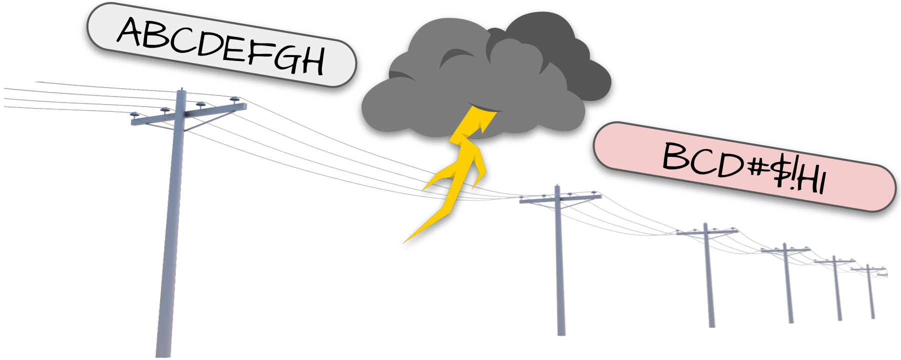
Cipher Metrics: Noisy Channels

- How does **low diffusion** impact communication across a **noisy channel**?



Cipher Metrics: Noisy Channels

- How does **low diffusion** impact communication across a **noisy channel**?
    - Low diffusion = **more tolerant** to corrupted symbols



Questions?

Next time on CS 4440…

Block ciphers, AES, secure channels