Week 2: Lecture B
Message Confidentiality

Thursday, August 29, 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:

- Helpful Resources
- Introduction
- Objectives
- Start by reading this!
  - Working in the VM
  - Testing your Solutions
- Part 1: Hash Collisions
  - Prelude: Collisions
  - Prelude: FastColl
  - Collision Attack
  - What to Submit
- Part 2: Length Extension
  - Prelude: Merkle-Damgår
  - Length Extension Attack
  - What to Submit
- Part 3: Cryptanalysis
  - Prelude: Ciphers
  - Cryptanalysis Attack
  - Extra Credit
  - What to Submit
- Part 4: Signature Forgery
  - Prelude: RSA Signatures
  - Prelude: Bleichenbacher
  - Forgery Attacks
  - What to Submit

Announcements

utahsec

See Discord for meeting info!

utahsec.cs.utah.edu

Questions?

Last time on CS 4440…

Message Integrity
Kerckhoffs's Principles
Pseudo-random Functions
Hashes and HMACs

Message Integrity

- **Goal:** communicate answers while taking the final exam
- **Countermeasure:** randomized seating + curved grading
- **Threat:** Mallory may **change** the message
- **Counter-countermeasure: ???**

Message Integrity

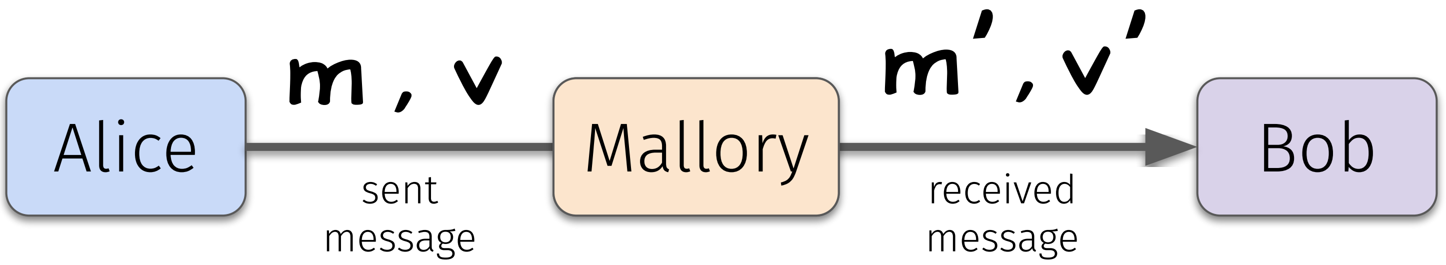
- **Goal:** communicate answers while taking the final exam
- **Approach:** include a **message-dependent message** with the sent message
  - Let **v** = ***f*** **(m)**

Message Integrity

- **Goal:** communicate answers while taking the final exam
- **Approach:** include a **message-dependent message** with the sent message
  - Let **v** = ***f*** **(m)**
- Bob accepts message if ***f*** **(m')** = **v'**

Message Integrity

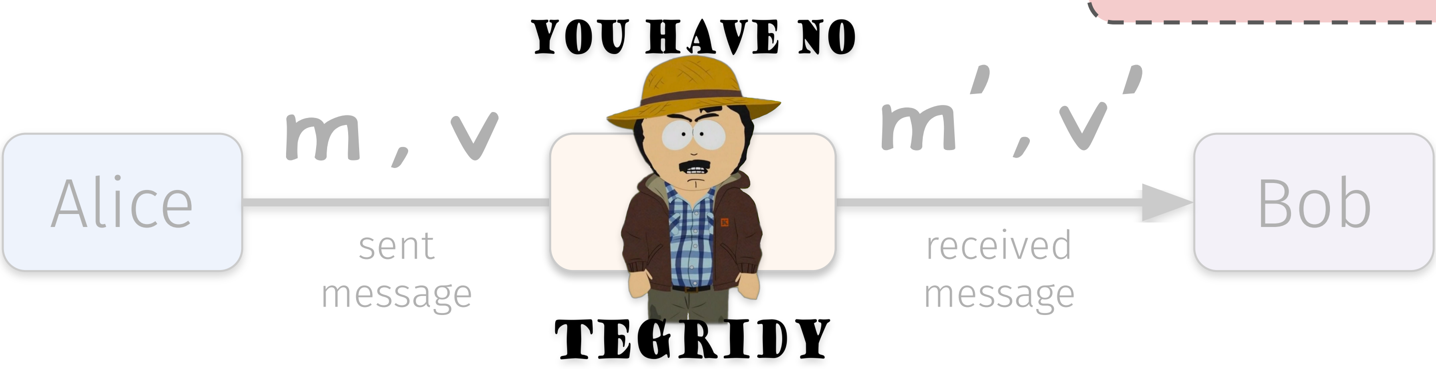
- **Goal:** communicate answers while taking the final exam
- **Approach:** include a **message-dependent message** with the sent message
  - Let **v** = ***f*** **(m)**
- Bob accepts message if ***f*** **(m')** = **v'**
- If check **fails**, **???**



Message Integrity

- **Goal:** communicate answers while taking the final exam
- **Approach:** include a **message-dependent message** with the sent message
  - Let **v** = ***f*** **(m)**
- Bob accepts message if ***f*** **(m’)** = **v’**
- If check **fails**, **m’** is **untrusted**

$f\,(m') \mathrel{!=} v'$



What should a strong f(m) look like?

- Idea 1: **Random Function:**
  - **???**

What should a strong f(m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical—why?**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure—why?**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced

- Idea 2: **Pseudo-random Function Family (PRF):**
  - **???**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced

- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows **???**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced
- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows set, but not **which function** is chosen
  - **Practical**—**why?**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced

- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows set, but not **which function** is chosen
  - **Practical**—easy and fast to use/share
  - **Secure**—**why?**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced
- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows set, but not **which function** is chosen
  - **Practical**—easy and fast to use/share
  - **Secure**—brute-forcing insanely costly (but possible)

What should a strong f(m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced

- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows set, but not **which function** is chosen
  - **Practical**—easy and fast to use/share
  - **Secure**—brute-forcing insanely costly (but possible)

Think of these as **abstract categories**

What should a strong *f* (m) look like?

- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced
- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows set, but not **which function** is chosen
  - **Practical**—easy and fast to use/share
  - **Secure**—brute-forcing insanely costly (but possible)

Think of these as **abstract categories**

How we **"grade"** actual candidate implementations (e.g., **SHA-256** vs. **HMAC-SHA-256**)

Is a pseudo-random function as secure as a random function?

More secure 0%

Equally secure 0%

Less secure (but still extremely secure) 0%

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

What should a strong *f* (m) look like?

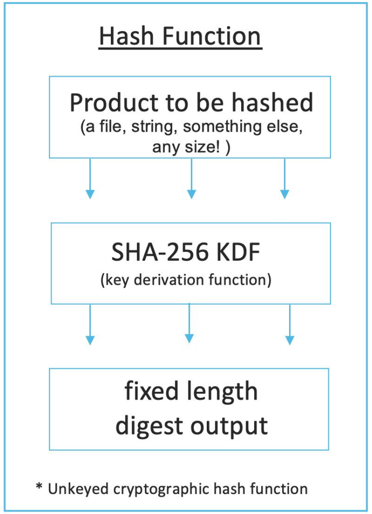
- Idea 1: **Random Function:**
  - Picking from a **seemingly infinite set** of functions
  - **Impractical**—difficult and slow to use/share
  - **Secure**—cannot be brute-forced
- Idea 2: **Pseudo-random Function Family (PRF):**
  - **Subset so large** it seems to be a random function
  - Mallory knows set, but not **which function** is chosen
  - **Practical**—easy and fast to use/share
  - **Secure**—brute-forcing insanely costly (but possible)
  - **Less secure** than random functions—**but very secure**
    - Still too much entropy to feasibly brute-force

Think of these as **abstract categories**

How we **"grade"** actual candidate implementations (e.g., **SHA-256** vs. **HMAC-SHA-256**)

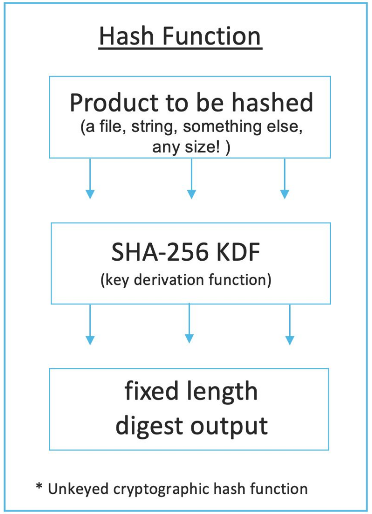
Implementing *f* (m)

- Option 1: **Cryptographic Hash**
  - E.g., SHA256
  - **Not strong PRFs—why?**



Implementing f(m)

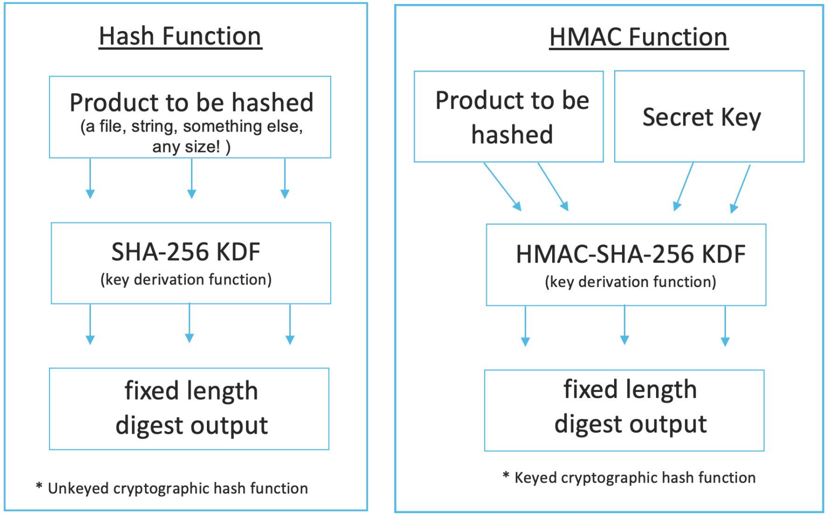
- Option 1: **Cryptographic Hash**
  - E.g., SHA256
  - **Not strong PRFs**
    - Chained construction
    - Length extension attacks



Implementing *f* (m)

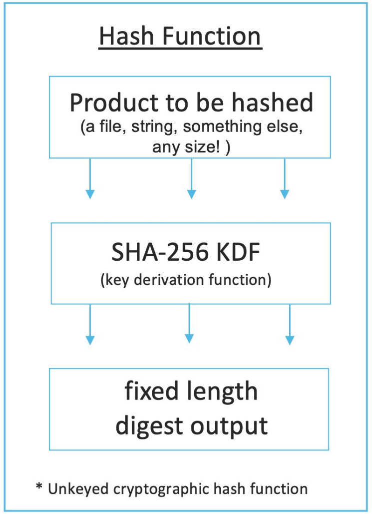
- Option 1: **Cryptographic Hash**
  - E.g., SHA256
  - **Not strong PRFs**
    - Chained construction
    - Length extension attacks

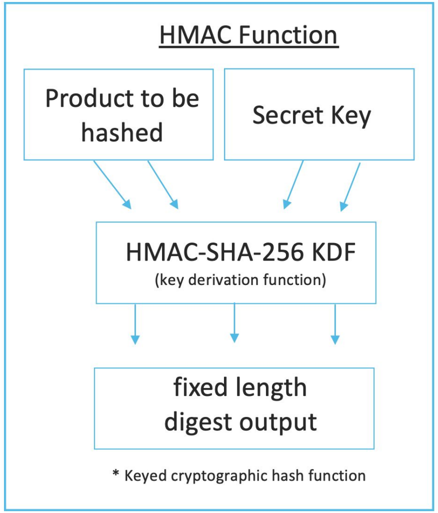
- Option 2: **Message Auth. Code (MAC)**
  - E.g., HMAC-SHA256
  - **Believed to be PRFs—why?**



Implementing *f* (m)

- Option 1: **Cryptographic Hash**
  - E.g., SHA256
  - **Not strong PRFs**
    - Chained construction
    - Length extension attacks
- Option 2: **Message Auth. Code (MAC)**
  - E.g., HMAC-SHA256
  - **Believed to be PRFs**
    - Nested construction
    - Thwarts length extension





Pitfalls of Hashes

- Is *every* hash functions ever created suitable for **cryptographic use today**?

Pitfalls of Hashes

- Is *every* hash functions ever created suitable for **cryptographic use today**?
  - **No way! MD5, SHA-1,** and many others have long been defeated

Lifetimes of popular cryptographic hashes (the rainbow chart)																												
Function	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Snefru																												
MD2 (128-bit)[1]																												
MD4																												
MD5															[2]													
RIPEMD															[2]													
HAVAL-128[1]															[2]													
SHA-0																												
SHA-1																												[3]
RIPEMD-160																												
SHA-2 family																		[4]										
SHA-3 (Keccak)																												

Key	Didn't exist/not public	Under peer review	Considered strong	Minor weakness	Weakened	Broken	Collision found

Pitfalls of Hashes

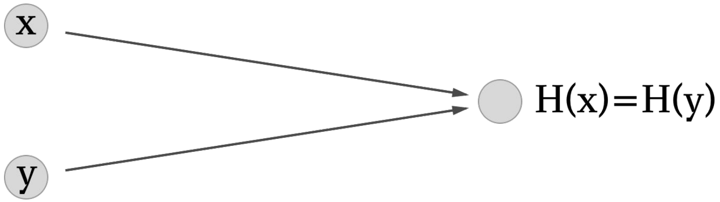
- To be **crypto-safe**, a hash function must be **resilient to what attacks?**

Pitfalls of Hashes

- To be **crypto-safe**, a hash function must be **resilient to what attacks?**
  1. **Collision Attack**
     - **???**

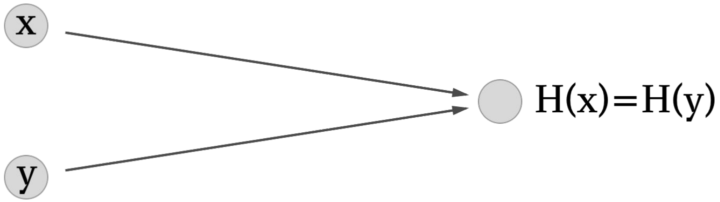
Pitfalls of Hashes

- To be **crypto-safe**, a hash function must be **resilient to what attacks?**
  1. **Collision Attack**
     - Mallory finds $\mathbf{m_1} \mathbf{\,!=\,} \mathbf{m_2}$ such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$



Pitfalls of Hashes

- To be **crypto-safe**, a hash function must be **resilient to what attacks?**
  1. **Collision Attack**
     - Mallory finds $\mathbf{m_1}$ **!=** $\mathbf{m_2}$ such that $h\,(\mathbf{m_1}) = h\,(\mathbf{m_2})$
  2. **Second Pre-image Attack**
     - **???**

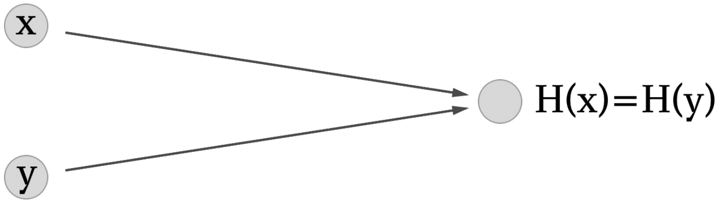


Pitfalls of Hashes

- To be **crypto-safe**, a hash function must be **resilient to what attacks?**

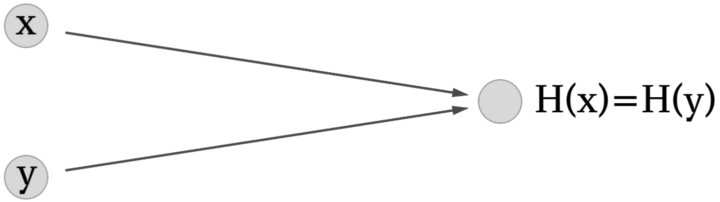
1. **Collision Attack**
   - Mallory finds $\mathbf{m_1}$ **!=** $\mathbf{m_2}$ such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$

2. **Second Pre-image Attack**
   - Given $\mathbf{m_1}$, Mallory finds $\mathbf{m_2}$ **!=** $\mathbf{m_1}$ such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$



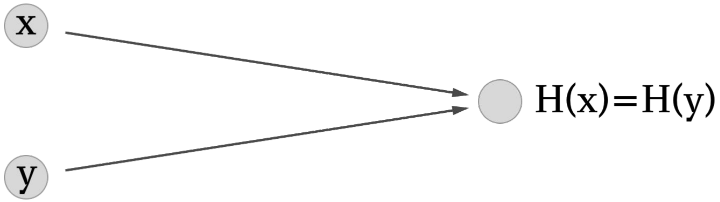
Pitfalls of Hashes

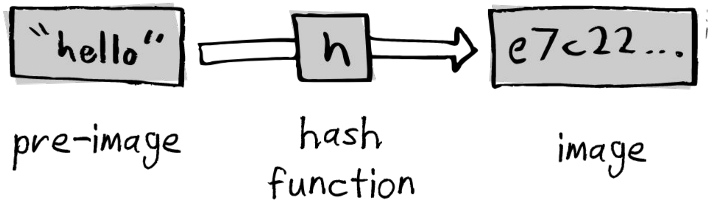
- To be **crypto-safe**, a hash function must be **resilient to what attacks?**
  1. **Collision Attack**
     - Mallory finds $\mathbf{m_1}$ **!=** $\mathbf{m_2}$
       such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$
  2. **Second Pre-image Attack**
     - Given $\mathbf{m_1}$, Mallory finds $\mathbf{m_2}$ **!=** $\mathbf{m_1}$
       such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$
  3. **First Pre-image Attack**
     - **???**



Pitfalls of Hashes

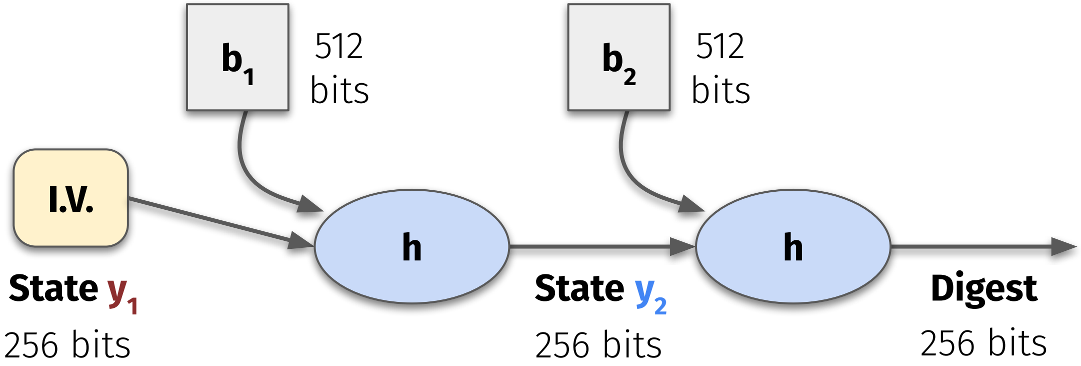
- To be **crypto-safe**, a hash function must be **resilient to what attacks?**
  1. **Collision Attack**
     - Mallory finds $\mathbf{m_1} \mathbf{!=} \mathbf{m_2}$ such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$
  2. **Second Pre-image Attack**
     - Given $\mathbf{m_1}$, Mallory finds $\mathbf{m_2} \mathbf{!=} \mathbf{m_1}$ such that $\boldsymbol{h}\,(\mathbf{m_1}) = \boldsymbol{h}\,(\mathbf{m_2})$
  3. **First Pre-image Attack**
     - Given $\boldsymbol{h}\,(\mathbf{m})$, Mallory finds **m**





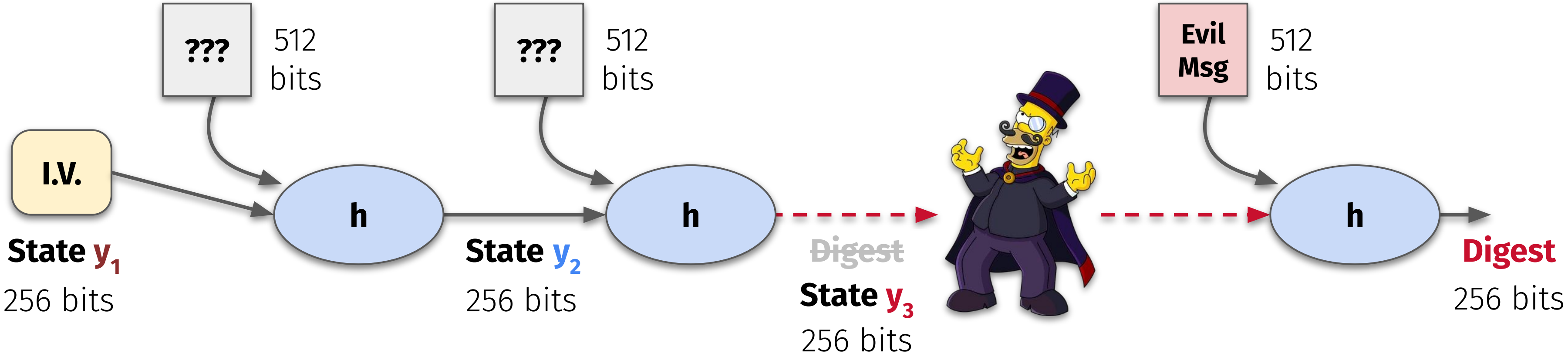
Merkle–Damgård Hashes: Length Extension Attacks

- Merkle–Damgård construction: digest is formed from **the last chaining value**



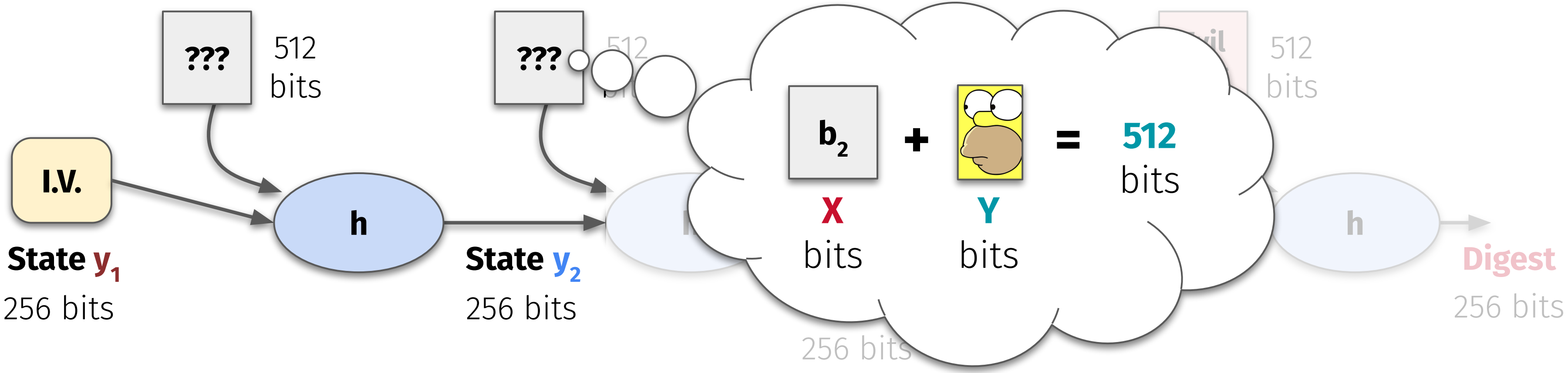
Merkle–Damgård Hashes: Length Extension Attacks

- Merkle–Damgård construction: digest is formed from **the last chaining value**
- Nothing stopping Mallory from **continuing** the hash chain…
  - Mallory **doesn't need** to know the **previous blocks' plaintext**



Merkle–Damgård Hashes: Length Extension Attacks

- Merkle–Damgård construction: digest is formed from **the last chaining value**
- Nothing stopping Mallory from **continuing** the hash chain…
  - Mallory **doesn't need** to know the **previous blocks' plaintext**
  - But she does know that the **last block was padded** to 512 bits



Merkle–Damgård Hashes: Length Extension Attacks

Mallory's resulting hash **digest**

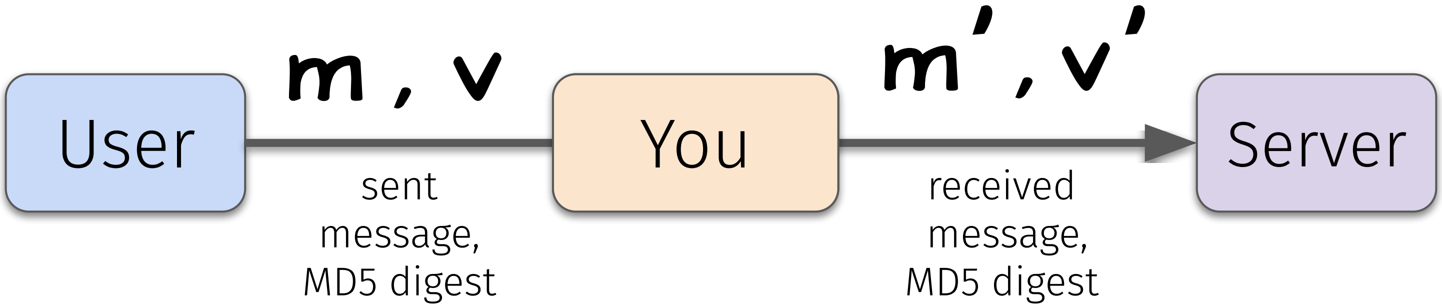
==

$$\textbf{hash}\,(\,\underbrace{\underbrace{\textbf{original}\;||\;\textbf{pad}}_{m}\;||\;\textbf{evil}}_{m'}\,)$$

$$f(m') = v'$$

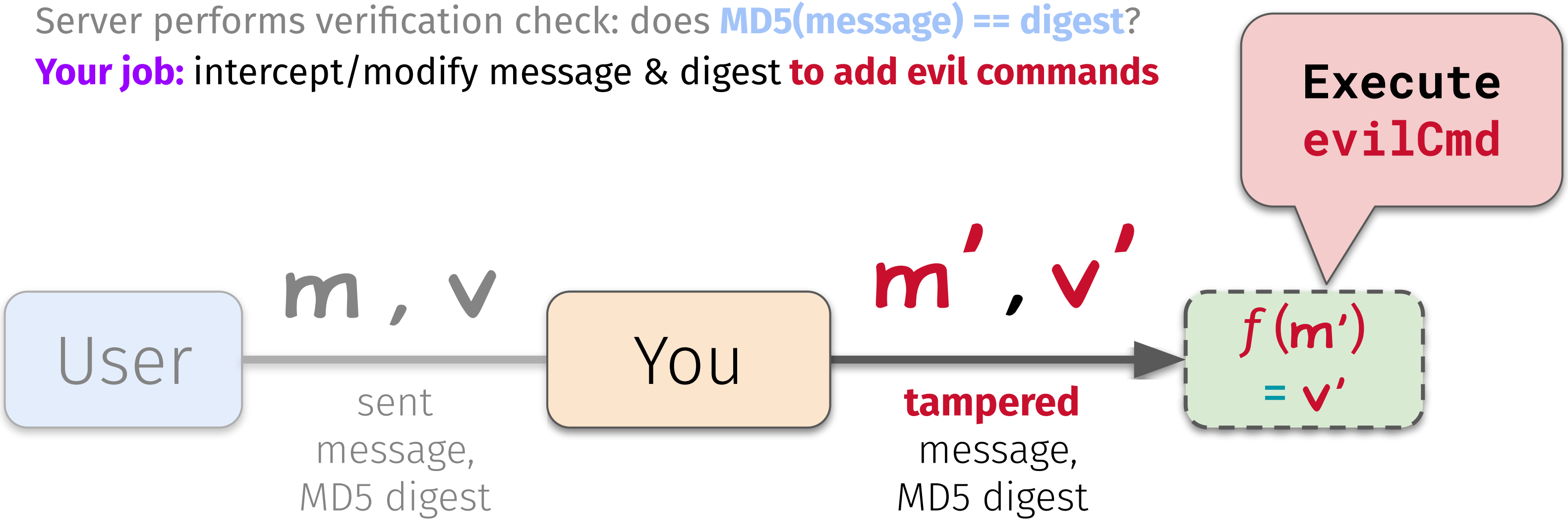
Merkle–Damgård Hashes: Length Extension Attacks

- **Project 1 Part 2:** attack a server that accepts commands
  - User provides message: a secret password + a list of commands
  - User also provides a token that's the **MD5 digest** of the message
  - Server performs verification check: does **MD5(message) == digest**?



Merkle–Damgård Hashes: Length Extension Attacks

- **Project 1 Part 2:** attack a server that accepts commands
  - User provides message: a secret password + a list of commands
  - User also provides a token that's the **MD5 digest** of the message
  - Server performs verification check: does **MD5(message) == digest**?
  - **Your job:** intercept/modify message & digest **to add evil commands**



Exercise: Attacks on Message Integrity

Untampered	v' v	m' m	f(m') v'

Exercise: Attacks on Message Integrity

Untampered	$v' = v$	$m' = m$	$f(m') = v'$
Message Truncated	$v' \quad v$	$m' \quad m$	$f(m') \quad v'$
Hash Collision	$v' \quad v$	$m' \quad m$	$f(m') \quad v'$
Length Extension	$v' \quad v$	$m' \quad m$	$f(m') \quad v'$

Exercise: Attacks on Message Integrity

Untampered	$v' = v$	$m' = m$	$f(m') = v'$
Message Truncated	$v' = v$	$m' \mathrel{!=} m$	$f(m') \mathrel{!=} v$
Hash Collision	$v' \quad v$	$m' \quad m$	$f(m') \quad v'$
Length Extension	$v' \quad v$	$m' \quad m$	$f(m') \quad v'$

Exercise: Attacks on Message Integrity

Untampered	$v' = v$	$m' = m$	$f(m') = v'$
Message Truncated	$v' = v$	$m' \mathrel{!=} m$	$f(m') \mathrel{!=} v'$
Hash Collision	$v' = v$	$m' \mathrel{!=} m$	$f(m') = v'$
Length Extension	$v' \quad v$	$m' \quad m$	$f(m') \quad v'$

Exercise: Attacks on Message Integrity

Untampered	$v' = v$	$m' = m$	$f(m') = v'$
Message Truncated	$v' = v$	$m' \mathrel{!=} m$	$f(m') \mathrel{!=} v'$
Hash Collision	$v' = v$	$m' \mathrel{!=} m$	$f(m') = v'$
Length Extension	$v' \mathrel{!=} v$	$m' \mathrel{!=} m$	$f(m') = v'$

Questions?

This time on CS 4440…

Message Confidentiality
Simple Substitution Ciphers
Cipher Cryptanalysis

Message Confidentiality

Message Confidentiality

- Two parties want to communicate across an untrusted intermediary
- **Confidentiality: ???**

Message Confidentiality

- Two parties want to communicate across an untrusted intermediary
- **Confidentiality:** ensure that only **trusted parties** can read the message

Message Confidentiality

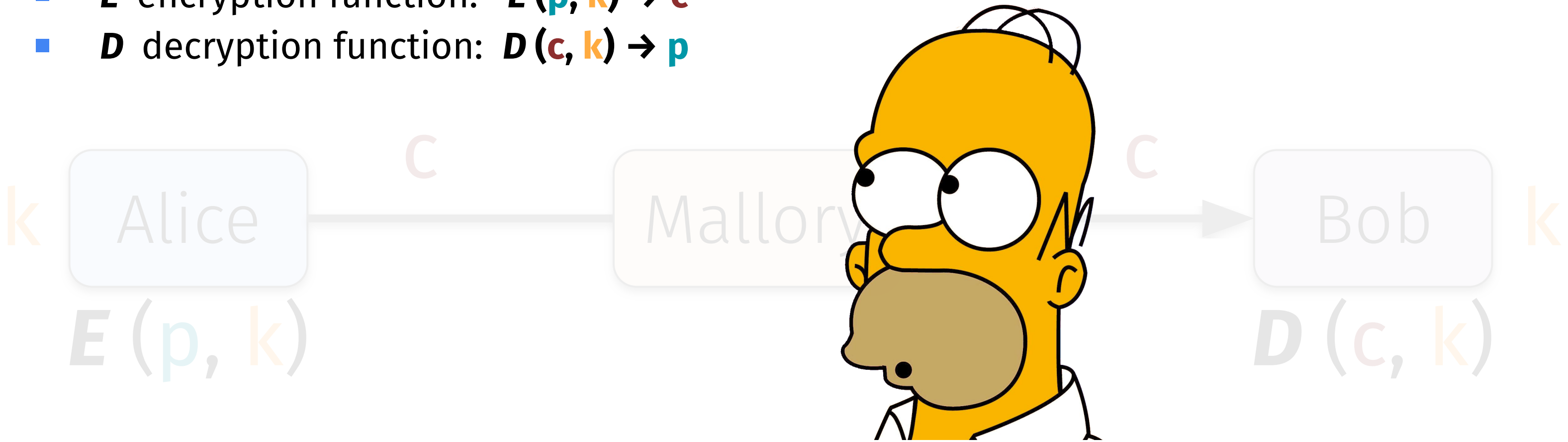
- **Confidentiality:** ensure that only **trusted parties** can read the message
- Terminology
  - **p** plaintext: original, readable message
  - **c** ciphertext: transmitted, unreadable message
  - **k** secret key: known only to Alice and Bob; facilitates **p → c** and **c → p**
  - ***E*** encryption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

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 $p \rightarrow c$ and $c \rightarrow p$
  - *E* encryption function: $E(p, k) \rightarrow c$
  - *D* decryption function: $D(c, k) \rightarrow p$



Substitution Ciphers

Substitution Ciphers

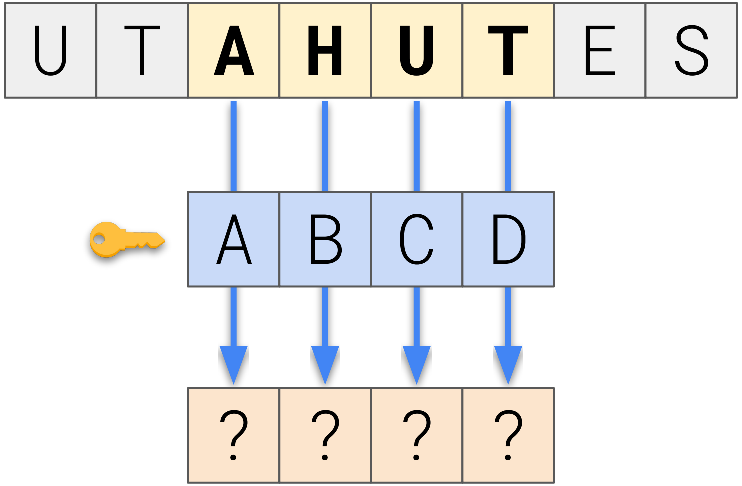
- We define a substitution cipher **key** as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet

A	B	C	D
?	?	?	?

Substitution Ciphers

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

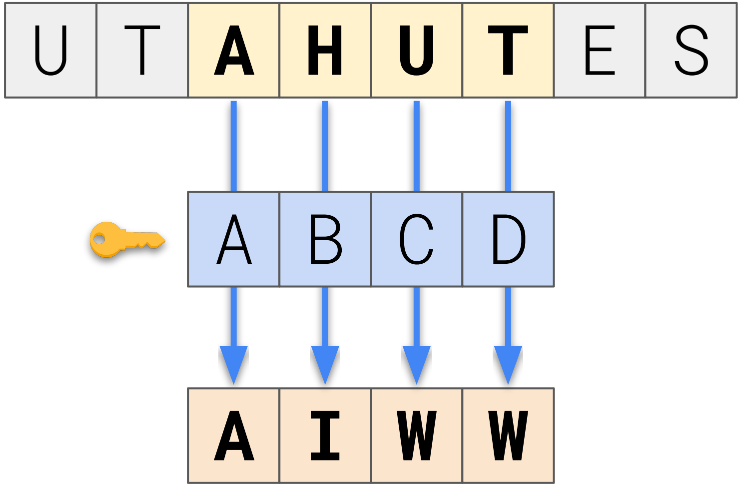
A	B	C	D
0	**1**	**2**	**3**



Substitution Ciphers

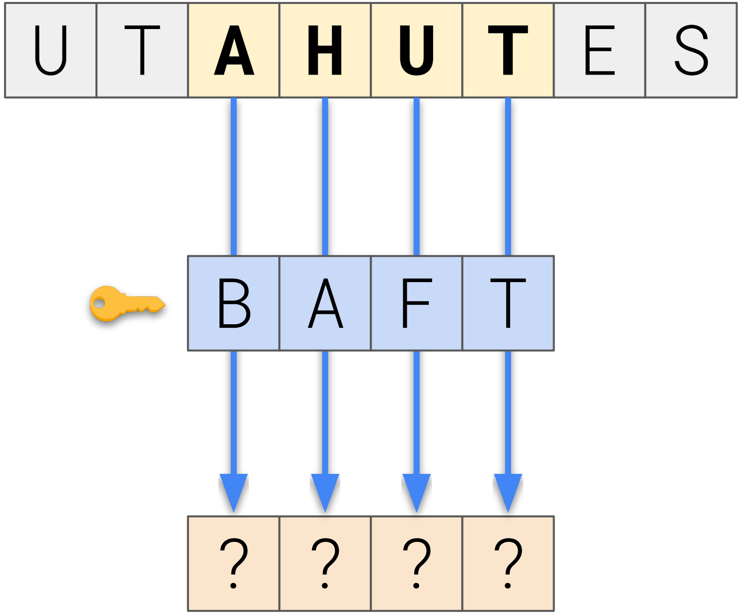
- We define a substitution cipher **key** as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet

A	B	C	D
0	1	2	3



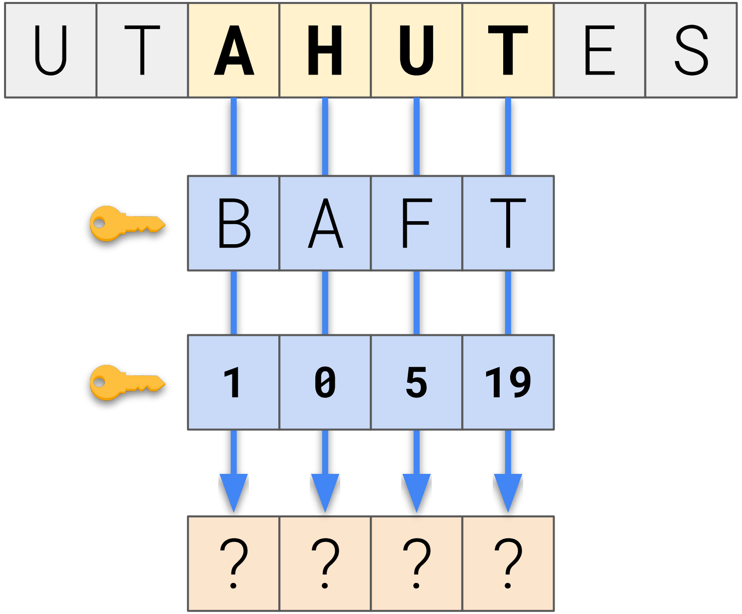
Substitution Ciphers

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



Substitution Ciphers

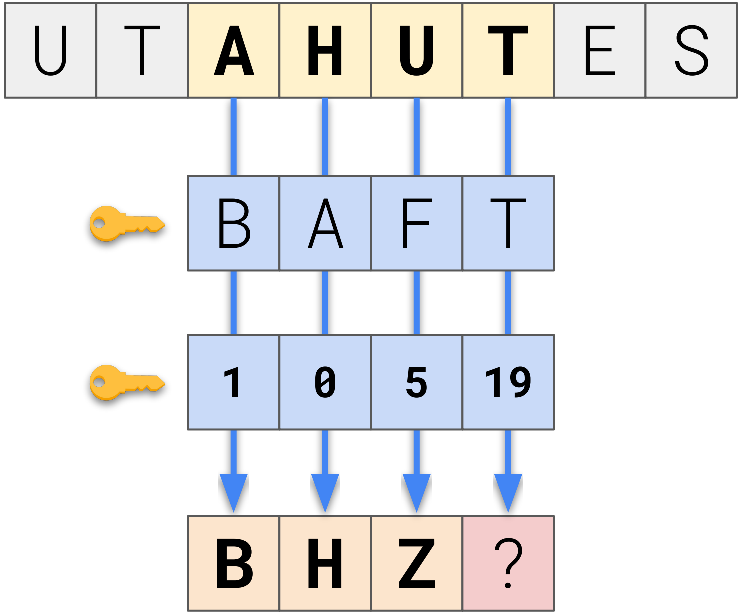
- We define a substitution cipher **key** as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet



Substitution Ciphers

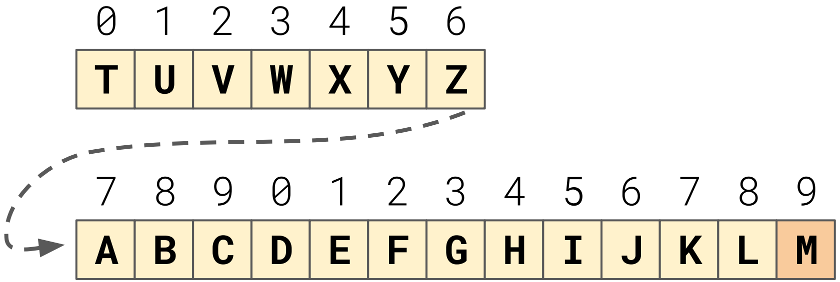
- We define a substitution cipher **key** as a set of **shifts**
- Each shift represented by a **letter**
  - Relative position in the alphabet
- Shift goes past end of alphabet?

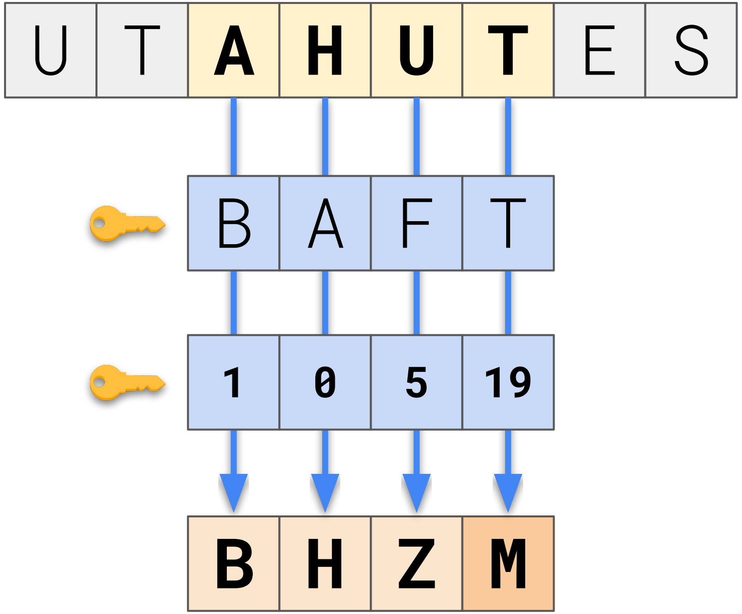
0	1	2	3	4	5	6
T	**U**	**V**	**W**	**X**	**Y**	**Z**



Substitution Ciphers

- We define a substitution cipher **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!





Questions?

Caesar Cipher

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

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: $c_i := (p_i + k) \bmod 26$
  - Decryption: $p_i := (c_i - k) \bmod 26$
- Example for k = 3:
  - Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - +Shift: 33333333333333333333333333
  - =Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

  - Plain: go utes beat wash st
  - +Key: 33 3333 3333 3333 33
  - =Cipher: ?? ???? ???? ???? ??

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: $c_i := (p_i + k) \bmod 26$
  - Decryption: $p_i := (c_i - k) \bmod 26$
- Example for k = 3:
  - Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - +Shift: 33333333333333333333333333
  - =Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

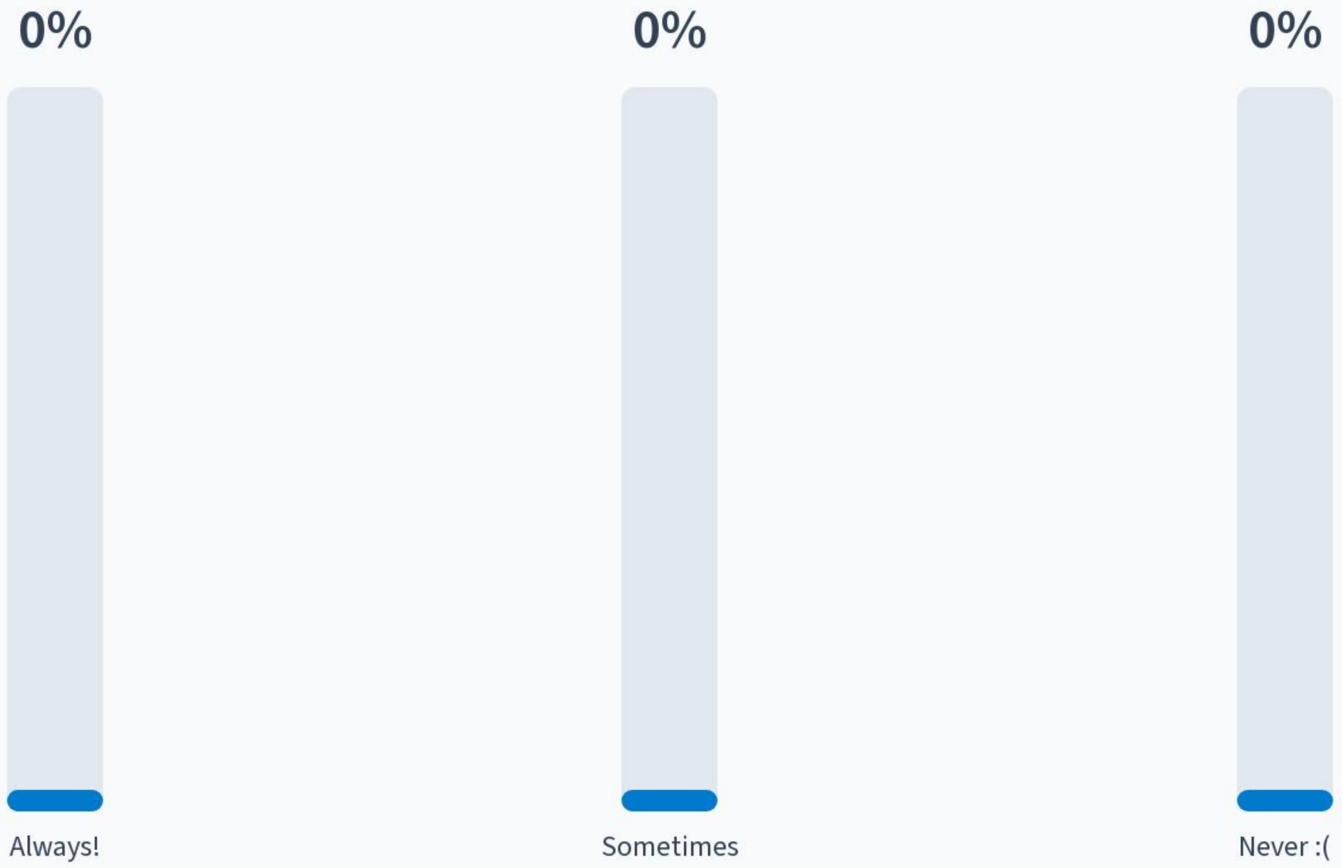
  - Plain: go utes beat wash st
  - +Key: 33 3333 3333 3333 33
  - =Cipher: jr xwhv ehdw zdvk vw

Caesar 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: $c_i := (p_i + k) \bmod 26$
  - Decryption: p_i
- Example for k =
  - Plain: AB
  - +Shift: 33333333333333333333333333
  - =Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC
  - Plain: `go utes beat wash st`
  - +Key: `33 3333 3333 3333 33`
  - =Cipher: `jr xwhv ehdw zdvk vw`

Are Caesar Ciphers **secure**?

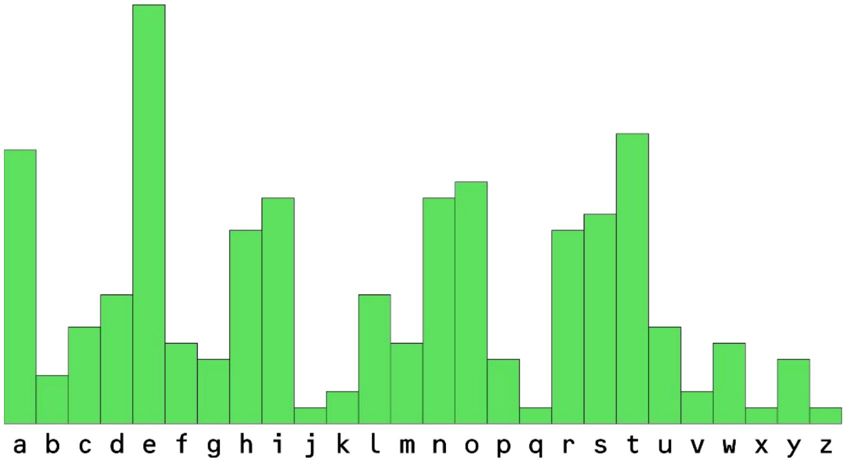
Are Caesar Ciphers secure?



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

Caesar Cipher Cryptanalysis

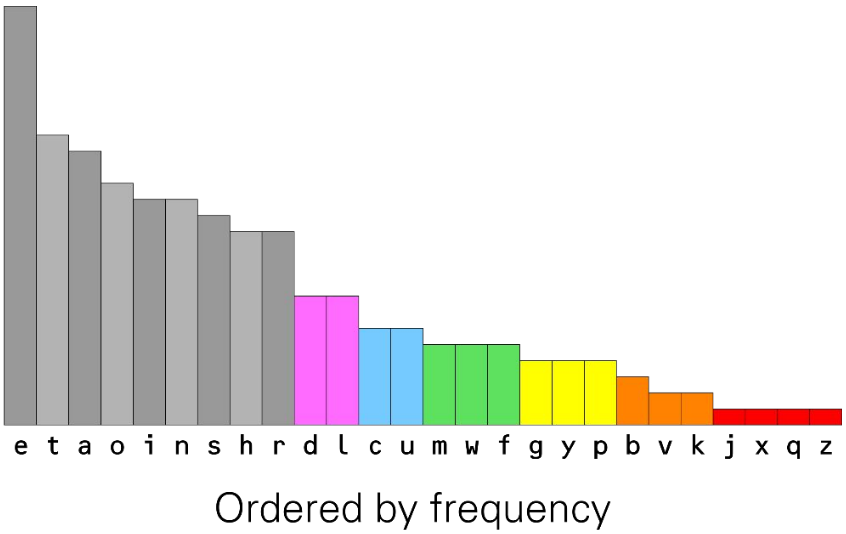
- **Observation:** simple substitution ciphers don't alter **symbol frequency**



Letter frequency for the English language

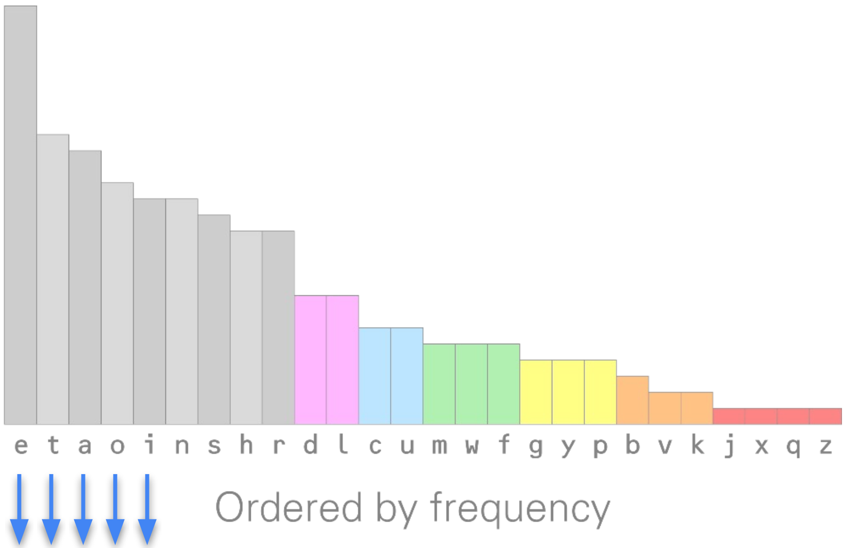
Caesar Cipher Cryptanalysis

- **Problem:** How can we beat brute forcing?
- **Observation:** simple substitution ciphers don't alter **symbol frequency**



Caesar Cipher Cryptanalysis

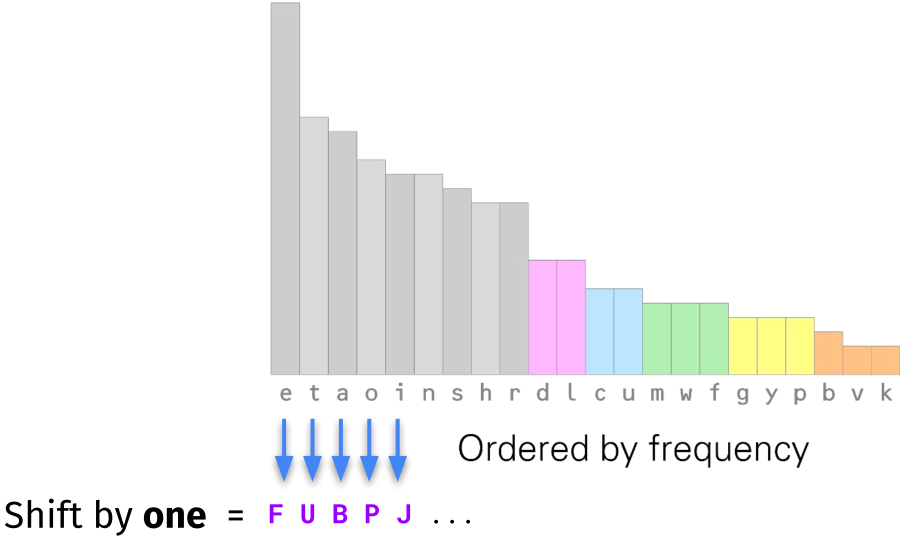
- **Problem:** How can we beat brute forcing?
- **Observation:** simple substitution ciphers don't alter **symbol frequency**



Shift by **one** = F U B P J . . .

Trial-and-Error Caesar Cryptanalysis

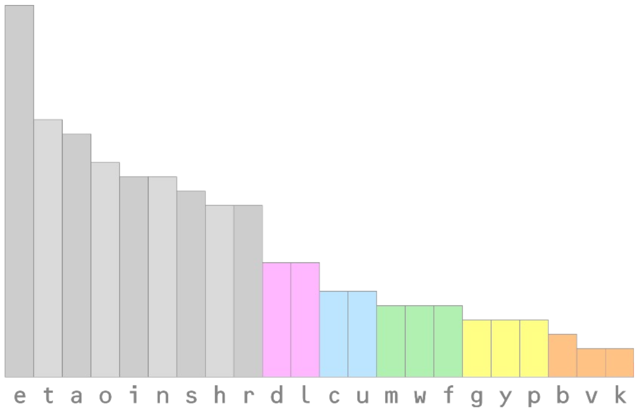
- In Caesar ciphers, the key is only **a single shift** applied repeatedly



Trial-and-Error Caesar Cryptanalysis

- In Caesar ciphers, the key is only **a single shift** applied repeatedly

- Thus, there must be **one out of 26** **reverse shifts** that, when applied:



Ordered by frequency

Shift by **one** = F U B P J . . .

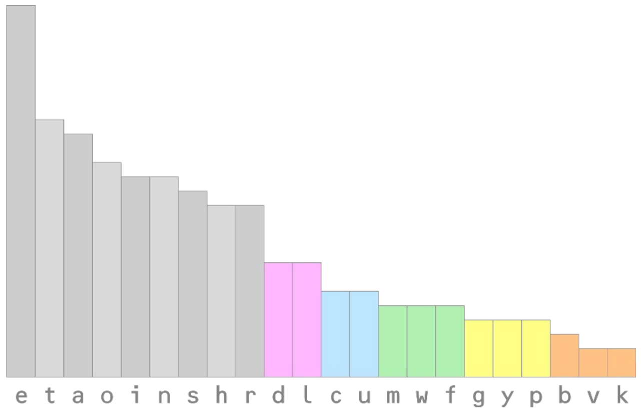
Reverse shift = E T A O I . . .

Trial-and-Error Caesar Cryptanalysis

- In Caesar ciphers, the key is only **a single shift** applied repeatedly

- Thus, there must be **one out of 26 reverse shifts** that, when applied:
  - Produces **understandable** plaintext
  - Matches the source language's observed **letter frequencies**

- Before computers, this was all done **by hand** via paper/pencil!



Ordered by frequency

Shift by **one** = F U B P J . . .

Reverse shift = E T A O I . . .

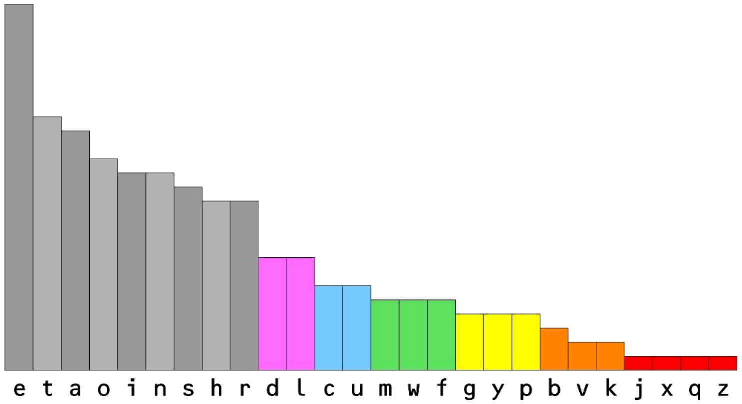
Trial-and-Error Caesar Cryptanalysis

- **Observation:** simple substitution ciphers don't alter **symbol frequency**

Ciphertext: FCWLRMCLWYMCFCSBCYMYKQJBFCGDACKGMX

C	Freq	P	Shift	Shift	Key
C	21%	E	E->C	24	Y
M	12%	?	?	?	?

21% >> 12% → "C" was probably "E"



Ordered by frequency

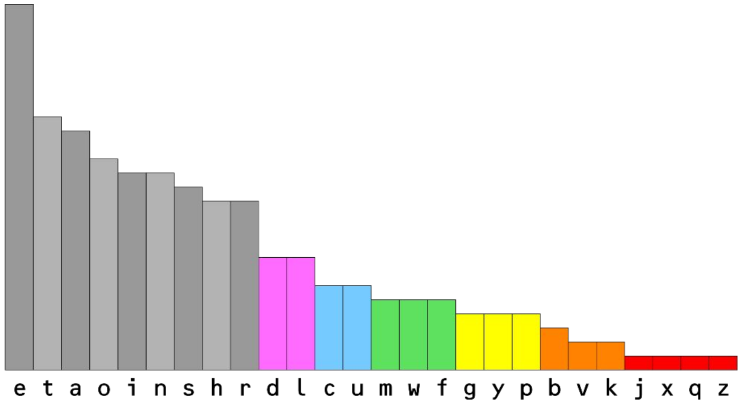
Trial-and-Error Caesar Cryptanalysis

- **Observation:** simple substitution ciphers don't alter **symbol frequency**

Ciphertext: LJSGUKJYSEKDLJGGAKWOGLHWLJNWFZLVEX

C	Freq	P	Shift	Shift	Key
L	15%	E	E->L	7	H
L	15%	T	T->L	18	S
J	13%	?	?	?	?

Look at most common letters ('E', 'T', 'A')



Ordered by frequency

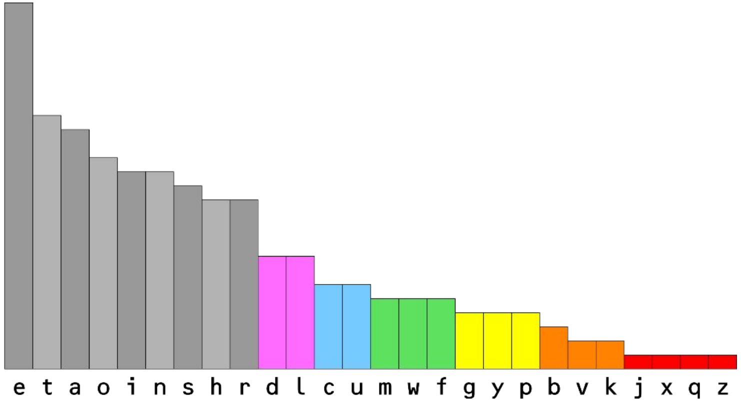
Trial-and-Error Caesar Cryptanalysis

- **Observation:** simple substitution ciphers don't alter **symbol frequency**

Ciphertext: WLKKAXVGACKLWGKWFFLQSGALWFGAAXWKJ

C	Freq	P	Shift	Key
W	15%	E,T,A	18,3,22	**S**,D,W
K	15%	E,T,A	6,17,10	?
A	13%	E,T,A	22,7,0	?

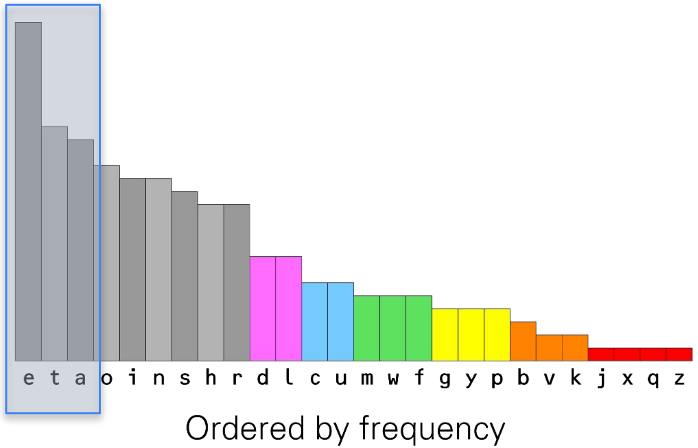
Look at most common letters ('E', 'T', 'A')



Ordered by frequency

Trial-and-Error Caesar Cryptanalysis

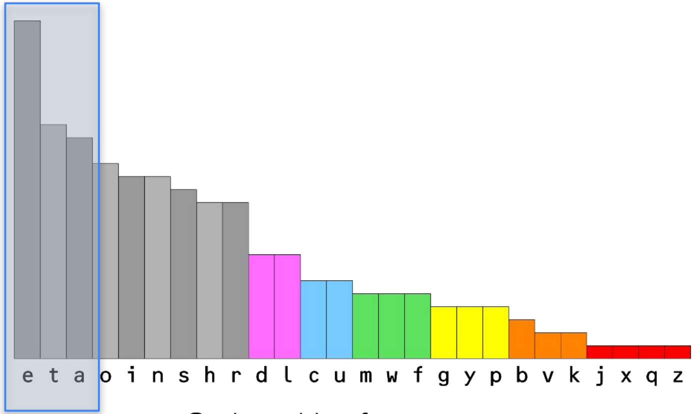
- **Narrowing down the search**
  - If a letter is most common by a **large margin**, it's probably a shifted **E**
  - Not a large margin? Try to find candidates for shifting **E**, **T**, and **A**



Trial-and-Error Caesar Cryptanalysis

- **Narrowing down the search**
  - If a letter is most common by a **large margin**, it's probably a shifted **E**
  - Not a large margin? Try to find candidates for shifting **E**, **T**, and **A**

- **Trial and error**
  - Perform **incremental decryption** and check
  - Does one candidate key reveal **more English**?



Ordered by frequency

Statistics-based Caesar Cryptanalysis

- A more elegant solution: **Chi-square Test**
  1. Generate **all 26 possible** reverse-shifted **strings** from the ciphertext

```
A: IYMBWXIXIH          N: VLZOJKVKVU
B: HXLAVWHWHG          O: UKYNIJUJUT
C: GWKZUVGVGF          P: TJXMHITITS
D: FVJYTUFUFE          Q: SIWLGHSHSR
E: EUIXSTETED          R: RHVKFGRGRQ
F: DTHWRSDSDC          S: QGUJEFQFQP
G: CSGVQRCRCB          T: PFTIDEPEPO
H: BRFUPQBQBA          U: OESHCDODON
I: AQETOPAPAZ          V: NDRGBCNCNM
J: ZPDSNOZOZY          W: MCQFABMBML
K: YOCRMNYNYX          X: LBPEZALALK
L: XNBQLMXMXW          Y: KAODYZKZKJ
M: WMAPKLWLWV          Z: JZNCXYJYJI
```

Statistics-based Caesar Cryptanalysis

- A more elegant solution: **Chi-square Test**
  1. Generate **all 26 possible reverse-shifted strings** from the ciphertext
  2. Calculate **observed letter frequencies** for all 26 letters, per all 26 reverse-shifted strings

```
A: IYMBWXIXIH          N: VLZOJKVKVU
B: HXLAVWHWHG          O: UKYNIJUJUT
C: GWKZUVGVGF          P: TJXMHITITS
D: FVJYTUFUFE          Q: SIWLGHSHSR
E: EUIXSTETED          R: RHVKFGRGRQ
F: DTHWRSDSDC          S: QGUJEFQFQP
G: CSGVQRCRCB          T: PFTIDEPEPO
H: BRFUPQBQBA          U: OESHCDODON
I: AQETOPAPAZ          V: NDRGBCNCNM
J: ZPDSNOZOZY          W: MCQFABMBML
K: YOCRMNYNYX          X: LBPEZALALK
L: XNBQLMXMXW          Y: KAODYZKZKJ
M: WMAPKLWLWV          Z: JZNCXYJYJI
```

Statistics-based Caesar Cryptanalysis

- A more elegant solution: **Chi-square Test**
  1. Generate **all 26 possible reverse-shifted strings** from the ciphertext
  2. Calculate **observed letter frequencies** for all 26 letters, per all 26 reverse-shifted strings
  3. Perform **chi-square test** on each string to find the **best-fit reverse-shift** (i.e., **lowest score**)

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

O_i = *observed* count for that letter (i.e., its total occurrences in the string)

E_i = *expected* count for that letter = **EnglishFreq_i** * **StringLength**

```
A: IYMBWXIXIH: 291.39      N: VLZOJKVKVU: 341.77
B: HXLAVWHWHG: 107.28      O: UKYNIJUJUT: 306.11
C: GWKZUVGVGF: 236.00      P: TJXMHITITS: 145.08
D: FVJYTUFUFE: 127.44      Q: SIWLGHSHSR: 25.58
E: EUIXSTETED: 77.16       R: RHVKFGRGRQ: 159.45
F: DTHWRSDSDC: 29.73       S: QGUJEFQFQP: 1035.24
G: CSGVQRCRCB: 157.77      T: PFTIDEPEPO: 50.52
H: BRFUPQBQBA: 487.57      U: OESHCDODON: 20.48
I: AQETOPAPAZ: 265.38      V: NDRGBCNCNM: 37.56
J: ZPDSNOZOZY: 1227.21     W: MCQFABMBML: 171.27
K: YOCRMNYNYX: 118.94      X: LBPEZALALK: 178.02
L: XNBQLMXMXW: 726.79      Y: KAODYZKZKJ: 722.45
M: WMAPKLWLWV: 71.82       Z: JZNCXYJYJI: 806.81
```

Statistics-based Caesar Cryptanalysis

- A more elegant solution: **Chi-square Test**
  1. Generate **all 26 possible reverse-shifted strings** from the ciphertext
  2. Calculate **observed letter frequencies** for all 26 letters, per all 26 reverse-shifted strings
  3. Perform **chi-square test** on each string to find the **best-fit reverse-shift** (i.e., **lowest score**)
  4. To get the key, convert the reverse-shift to its forward-shift!

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

$\mathbf{O_i}$ = *observed* count for that letter
(i.e., its total occurrences in the string)

$\mathbf{E_i}$ = *expected* count for that letter
= **$\mathbf{EnglishFreq_i}$** * **StringLength**

```
A: IYMBWXIXIH: 291.39     N: VLZOJKVKVU: 341.77
B: HXLAVWHWHG: 107.28     O: UKYNIJUJUT: 306.11
C: GWKZUVGVGF: 236.00     P: TJXMHITITS: 145.08
D: FVJYTUFUFE: 127.44     Q: SIWLGHSHSR: 25.58
E: EUIXSTETED: 77.16      R: RHVKFGRGRQ: 159.45
F: DTHWRSDSDC: 29.73      S: QGUJEFQFQP: 1035.24
G: CSGVQRCRCB: 157.77     T: PFTIDEPEPO: 50.52
H: BRFUPQBQBA: 487.57     U: OESHCDODON: 20.48
I: AQETOPAPAZ: 265.38     V: NDRGBCNCNM: 37.56
J: ZPDSNOZOZY: 1227.21    W: MCQFABMBML: 171.27
K: YOCRMNYNYX: 118.94     X: LBPEZALALK: 178.02
L: XNBQLMXMXW: 726.79     Y: KAODYZKZKJ: 722.45
M: WMAPKLWLWV: 71.82      Z: JZNCXYJYJI: 806.81
```

Attacking Ciphers

Brute-forcing
every possible key

Cryptanalysis

Questions?

Vigènere Cipher

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

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: `?????? ??????`

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

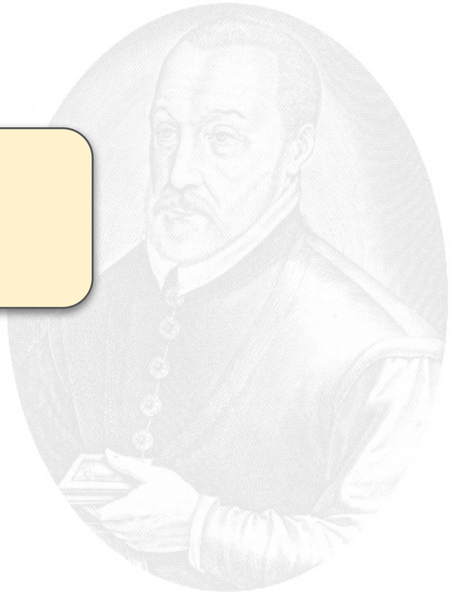
- For an **n**-letter keyword **k** …
  - Encryption: $\mathbf{c_i := (p_i + k_{i \bmod n}) \bmod 26}$
  - Decryption: $\mathbf{p_i := (c_i - k_{i \bmod n}) \bmod 26}$

- 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

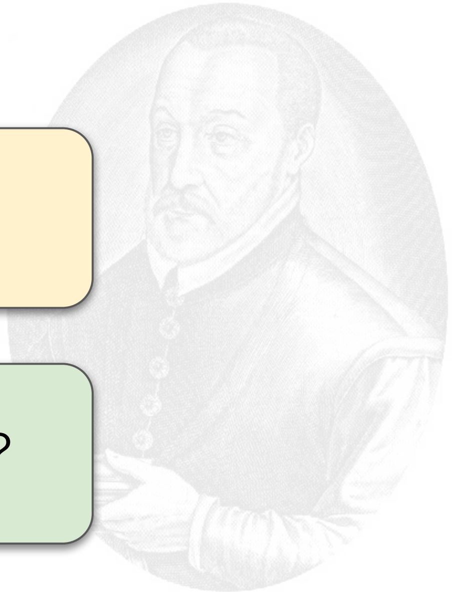
- First described by Bellaso in 1553
  - Later misattributed to Vigènere
- Encrypts succes
  ciphers determi
- For an **n**-letter k
  - Encryption: $c_i := (p_i + k_{i \bmod n}) \bmod 26$
  - Decryption: $p_i := (c_i - k_{i \bmod n}) \bmod 26$
- Example for k = ABC (i.e., $k_0 = 0, k_1 = 1, k_2 = 2$)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

Can you **brute-force** it?



Vigènere Ciphers

- First described by Bellaso in 1553
  - Later misattributed to Vigènere
- Encrypts succes… **ciphers** determi…
- For an **n**-letter k… keyword k …
  - Encryption: $c_i := (p_i + k_{i \bmod n}) \bmod 26$
  - Decryption: p_i …
- Example for k = …
  - Plain: bl…
  - +Key: 012012 012012
  - =Cipher: bcdbcd anczpp



Can you **brute-force** it?

What about **cryptanalysis**?

Vigènere Cipher Cryptanalysis

- Figure out how to **simplify a Vigenere cipher** into a **Caesar cipher**
  - Break it down into groups of letters—**grouped by column** (i.e., key-shift position)

```
ELFMASBQDXISZMNMHIBFEFQIMEUVNGLMLRETHAZAQPPDOTEGDEDONLYVZJNWHCKBLPPQWD
QZZGFFUKDWCIXWPZKKSIDYBGBATBUMOWFMYGFBPKYVELFHRHBMDMESJLQMZVHSXMCPDIOW
KJKLOVFGOWCBSIOOPAYVDEZWJYKORVFGOAYVHMMZJGDAEEORWYKQYWUYQOKQEXISZMNMCI
UINFCBZLJGWHKZEQFBPORMCIVDKFOVEISWJYKVOGMCOAXOELFRKGBPPMTDNGWXEPZUNSLJ
PHCMPOYUPRXVKXYZNIIWIAXBZXISXSDPCSPAWFNASSWSDACPPEEWJLRMESJZALDPPCESIS
XLXSOSUGGMOXPXWUUQPXSSAZYZYWBMEFQBSEUHDWNCOITKEXOJFROMYDKQXIEVAOKARSPR
BGBQEFFTKJOWYIPTPZOBSYHGQJSVLXFGKFDPPHVRAKBCRWBMEFQMGISHDMCBZHFOZTOIEW
MSXGGAVMCSSAVLPVFRPZOLFHFQKFFQYGFGPZOUELBHPZOGSEWSPZOECSOULWBAZRBGDWSA
YXNONJSMOEORYSXBASTGETVGASTGAKCBSIBAKMXBZJNCJWIBSIZOOCPWCPPCGAXOLVPIJV
DPPJJFOLDPFKSSWDSHPWUVAQRIGINOZWKUTWUOGWKVOQVGPZKDPXISSJYVRPFPKOCSTVFU
WJNTPWTHDWIJCIBYKFOWQLJGXSDPCSPAPAVMDFFTKJOTPEWWJYDPPHVRAUKTWWBTPWBBSI
NGWQSVREUZASCBTENVKMCMMVPYAFDPPHVRAEOMEWVDSADPSMTPKOVQYKUSWEKBELFZKUKT
LPMSUSXLEEMYOLYBSINOXGEBSMTJEGVMYXFBYGEVEISKWDDMCWPPYZKSCIBQPKGQELBBCW
BIYHWSJYOIYGFCJZSAXMORKXDMYWQSWCSVRSGVEKDQXITSNNOLTRWWALXIXXPFADKBPXPH
DWSADYFGHGGETXUSZLRMZHPFAVYVLPERKFXGVISOXSDAZWPTPWXMYXFFEFQKZRWSNKKBTS
OGDSVNHEZHDJYCRLQWLWCQYFVHEKZZZQQHHQDWWHZCQSBMZYUCBQYCCIMSIWXBMCXOHLOZ
```

column1

Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

- Shift: ##########################

column2

Cipher: BCDEFGHIJKLMNOPQRSTUVWXYZA

- Shift: ##########################

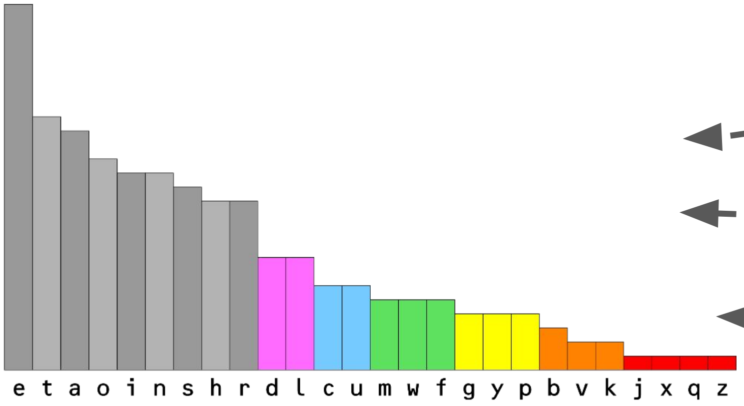
column3

Cipher: EFGHIJKLMNOPQRSTUVWXYZABCD

- Shift: ##########################

Vigènere Cipher Cryptanalysis

- Figure out how to **simplify a Vigenere cipher** into a **Caesar cipher**
  - Break it down into groups of letters—**grouped by column** (i.e., key-shift position)
  - Then, use frequency analysis to derive the key (shift) **for each letter-column**



Ordered by frequency

column1

Cipher:	DEFGHIJKLMNOPQRSTUVWXYZABC
- Shift:	33333333333333333333333333
= Plain:	ABCDEFGHIJKLMNOPQRSTUVWXYZ

K = 3

column2

Cipher:	BCDEFGHIJKLMNOPQRSTUVWXYZA
- Shift:	11111111111111111111111111
= Plain:	ABCDEFGHIJKLMNOPQRSTUVWXYZ

K = 1

column3

Cipher:	EFGHIJKLMNOPQRSTUVWXYZABCD
- Shift:	44444444444444444444444444
= Plain:	ABCDEFGHIJKLMNOPQRSTUVWXYZ

K = 4

Vigènere Cipher Cryptanalysis

- How to find **key length**? The **Kasiski method**
  - Published 1863 by Kasiski
  - **Repeated strings** in long plaintext will sometimes, by coincidence, be encrypted with same key letters

Distance = multiple of key length; can find multiple repeats to narrow down.

- Example:
  - Plain: **CRYPTO**ISSHORTFOR**CRYPTO**GRAPHY
  - +Key: **ABCDAB**CDABCDABCD**ABCDAB**CDABCD
  - =Cipher: CSASTPKVSIQUTGQUCSASTPIUAQJB

Distance = 16

Key length = ???

Vigènere Cipher Cryptanalysis

- How to find **key length**? The **Kasiski method**
  - Published 1863 by Kasiski
  - **Repeated strings** in long plaintext will sometimes, by coincidence, be encrypted with same key letters

Distance = multiple of key length; can find multiple repeats to narrow down.

- Example:
  - Plain: **CRYPTO**ISSHORTFOR**CRYPTO**GRAPHY
  - +Key: **ABCDAB**CDABCDABCD**ABCDAB**CDABCD
  - =Cipher: **CSASTP**KVSIQUTGQU**CSASTP**IUAQJB

Distance = 16

Key length = **16, 8, 4, 2,** or **1**

Kasiski Method

- Let's look at an example:

Plaintext =

```
THERE ARETW OWAYS OFCON STRUC TINGA SOFTW AREDE SIGNO NEWAY
ISTOM AKEIT SOSIM PLETH ATTHE REARE OBVIO USLYN ODEFI CIENC
IESAN DTHEO THERW AYIST OMAKE ITSOC OMPLI CATED THATT HEREA
RENOO BVIOU SDEFI CIENC IESTH EFIRS TMETH ODISF ARMOR EDIFF
```

 = **SYSTE M**

Ciphertext =

```
LFWKI MJCLP SISWK HJOGL KMVGU RAGKM KMXMA MJCVX WUYLG GIISW
ALXAE YCXMF KMKBQ BDCLA EFLFW KIMJC GUZUG SKECZ GBWYM OACFV
MQKYF WXTWM LAIDO YQBWF GKSDI ULQGV SYHJA VEFWB LAEFL FWKIM
JCFHS NNGGN WPWDA VMQFA AXWFZ CXBVE LKWML AVGKY EDEMJ XHUXD
```

Kasiski Method

- Let's look at an example:

p	THERE	ARETW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	LFWKI	MJCLP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	MJCVX	WUYLG	GIISW

p	ISTOM	AKEIT	SOSIM	PLETH	ATTHE	REARE	OBVIO	USLYN	ODEFI	CIENC
	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST
c	ALXAE	YCXMF	KMKBQ	BDCLA	EFLFW	KIMJC	GUZUG	SKECZ	GBWYM	OACFV

p	IESAN	DTHEO	THERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THATT	HEREA
	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM
c	MQKYF	WXTWM	LAIDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEFL	FWKIM

p	RENOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TMETH	ODISF	ARMOR	EDIFF
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD

Kasiski Method

- Let’s look at an example:

p	**THERE**	**ARE**TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	LFWKI	MJCLP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	MJCVX	WUYLG	GIISW
p	ISTOM	AKEIT	SOSIM	PLETH	AT**THE**	**REARE**	OBVIO	USLYN	ODEFI	CIENC
	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST
c	ALXAE	YCXMF	KMKBQ	BDCLA	EFLFW	KIMJC	GUZUG	SKECZ	GBWYM	OACFV
p	IESAN	DTHEO	THERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THAT**T**	**HEREA**
	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM
c	MQKYF	WXTWM	LAIDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEFL	FWKIM
p	**RE**NOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TMETH	ODISF	ARMOR	EDIFF
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD

Kasiski Method

- Let's look at an example:

p	**THERE**	**ARE**TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
	SYSTE	**MSY**ST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	LFWKI	MJCLP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	MJCVX	WUYLG	GIISW
p	ISTOM	AKEIT	SOSIM	PLETH	AT**THE**	**REARE**	OBVIO	USLYN	ODEFI	CIENC
	STEMS	YSTEM	SYSTE	MSYST	EM**SYS**	**TEMSY**	STEMS	YSTEM	SYSTE	MSYST
c	ALXAE	YCXMF	KMKBQ	BDCLA	EFLFW	KIMJC	GUZUG	SKECZ	GBWYM	OACFV
p	IESAN	DTHEO	THERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THAT**T**	**HEREA**
	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEM**S**	**YSTEM**
c	MQKYF	WXTWM	LAIDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEFL	FWKIM
p	**RE**NOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TMETH	ODISF	ARMOR	EDIFF
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD

Kasiski Method

- Let’s look at an example:

p	**THERE**	**ARE**TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
	SYSTE	**MSY**ST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	**LFWKI**	**MJC**LP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	MJCVX	WUYLG	GIISW
p	ISTOM	AKEIT	SOSIM	PLETH	AT**THE**	**REARE**	OBVIO	USLYN	ODEFI	CIENC
	STEMS	YSTEM	SYSTE	MSYST	EM**SYS**	**TEMSY**	STEMS	YSTEM	SYSTE	MSYST
c	ALXAE	YCXMF	KMKBQ	BDCLA	EF**LFW**	**KIMJC**	GUZUG	SKECZ	GBWYM	OACFV
p	IESAN	DTHEO	THERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THAT**T**	**HEREA**
	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEM**S**	**YSTEM**
c	MQKYF	WXTWM	LAIDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEF**L**	**FWKIM**
p	**RE**NOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TMETH	ODISF	ARMOR	EDIFF
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	**JC**FHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD

Kasiski Method

- Let's look at an example:

p	**THERE**	**ARE**TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
	SYSTE	**MSY**ST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	**LFWKI**	**MJC**LP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	MJCVX	WUYLG	GIISW
p	ISTOM	AKEIT	SOSIM	PLETH	AT**THE**	**REARE**	OBVIO	USLYN	ODEFI	CIENC
	STEMS	YSTEM	SYSTE	MSYST	EM**SYS**	**TEMSY**	STEMS	YSTEM	SYSTE	MSYST
c	ALXAE	YCXMF	KMKBQ	BDCLA	EF**LFW**	**KIMJC**	GUZUG	SKECZ	GBWYM	OACFV
p	IESAN	DTH**EO**	**TH**ERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THAT**T**	**HEREA**
	EMSYS	TEM**SY**	**ST**EMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEM**S**	**YSTEM**
c	MQKYF	WXT**WM**	**LA**IDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEF**L**	**FWKIM**
p	**RE**NOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TM**ETH**	**O**DISF	ARMOR	EDIFF
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SY**STE**	**M**SYST	EMSYS	TEMSY
c	**JC**FHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LK**WML**	**A**VGKY	EDEMJ	XHUXD

Kasiski Method

- Let’s look at an example:

p	**THERE**	**ARE**TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	**ARE**DE	SIGNO	NEWAY
	SYSTE	**MSY**ST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	**MSY**ST	EMSYS	TEMSY
c	**LFWKI**	**MJC**LP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	**MJC**VX	WUYLG	GIISW
p	ISTOM	AKEIT	SOSIM	PLETH	AT**THE**	**REARE**	OBVIO	USLYN	ODEFI	CIENC
	STEMS	YSTEM	SYSTE	MSYST	EM**SYS**	**TEMSY**	STEMS	YSTEM	SYSTE	MSYST
c	ALXAE	YCXMF	KMKBQ	BDCLA	EF**LFW**	**KIMJC**	GUZUG	SKECZ	GBWYM	OACFV
p	IESAN	DTH**EO**	**TH**ERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THAT**T**	**HEREA**
	EMSYS	TEM**SY**	**ST**EMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEM**S**	**YSTEM**
c	MQKYF	WXT**WM**	**LA**IDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEF**L**	**FWKIM**
p	**RE**NOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TM**ETH**	**O**DISF	ARMOR	EDIFF
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SY**STE**	**M**SYST	EMSYS	TEMSY
c	**JC**FHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LK**WML**	**A**VGKY	EDEMJ	XHUXD

Kasiski Method

- Let’s look at an 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**
	EMSYS	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
	SYSTE	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

Kasiski Method

- How can we find the key length?

Substring	Length	Positions	Distances
LWFKIMJC	8		
WMLA	4		
MJC	3		
ISW	3		
KMK	3		
VMQ	3		

Kasiski Method

- Create a table of substring **positions**; then calculate their **distances**

Substring	Length	Positions	Distances
LWFKIMJC	8	(0,72) (72,144) (0,144)	
WMLA	4	(108,182)	
MJC	3	(5,35)	
ISW	3	(11,47)	
KMK	3	(28,60)	
VMQ	3	(99,165)	

Kasiski Method

- Create a table of substring **positions**; then calculate their **distances**

Substring	Length	Positions	Distances
LWFKIMJC	8	(0,72) (72,144) (0,144)	72, 72, 144
WMLA	4	(108,182)	74
MJC	3	(5,35)	30
ISW	3	(11,47)	36
KMK	3	(28,60)	32
VMQ	3	(99,165)	66

Kasiski Method

- Find the **factors** (aka divisors) of each substring distance

Substring	Length	Distance Factors	Distances
LWFKIMJC	8		72, 72, 144
WMLA	4		74
MJC	3		30
ISW	3		36
KMK	3		32
VMQ	3		66

Kasiski Method

- Find the **factors** (aka divisors) of each substring distance

Substring	Length	Distance Factors	Distances
LWFKIMJC	8	1,2,3,4,6,8,9,12,18,24,36,72	72, 72, 144
WMLA	4	1, 2, 37, 74	74
MJC	3	1, 2, 3, 5, 6, 10, 15, 30	30
ISW	3	1, 2, 3, 4, 6, 9, 12, 18, 36	36
KMK	3	1, 2, 4, 8, 16, 32	32
VMQ	3	1, 2, 3, 6, 11, 22, 33, 66	66

Kasiski Method

- Cull abnormalities; recall WMLA is from two **different** plaintext strings!

Substring	Length	Distance Factors	Distances
LWFKIMJC	8	1,2,3,4,6,8,9,12,18,24,36,72	72, 72, 144
WMLA	4	1, 2, 37, 74	74
MJC	3	1, 2, 3, 5, 6, 10, 15, 30	30
ISW	3	1, 2, 3, 4, 6, 9, 12, 18, 36	36
KMK	3	1, 2, 4, 8, 16, 32	32
VMQ	3	1, 2, 3, 6, 11, 22, 33, 66	66

Kasiski Method

- Compute the **greatest common factor** of substring distances

Substring	Length	Distance Factors	Distances
LWFKIMJC	8	1,2,3,4,6,8,9,12,18,24,36,72	72, 72, 144
MJC	3	1, 2, 3, 5, 6, 10, 15, 30	30
ISW	3	1, 2, 3, 4, 6, 9, 12, 18, 36	36
VMQ	3	1, 2, 3, 6, 11, 22, 33, 66	66

Kasiski Method

- Compute the **greatest common factor** of substring distances

Substring	Length	Distance Factors	Distances
LWFKIMJC	8	1,2,3,4,**6**,8,9,12,18,24,36,72	72, 72, 144
MJC	3	1, 2, 3, 5, **6**, 10, 15, 30	30
ISW	3	1, 2, 3, 4, **6**, 9, 12, 18, 36	36
VMQ	3	1, 2, 3, **6**, 11, 22, 33, 66	66

Kasiski Method

- To find outliers, you can make a table of **occurrences of distance factors**

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					

Kasiski Method

- To find outliers, you can make a table of **occurrences of distance factors**

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					

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					

Kasiski Method

- With key length in hand, divide ciphertext into **key-sized chunks**

```
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
```

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		

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
```

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
```

Chi-square on Reverse-shifted Column Strings

Column #1 String (with a **zero shift**): LJSGUKJYSEKDLJGGAKWOGLHWLJNWFZLVEX

```
{ "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 }
```

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

O_L = observed count for letter 'L' = 5.0

E_L = expected count for letter 'L'

= **EnglishFreq$_L$** * **ColumnStringLength**

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

= **1.3685**

Chi-square on Reverse-shifted Column Strings

Column #1 String (with a **zero shift**): LJSGUKJYSEKDLJGGAKWOGLHWLJNWFZLVEX

```
{ "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 }
```

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

O_L = observed count for letter 'L' = **5.0**

E_L = expected count for letter 'L'

= **EnglishFreq$_L$** * **ColumnStringLength**

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

= **1.3685**

X^2_L = (5.0 – 1.3685)^2 / 1.3685

= **9.6367**

1. Repeat for all other letters.
2. Sum = X^2 score for that shift
3. Repeat for the 25 other shifts
4. **Lowest score** **= the correct shift!**

Questions?

Next time on CS 4440...

One-time Pads, Transposition and Block Ciphers