

Week 2: Lecture B

Message Confidentiality

Thursday, August 28, 2025

Announcements: Project 1

- **Project 1: Crypto** released (see [Assignments](#) page on course website)
 - **Deadline:** Thursday, September 18th by 11:59PM

Project 1: Cryptography

Deadline: Thursday, September 18 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.

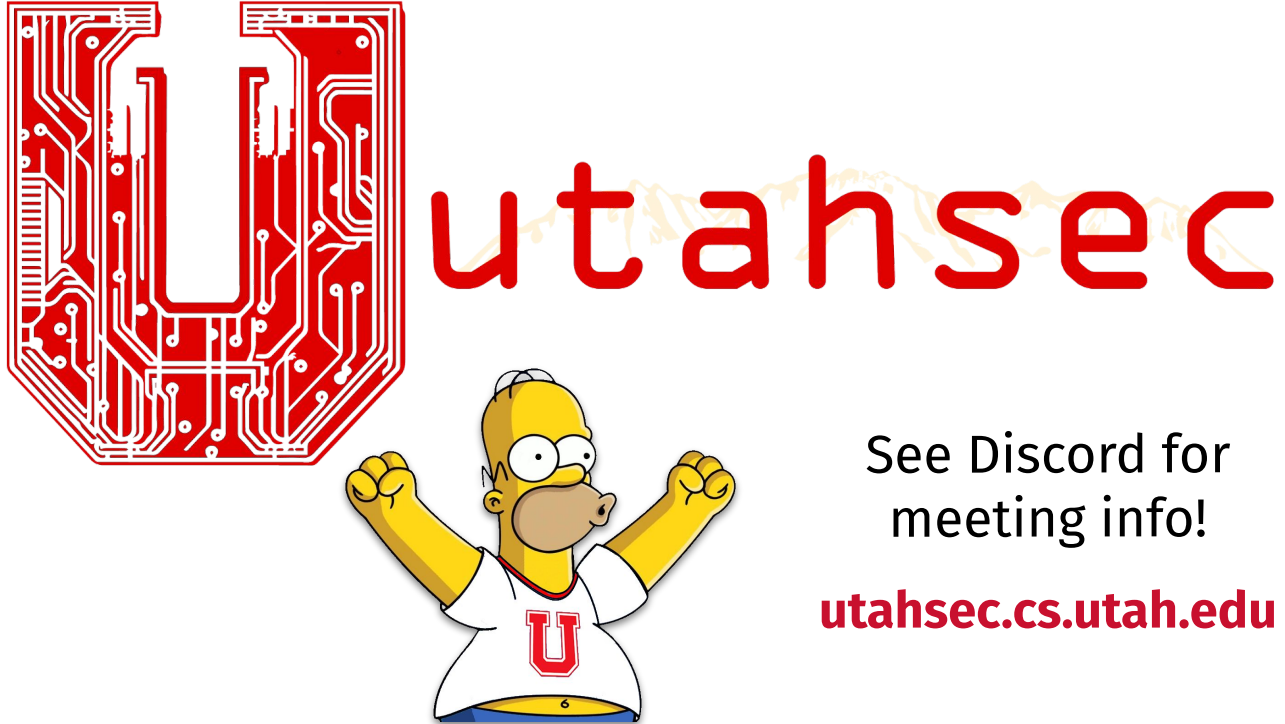
Helpful Resources

- [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-Damgård](#)
 - [Length Extension Attacks](#)
 - [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



See Discord for
meeting info!

utahsec.cs.utah.edu

Announcements



AI ACCELERATORS 101

With Daniel Kroening
Senior Principal Applied Scientist, Amazon



Kahlert

DISTINGUISHED COLLOQUIUM

Biography

Daniel Kroening is a Senior Principal Applied Scientist at Amazon, where he works on the correctness of the Neuron Compiler for distributed training and inference. Prior to joining Amazon, he worked as a Professor of Computer Science at the University of Oxford and is the co-founder of Diffblue Ltd., a University spinout that develops AI that targets code and code-like artefacts. He wrote the CBMC (for C), J BMC (for Java) and EBMC (for SystemVerilog) model checkers; CBMC is the engine of Kani (for verifying unsafe Rust). He has received the Semiconductor Research Corporation (SRC) Inventor Recognition Award, an IBM Faculty Award, a Microsoft Research SEIF Award, the Wolfson Research Merit Award, and the Rance Cleaveland Test-of-Time tool award. He serves on the CAV steering committee and was co-chair of FLOC 2018, EIC of Springer FMSD, and is co-author of the textbooks on Decision Procedures and Model Checking.



KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

Abstract

LLMs are arguably among the largest technology investments since the moon landing, and rely on custom hardware accelerators both for training and inference. The talk will cover accelerating LLM transformer architectures using the combination of a compiler and a systolic compute array. The key enabler to achieving meaningful performance using the systolic compute array are deep program analyses of the model architecture in the Neuron Compiler. I will briefly report on our effort to build a verified (using Lean) compiler from XLA/HLO to the Trainium ISA.

**Thursday, September 11,
2025**

Kennecott Mechanical Engineering Building (MEK)

Room 3550

5:15 PM Speaker

6:15 PM Pizza



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:** ???



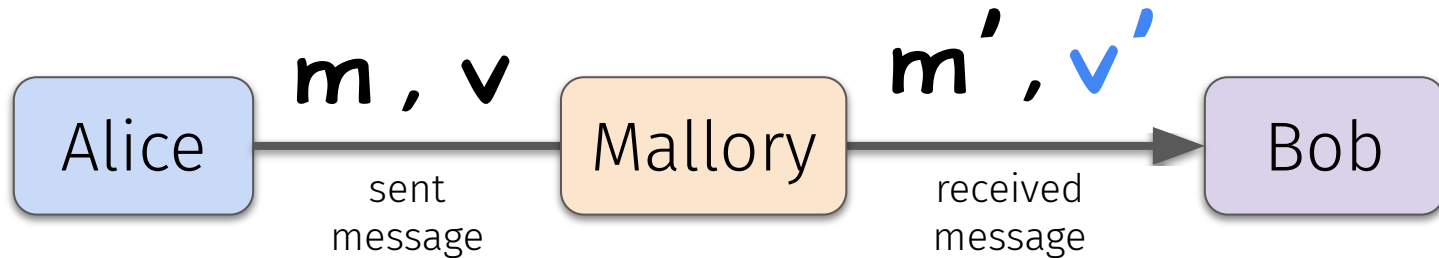
Message Integrity

- **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**



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**—easier/faster to use/share (fewer functions)
- **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**—easier/faster to use/share (fewer functions)
- **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

Think of these as
abstract categories

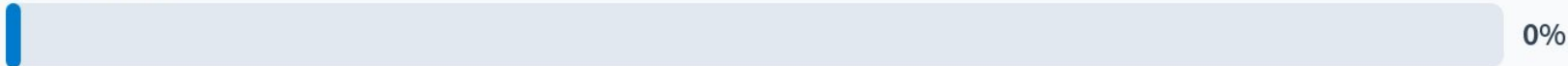
■ Idea 2: **Pseudo-random Function Family (PRF):**

- **Subset so large** it seems to be a random function
- Mallory knows set, but not **which function** is chosen
- **Practical**—easier/faster to use/share (fewer functions)
- **Secure**—brute-forcing insanely costly (but possible)

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?

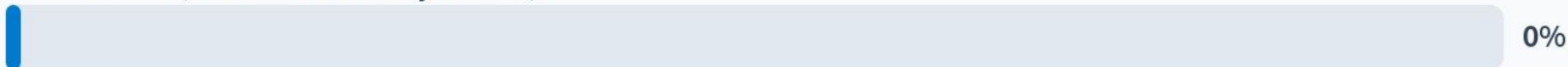
More secure



Equally secure



Less secure (but still extremely secure)



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):

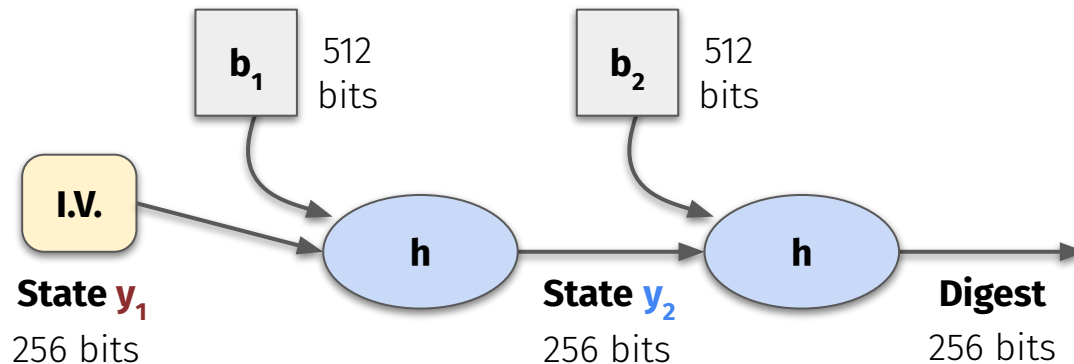
- **Subset so large** it seems to be a random function
- Mallory knows set, but not **which function** is chosen
- **Practical**—easier/faster to use/share (fewer functions)
- **Secure**—brute-forcing insanely costly (but possible)
- **Less secure** than random functions—**but very secure**
 - Still 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)

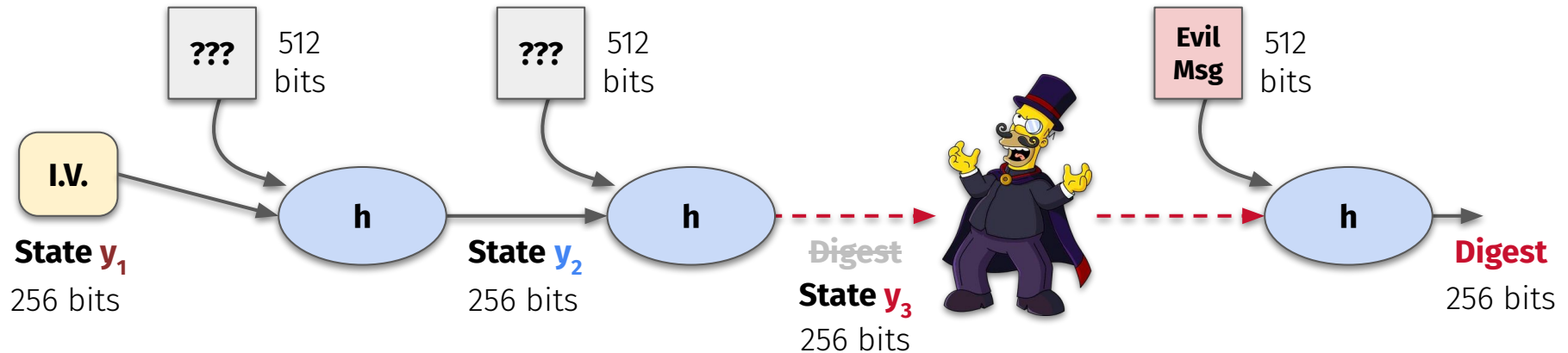
Candidate $f(m)$: Merkle–Damgård Hashes

- **Merkle–Damgård construction:** digest formed from **the last chaining value**
 - Partition message into **512-bit blocks**, with the last block **padded to 512 bits**



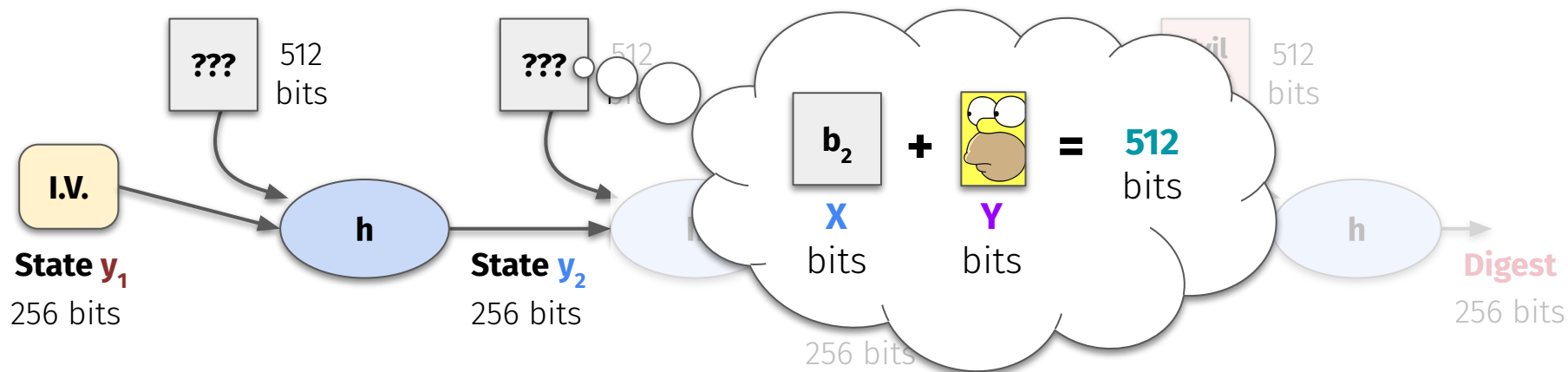
Merkle–Damgård Hashes: Length Extension Attacks

- **Merkle–Damgård construction:** digest formed from **the last chaining value**
 - Partition message into **512-bit blocks**, with the last block **padded to 512 bits**
- **Vulnerability:** nothing stopping Mallory from **continuing the hash chain...**
 - Mallory **doesn't know** previous blocks' plaintext



Merkle–Damgård Hashes: Length Extension Attacks

- **Merkle–Damgård construction:** digest formed from **the last chaining value**
 - Partition message into **512-bit blocks**, with the last block **added to 512 bits**
- **Vulnerability:** nothing stopping Mallory from **continuing the hash chain...**
 - Mallory **doesn't know** previous blocks' plaintext—only **how much padding was added**



Merkle–Damgård Hashes: Length Extension Attacks

Mallory's resulting hash **digest**

==

hash (original || pad || evil)

m

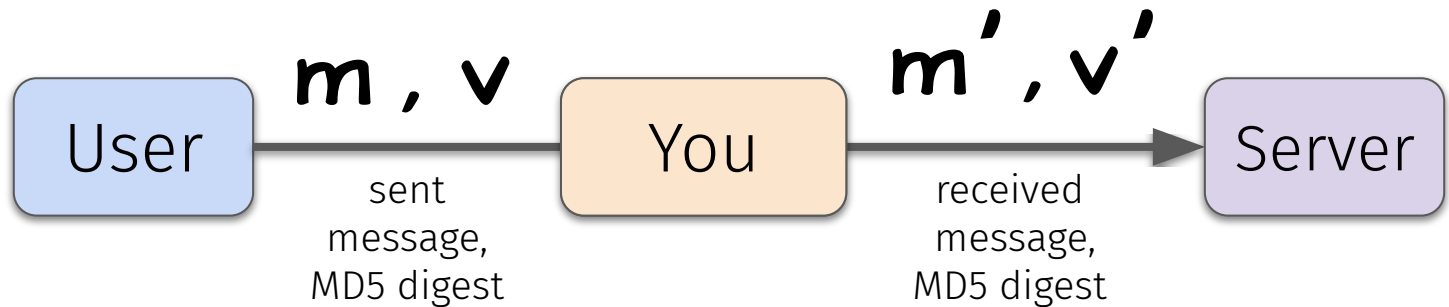
m'

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



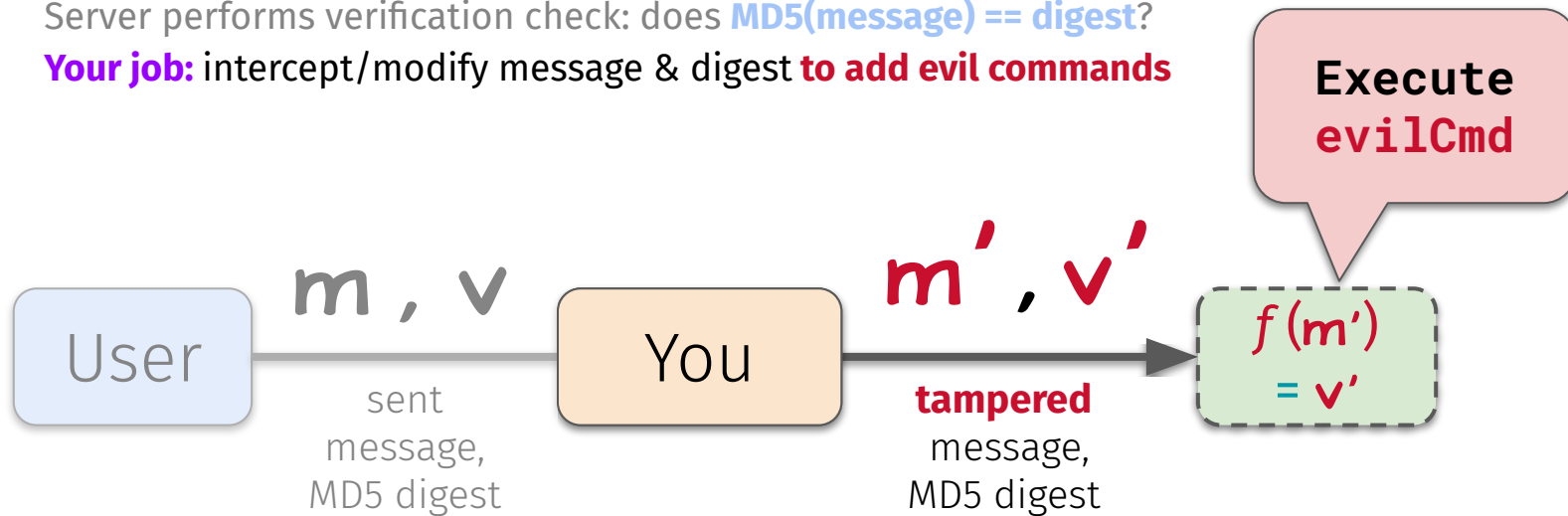
Merkle–Damgård Hashes: Length Extension Attacks

- **Project 1 Part 2:** attack a server that accepts commands
 - User provides message: **secret password + plaintext commands list**
 - 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: **secret password + plaintext commands list**
 - 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**



Other Hashing Pitfalls

- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**

Other Hashing Pitfalls

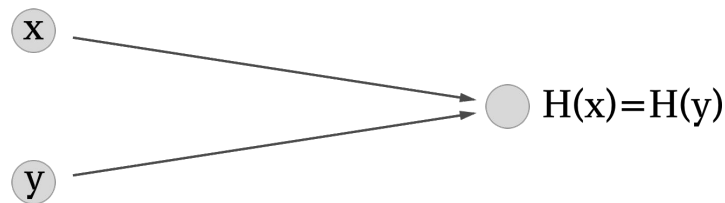
- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**
 1. **Collision Attack**
 - ???

Other Hashing Pitfalls

- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**

1. **Collision Attack**

- Mallory finds $m_1 \neq m_2$
such that $h(m_1) = h(m_2)$



Other Hashing Pitfalls

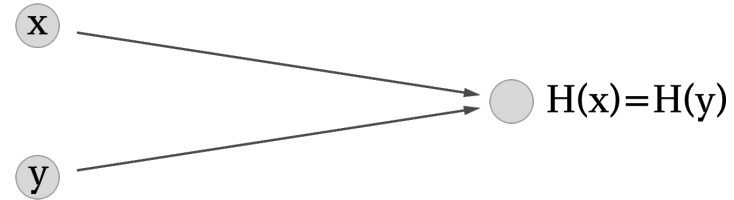
- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**

1. **Collision Attack**

- Mallory finds $m_1 \neq m_2$
such that $h(m_1) = h(m_2)$

2. **Second Pre-image Attack**

- ???



Other Hashing Pitfalls

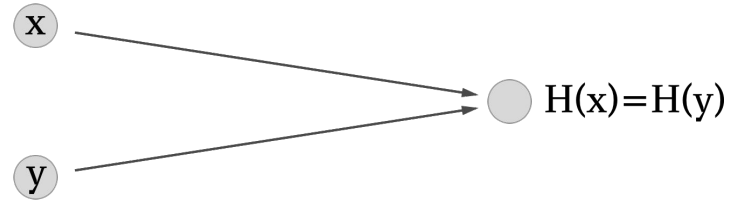
- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**

1. **Collision Attack**

- Mallory finds $m_1 \neq m_2$
such that $h(m_1) = h(m_2)$

2. **Second Pre-image Attack**

- Given m_1 , Mallory finds $m_2 \neq m_1$
such that $h(m_1) = h(m_2)$

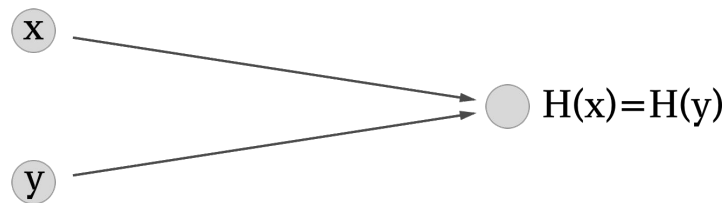


Other Hashing Pitfalls

- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**

1. **Collision Attack**

- Mallory finds $m_1 \neq m_2$
such that $h(m_1) = h(m_2)$



2. **Second Pre-image Attack**

- Given m_1 , Mallory finds $m_2 \neq m_1$
such that $h(m_1) = h(m_2)$

3. **First Pre-image Attack**

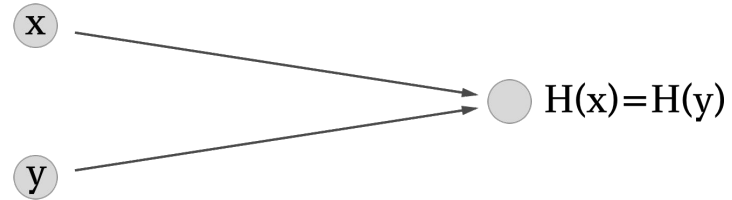
- ???

Other Hashing Pitfalls

- **To be safe**, a hash (Merkle-Damgård or not) must **withstand what attacks?**

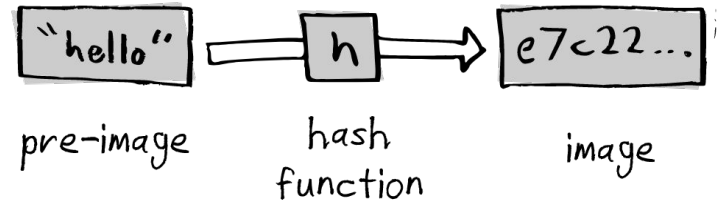
1. **Collision Attack**

- Mallory finds $m_1 \neq m_2$
such that $h(m_1) = h(m_2)$



2. **Second Pre-image Attack**

- Given m_1 , Mallory finds $m_2 \neq m_1$
such that $h(m_1) = h(m_2)$

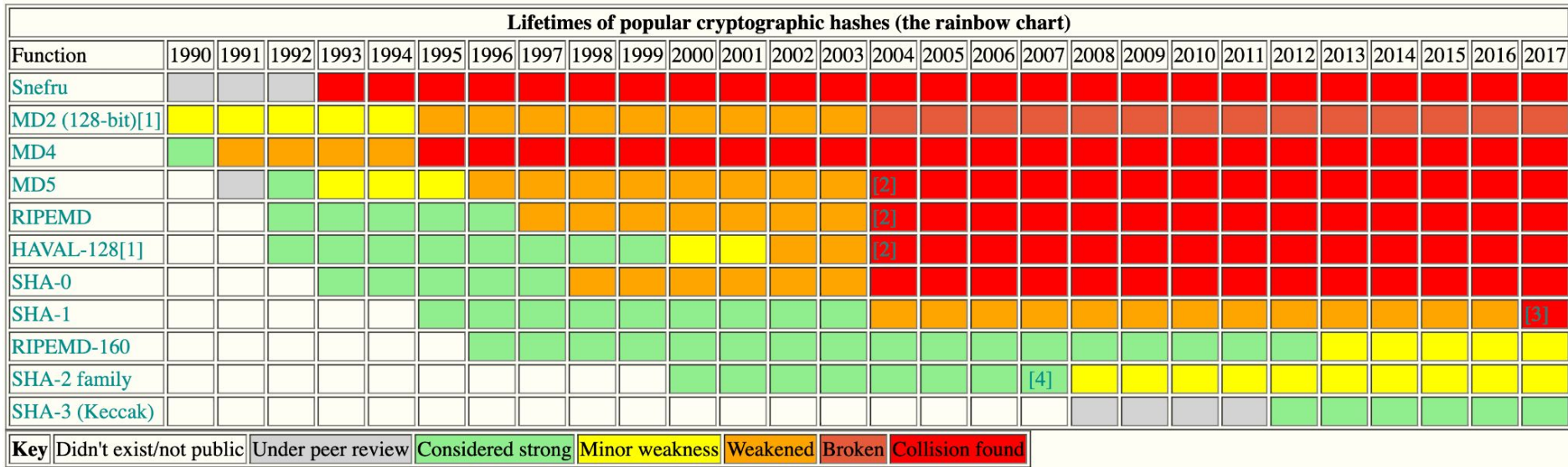


3. **First Pre-image Attack**

- Given $h(m)$, Mallory finds m

Other Hashing Pitfalls

- **MD5, SHA-1**, and many other hash functions **have long been defeated**
 - Collisions, first/second pre-image, and length extension attacks



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	$v' \neq v$	$m' \neq m$	$f(m') \neq v'$
Hash Collision	$v' \neq v$	$m' \neq m$	$f(m') = v'$
Length Extension	$v' = v$	$m' \neq m$	$f(m') = v'$

Exercise: Attacks on Message Integrity

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

Exercise: Attacks on Message Integrity

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

Exercise: Attacks on Message Integrity

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

Mitigating Length Extension: HMACs

- **HMAC:** keyed-hash message authentication code
 - **Improvements:** **nested construction** leveraging a **secret key** and **two paddings**
 - **Definition:** $\text{HMAC}(m) = \text{SHA256}((k \oplus \text{pad}_{\text{outer}}) || \text{SHA256}((k \oplus \text{pad}_{\text{inner}}) || m))$

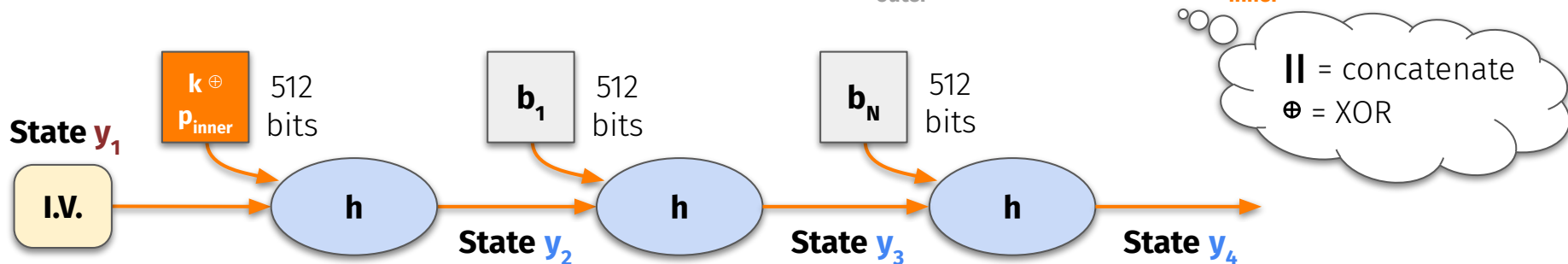
Not expected to memorize HMAC's low-level details...
but understand its **improvements over Merkle-Damgård!**

Mitigating Length Extension: HMACs

- **HMAC:** keyed-hash message authentication code

- **Improvements:** **nested construction** leveraging a **secret key** and **two paddings**

- **Definition:** $\text{HMAC}(m) = \text{SHA256}((k \oplus \text{pad}_{\text{outer}}) || \text{SHA256}((k \oplus \text{pad}_{\text{inner}}) || m))$



Can Mallory length extend at this point (state Y-4)?

Only if she knows the key!

0%

Yes, regardless of having the key!

0%

None of the above

0%

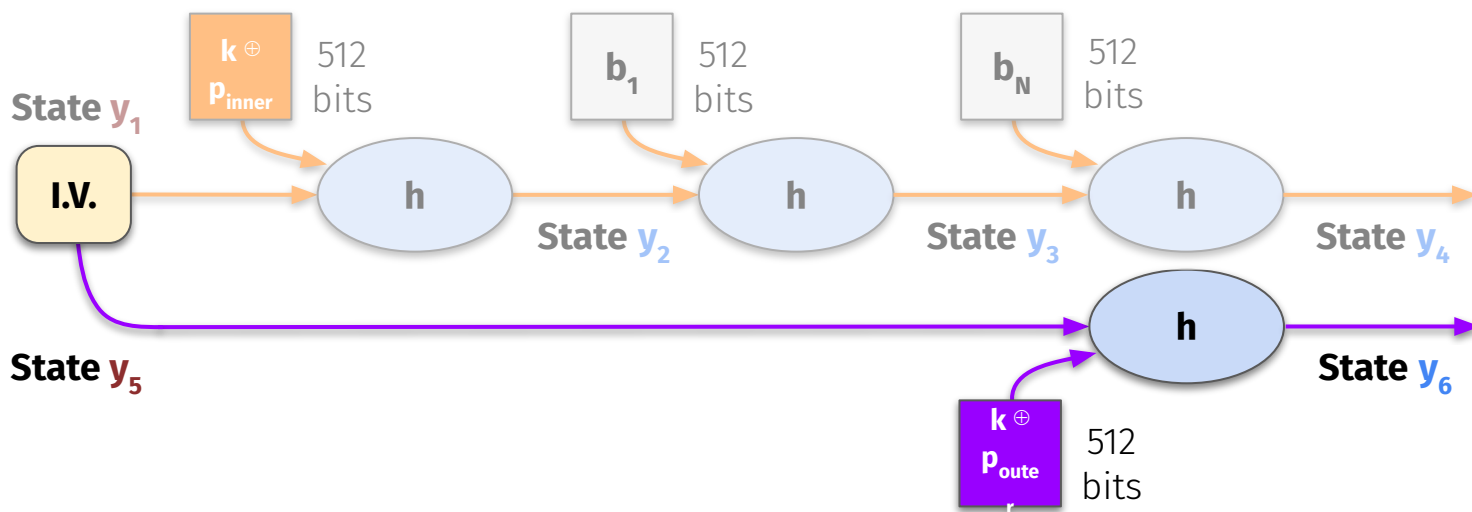


Mitigating Length Extension: HMACs

- **HMAC:** keyed-hash message authentication code

- **Improvements:** **nested construction** leveraging a **secret key** and **two paddings**

- **Definition:** $\text{HMAC}(m) = \text{SHA256}((k \oplus \text{pad}_{\text{outer}}) || \text{SHA256}((k \oplus \text{pad}_{\text{inner}}) || m))$

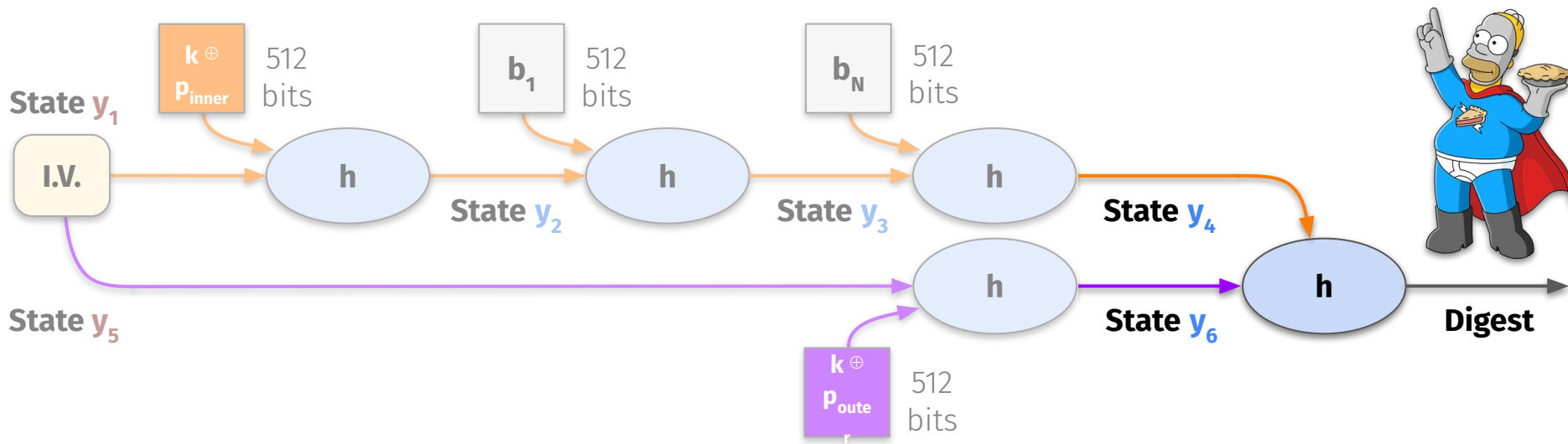


Mitigating Length Extension: HMACs

- **HMAC:** keyed-hash message authentication code

- **Improvements:** **nested construction** leveraging a **secret key** and **two paddings**

- **Definition:** $\text{HMAC}(m) = \text{SHA256}((k \oplus \text{pad}_{\text{outer}}) || \text{SHA256}((k \oplus \text{pad}_{\text{inner}}) || m))$

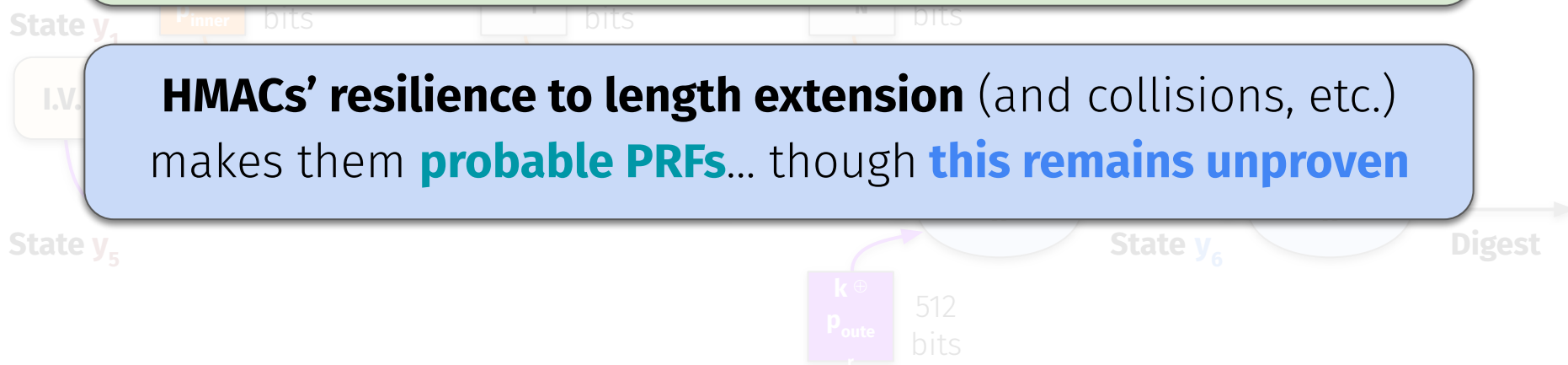


Mitigating Length Extension: HMACs

- **HMAC**: keyed-hash message authentication code

Merkle-Damgard's digests = **continuable internal states**.
HMAC's digests = just the **outer state**; **can't be continued!**

HMACs' resilience to length extension (and collisions, etc.)
makes them **probable PRFs**... though **this remains unproven**



Questions?



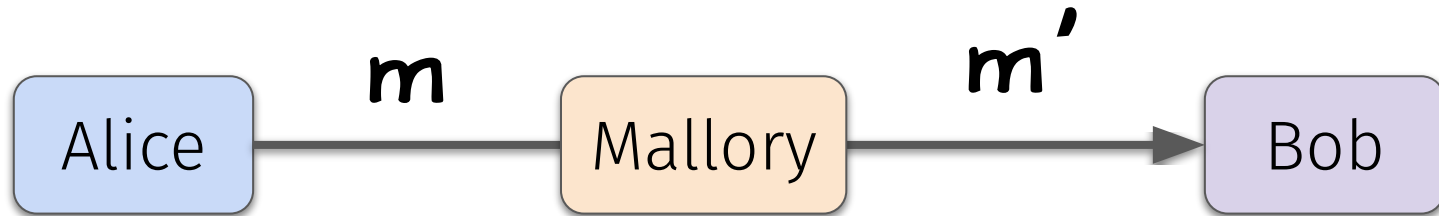
This time on CS 4440...

Message Confidentiality
Simple Substitution Ciphers
Cipher Cryptanalysis

Message Confidentiality

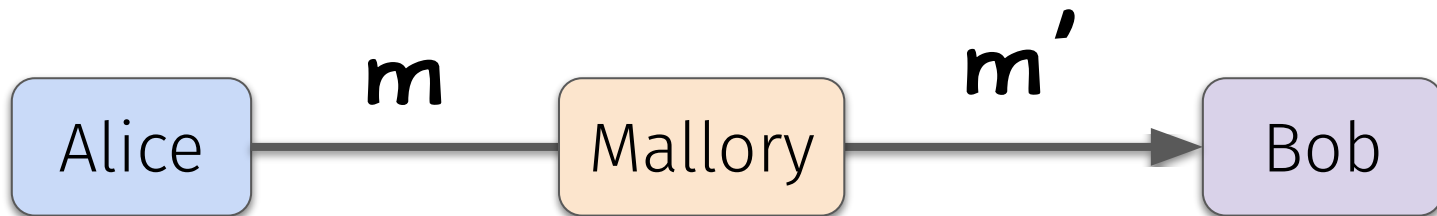
Message Confidentiality

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



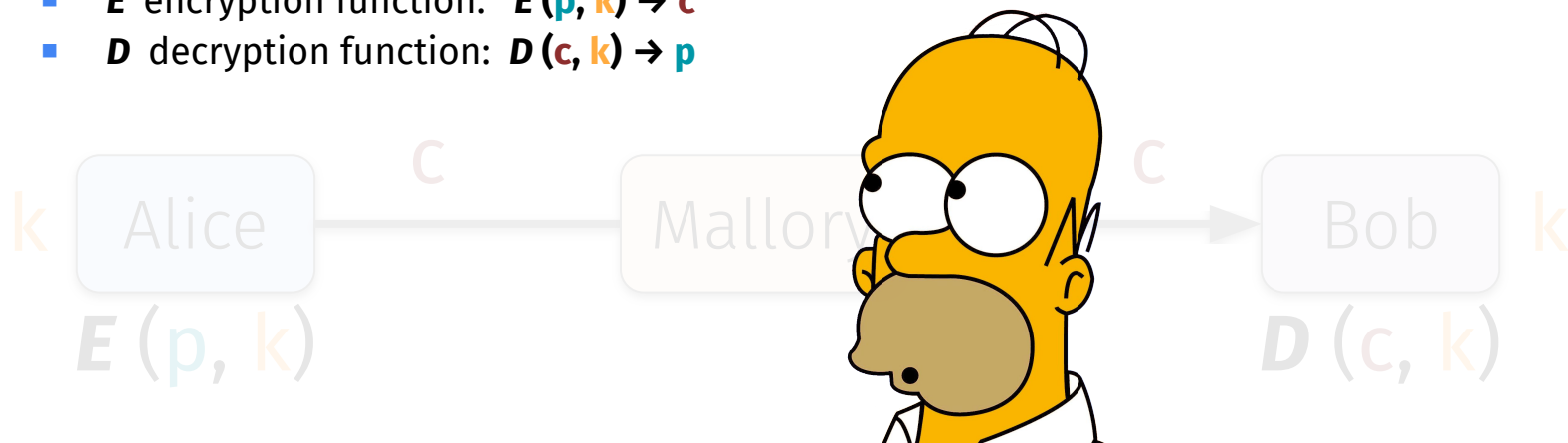
Message Confidentiality

- **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: $D(c, k) \rightarrow p$



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



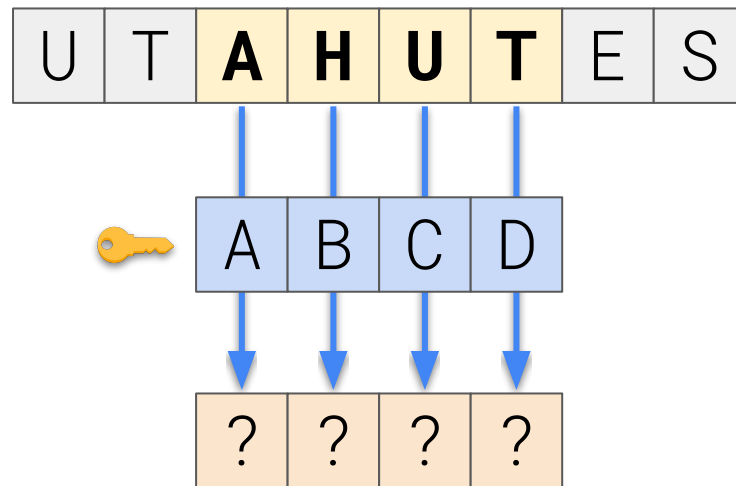
A	B	C	D
?	?	?	?

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

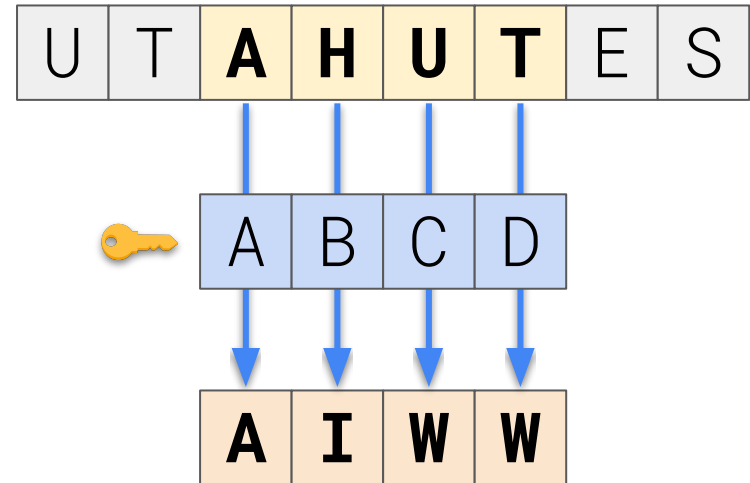


Substitution Ciphers

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

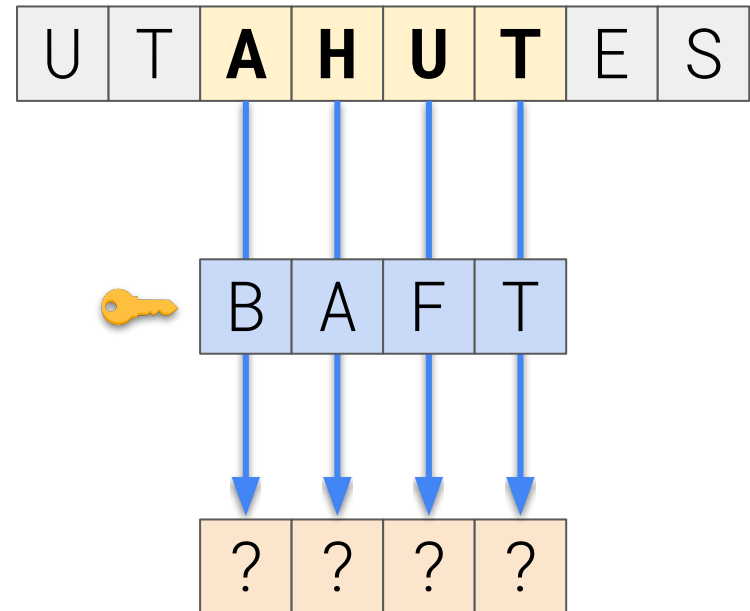


A	B	C	D
0	1	2	3



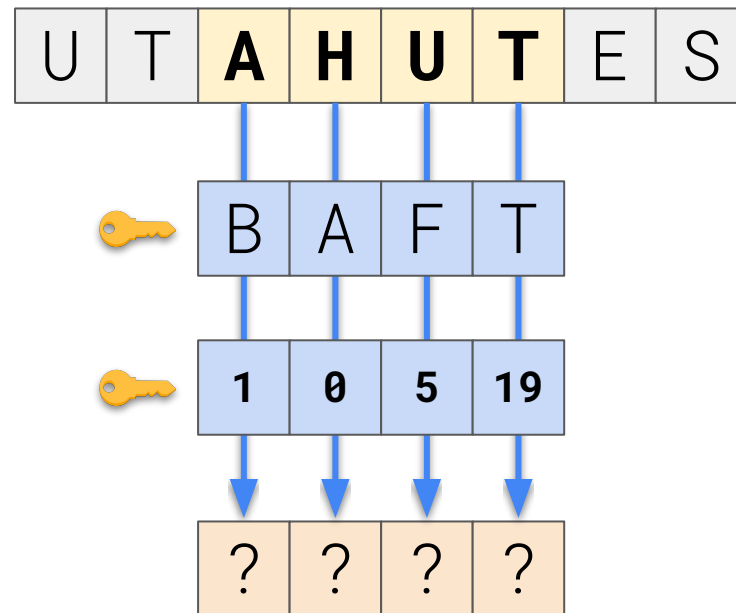
Substitution Ciphers

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



Substitution Ciphers

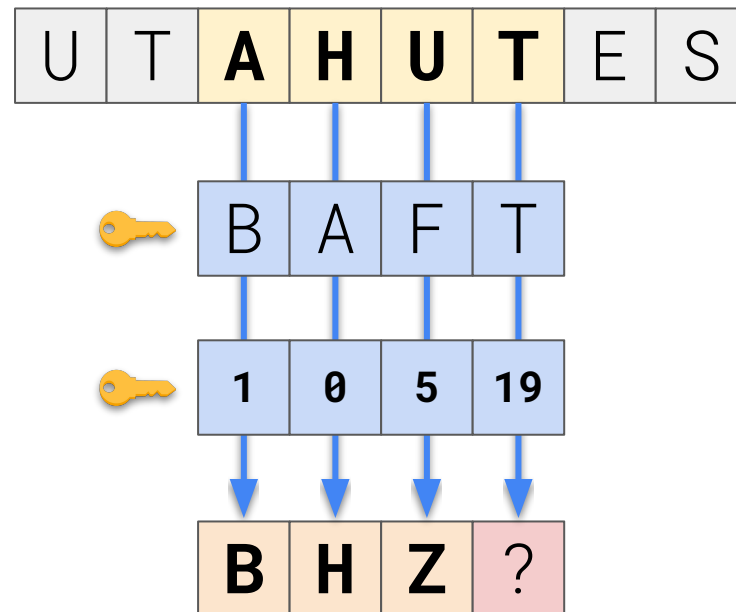
- We define a substitution cipher **key** as a set of **shifts**
- Each shift represented by a **letter**
 - Relative position in the alphabet



Substitution Ciphers

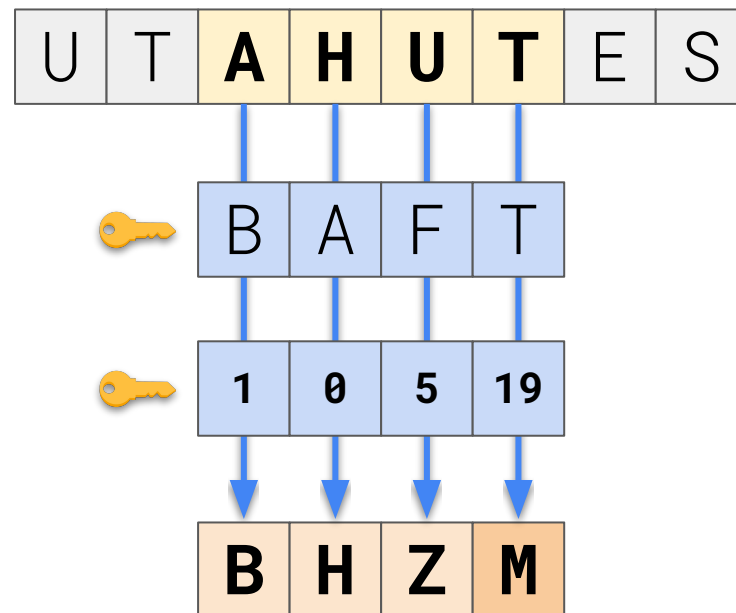
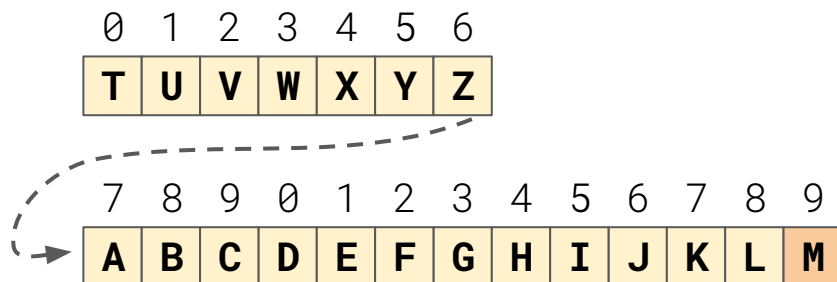
- We define a substitution cipher **key** as a set of **shifts**
- Each shift represented by a **letter**
 - Relative position in the alphabet
- Shift goes past end of alphabet?

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: $c_i := (p_i + k) \bmod 26$
 - Decryption: $p_i := (c_i - k) \bmod 26$



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: **333333333333333333333333333333**
 - =Cipher: **DEFGHIJKLMNOPQRSTUVWXYZABC**

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



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: **333333333333333333333333333333**
 - =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: $p_i = (c_i - k) \bmod 26$
- Example for $k = 3$
 - Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 - +Shift: 3333333333333333333333333333
 - =Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

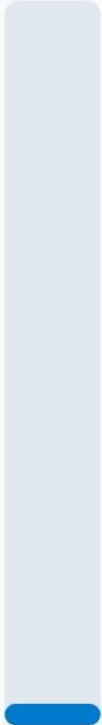
 - Plain: go utes beat wash st
 - +Key: 33 3333 3333 3333 33
 - =Cipher: jr xwhv ehdw zdvk vw

Are Caesar Ciphers **secure**?



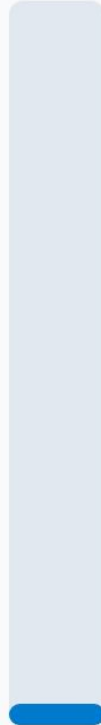
Are Caesar Ciphers secure?

0%



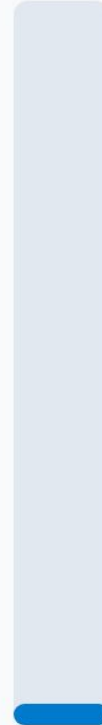
Always!

0%



Sometimes

0%

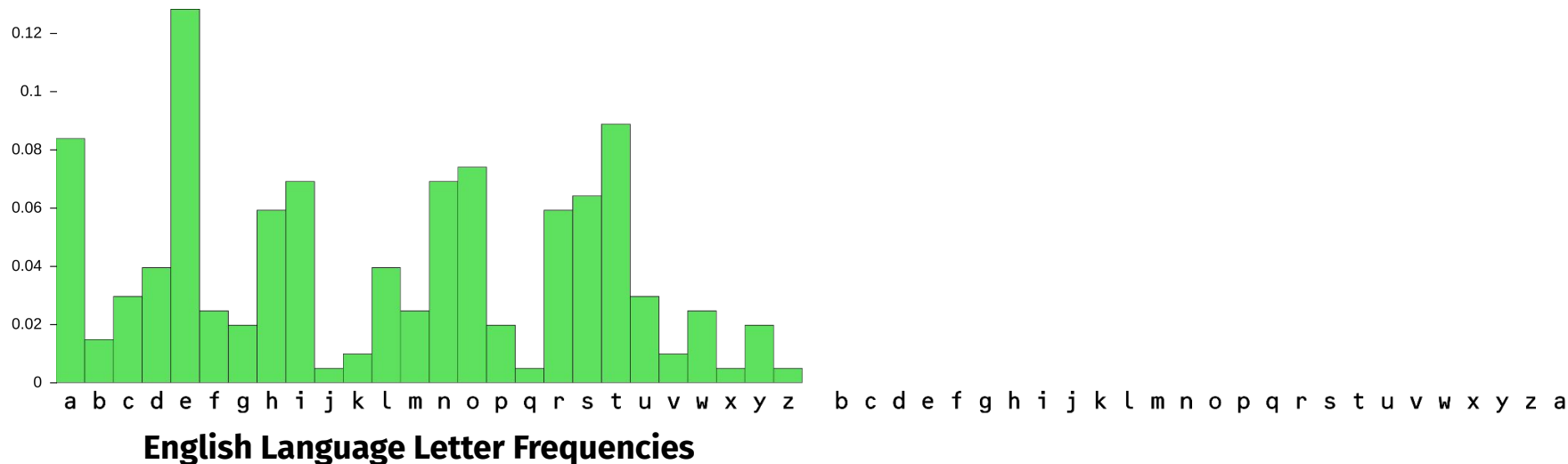


Never :(



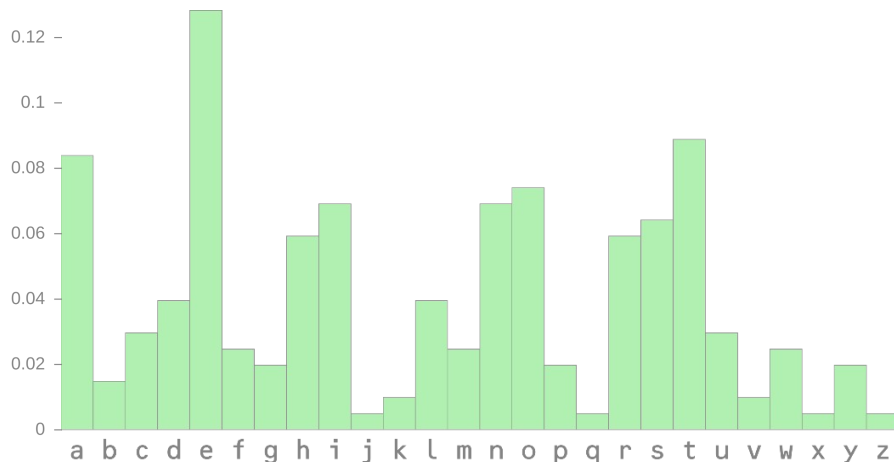
Caesar Cipher Cryptanalysis

- **Observation:** simple substitution ciphers don't alter **symbol frequency**
 - Finding the key: ???

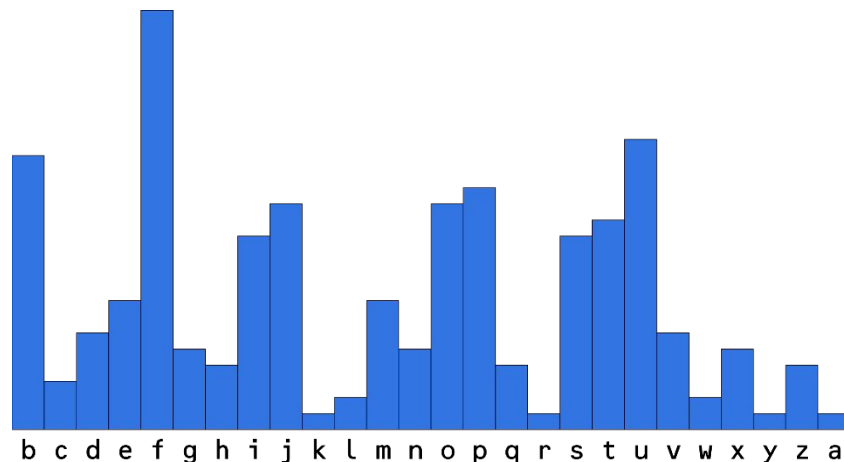


Caesar Cipher Cryptanalysis

- **Observation:** simple substitution ciphers don't alter **symbol frequency**
 - **Finding the key:** map **ciphertext letter frequencies** to their likely **English plaintext letters**



English Language Letter Frequencies



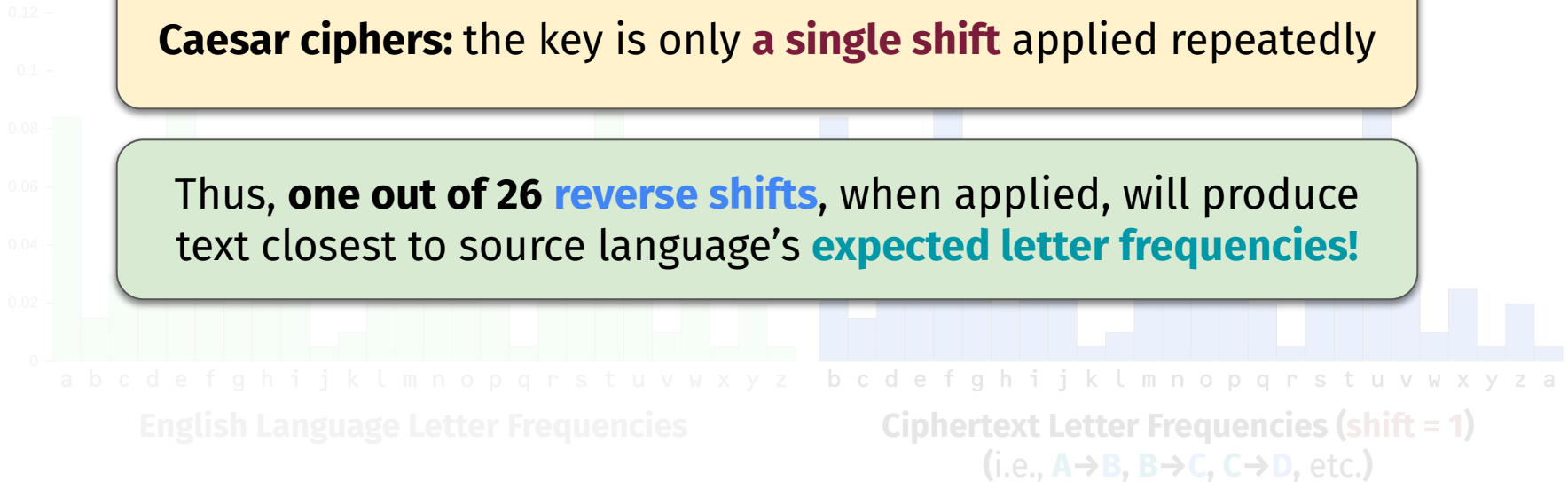
Ciphertext Letter Frequencies (**shift = 1**)
(i.e., **A**→**B**, **B**→**C**, **C**→**D**, etc.)

Caesar Cipher Cryptanalysis

- **Observation:** simple substitution ciphers don't alter **symbol frequency**
 - Finding the key: map **ciphertext letter frequencies** to their likely **English plaintext letters**

Caesar ciphers: the key is only **a single shift** applied repeatedly

Thus, **one out of 26 reverse shifts**, when applied, will produce text closest to source language's **expected letter frequencies!**



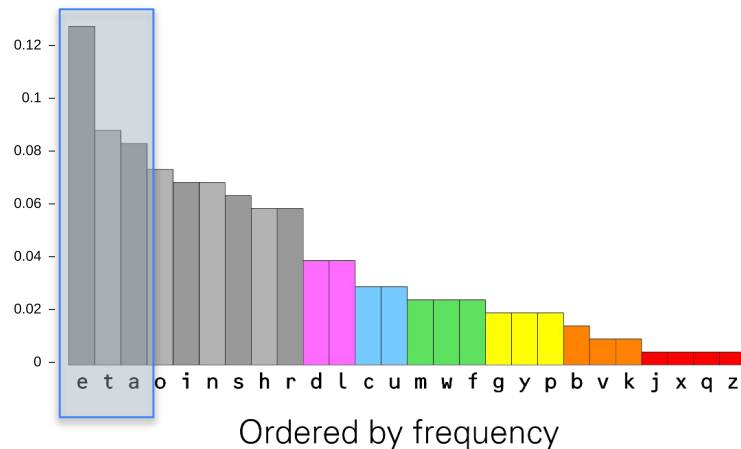
Old School: Caesar Cryptanalysis By-hand

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

Ciphertext: FCWLRMCLWYMCFCSCBCYMYKQJBFCGDACKGMX

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

21% >> 12% → “C” was probably “E”



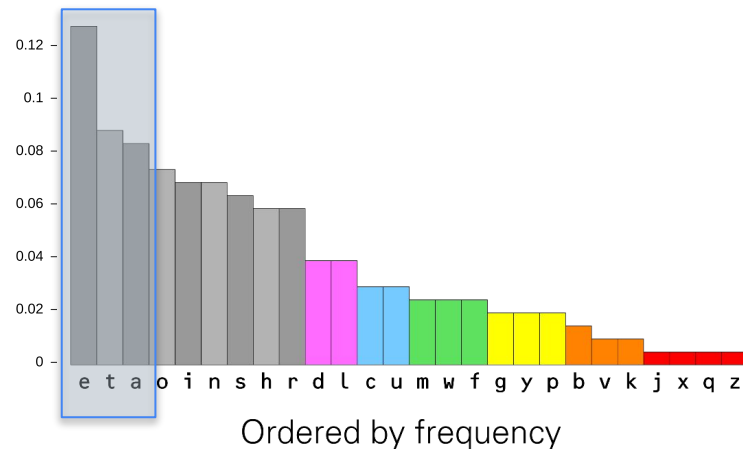
Old School: Caesar Cryptanalysis By-hand

- **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')



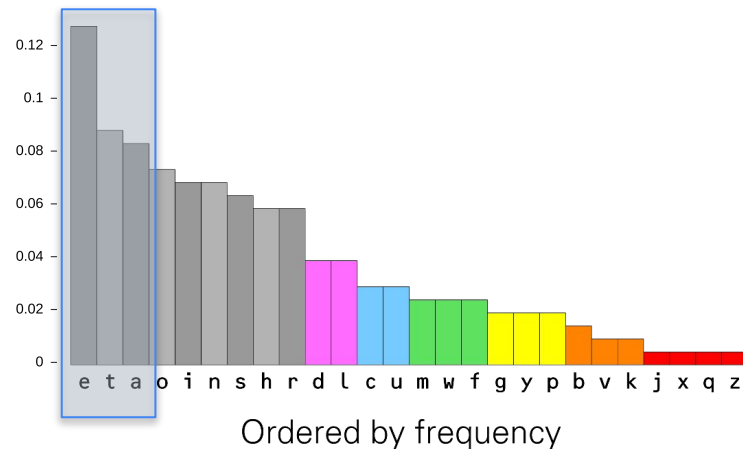
Old School: Caesar Cryptanalysis By-hand

- **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')



Old School: Caesar Cryptanalysis By-hand

■ Narrowing down the search

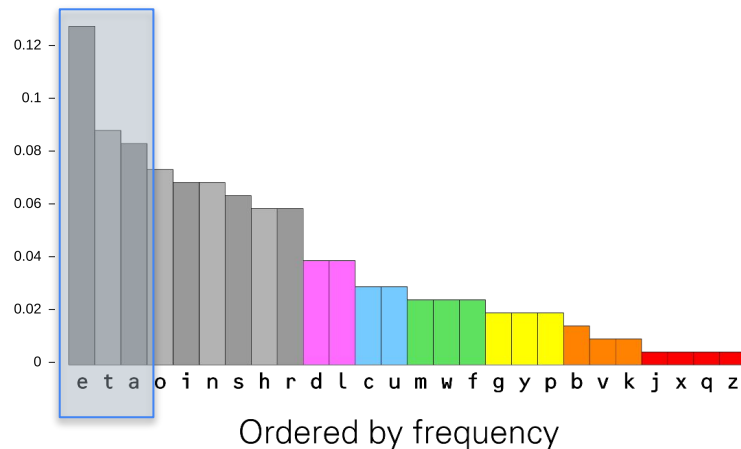
- If a letter is most common by a **large margin**, it's probably a shifted **E**
- Not a large margin? Try to find candidates for shifting **E**, **T**, and **A**

■ Trial and error

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

■ Caveats:

- ???



Old School: Caesar Cryptanalysis By-hand

■ Narrowing down the search

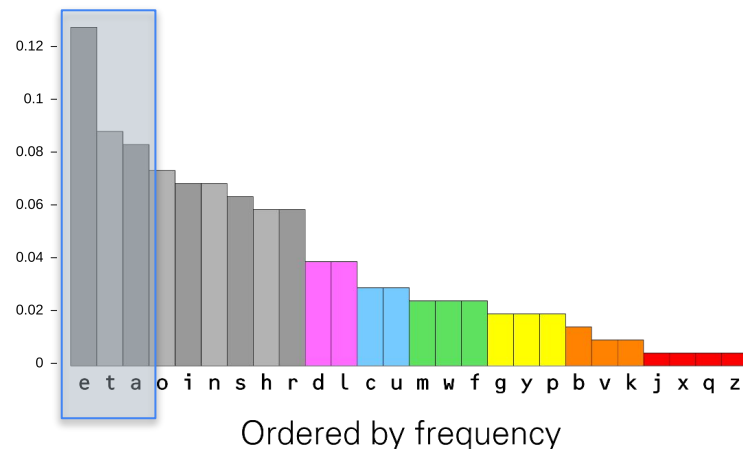
- If a letter is most common by a **large margin**, it's probably a shifted **E**
- Not a large margin? Try to find candidates for shifting **E**, **T**, and **A**

■ Trial and error

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

■ Caveats:

- Must recognize source language's **common words**
- What if you didn't speak English (or Russian, etc.) but **knew the language's letter frequencies**?



Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - First, generate **all 26 possible reverse-shifted strings** from the ciphertext

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU
Shift = 0	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU
Shift = -1	OEKHUWBYIXJUQSXUHQDTEKHWOCJUQSXUQHUUWUJJYDWCQHHYUT
Shift = -2	NDJGTCVAXHWITPRWTGPCSNDJGVNBITPRWTGPGVTIIXCVBPGGXTS
...	
Shift = -25	QGMJWFYDAKZLWSUZWJSFVQGMJYQELWSUZWJSJWYLLAFYESJJAWV

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - Then, calculate **X^2 score** (chi-square) for **each reverse-shifted string**

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU	X^2 Res.
Shift = 0	PF ^L IVEXCZJYKVRTYVIREUPF ^L IXPDKVRTYVIRIVXVKKZEXDRIIZVU	617.449

Example:

O_L = observed count for letter 'L' = 2.0

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - Then, calculate **X² score** (chi-square) for **each reverse-shifted string**

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU	X ² Res.
Shift = 0	PF L IVEXCZJYKVRTYVIREUPF L IXPDKVRTYVIRIVXVKKZEXDRIIZVU	617.449

Example:

O_L = observed count for letter 'L' = **2.0**

E_L = expected count for letter 'L'

$$= \text{EngFreq}_L * | \text{Ciphertext} |$$
$$= 0.04025 * 52 = 2.093$$

$$X^2_L = (2.0 - 2.093)^2 / 2.093 = 0.00413$$

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 }

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - Then, calculate **χ^2 score** (chi-square) for **each reverse-shifted string**

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPKVRTYVIRIVXVKKZEXDRIIZVU	χ^2 Res.
Shift = 0	PF L IVEXCZJYKVRTYVIREUPF L IXPKVRTYVIRIVXVKKZEXDRIIZVU	617.449

Example:

O_L = observed count for letter 'L' = **2.0**

E_L = expected count for letter 'L'

$$= \text{EngFreq}_L * |\text{Ciphertext}|$$

$$= 0.04025 * 52 = 2.093$$

$$\chi^2_L = (2.0 - 2.093)^2 / 2.093 = 0.00413$$

Repeat on remaining 25 letters and sum it up!

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 }
```

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

$$\chi^2_{\text{Shift}=0} = \chi^2_A + \dots + \chi^2_Z$$

$$= 617.449$$

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - Then, calculate **X^2 score** (chi-square) for **each reverse-shifted string**

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU	X^2 Res.
Shift = 0	PF ^L IVEXCZJYKVRTYVIREUPF ^L IXPDKVRTYVIRIVXVKKZEXDRIIZVU	617.449
Shift = -1	OEKHUWBYIXJUQSXUHQD ^T OEKHWOCJUQSXUQH ^U UWUJJYDWCQH ^H YUT	875.797
Shift = -2	NDJGTCVAXHWITPRWTGPCSNDJGVNB ^I T ^P RWTGPGT ^V T ^I I ^X CVBPGG ^X TS	260.953
...		
Shift = -25	QGMJWFYDAKZLWSUZ ^W JSFVQGMJYQELWSUZ ^W JSJWYLLAFY ^E SJJAWV	1094.048

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - Then, calculate **X^2 score** (chi-square) for **each reverse-shifted string**

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU	X^2 Res.	Key Letter
Shift = 0	PF ^L IVEXCZJYKVRTYVIREUPF ^L IXPDKVRTYVIRIVXVKKZEXDRIIZVU	617.449	A
Shift = -1	OEKHUDBWYIXJUQSXUHQD ^T OEKHWOCJUQSXUHQHUWUJJYDWCQH ^H YUT	875.797	B
Shift = -2	NDJGTCVAXHWITPRWTGPCSNDJGVNBITPRWTGPGT ^V TIIXCVBPGGXTS	260.953	C
Shift = -17	??	27.006	R
Shift = -25	QGMJWFYDAKZLWSUZWJSFVQGMJYQELWSUZWJSJWYLLAFYESJJAWV	1094.048	Z

- **Lowest X^2 score = the correct reverse shift...** find its letter and **you've found the key!**

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift **closest to English letter frequencies**
 - Then, calculate **X^2 score** (chi-square) for **each reverse-shifted string**

Ciphertext	PFLIVEXCZJYKVRTYVIREUPFLIXPDKVRTYVIRIVXVKKZEXDRIIZVU	X^2 Res.	Key Letter
Shift = 0	PF ^L IVEXCZJYKVRTYVIREUPF ^L IXPDKVRTYVIRIVXVKKZEXDRIIZVU	617.449	A
Shift = -1	OEKHUDBYIXJUQSXUHQD ^T OEKHWOCJUQSXUHQHUWUJJYDWCQH ^H YUT	875.797	B
Shift = -2	NDJGTCVAXHWITPRWTG ^S NDJGVNBITPRWTGPGT ^V TIIXCVBPGG ^X TS	260.953	C
Shift = -17	YOURENG ^L ISHTEACHERANDYOURGYMTEACHERAREGETTINGMARRIED	27.006	R
Shift = -25	QGMJWFYDAKZLWSUZ ^W JSFVQGMJYQELWSUZ ^W JSJWYLLAFYESJJAWV	1094.048	Z

- **Lowest X^2 score = the correct reverse shift...** find its letter and **you've found the key!**

Better Approach: Statistics-based Cryptanalysis

- **Intuition:** find the reverse-shift closest to English letter frequencies

- Then, calculate X^2 score for each string

Ciphertext			X^2 Res.	Key Letter
Shift = 0	PFLIVEXCZ		617.449	A
Shift = -1	OEKHUDBY		875.797	B
Shift = -2	NDJGTCVAX		260.953	C
Shift = -17	YOURENGLI		27.006	R
Shift = -25	QGMJWFYDA		1094.048	Z

1 ring

0 rings

- Lowest X^2 score = the correct reverse shift... find its letter and you've found the key!

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



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: $c_i := (p_i + k_{i \bmod n}) \bmod 26$
 - Decryption: $p_i := (c_i - k_{i \bmod n}) \bmod 26$
- Example for $k = \text{ABC}$ (i.e., $k_0 = 0, k_1 = 1, k_2 = 2$)
 - Plain: **bbbbbb** amazon
 - +Key: 012012 012012
 - =Cipher: ?????? ??????



Vigènere Ciphers

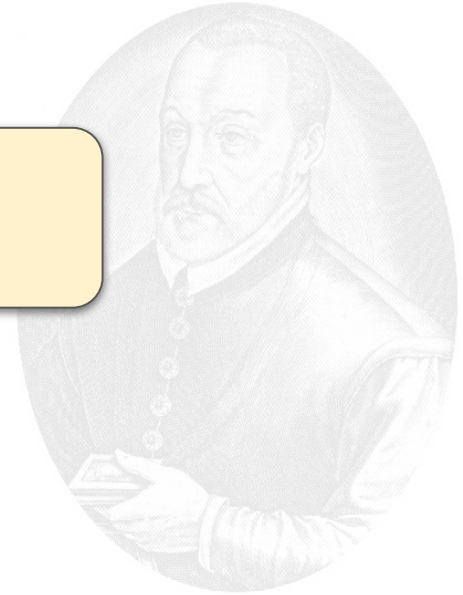
- First described by Bellaso in 1553
 - Later misattributed to Vigènere
- Encrypts successive letters via **sequence of Caesar ciphers** determined by the letters of a keyword
- For an **n**-letter keyword **k** ...
 - Encryption: $c_i := (p_i + k_{i \bmod n}) \bmod 26$
 - Decryption: $p_i := (c_i -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



Vigènere Ciphers

- First described by Bellaso in 1553
 - Later misattributed to Vigènere
- Encrypts successfully, but the ciphers determine the key
- For an n -letter keyword 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 $k = \text{ABC}$ (i.e., $k_0 = 0, k_1 = 1, k_2 = 2$)
 - Plain: **bbbbbb** amazon
 - +Key: 012012 012012
 - =Cipher: bcdbcd anczzp

Can you **brute-force** it?



Vigènere Ciphers

- First described by Bellaso in 1553
 - Later misattributed to Vigènere

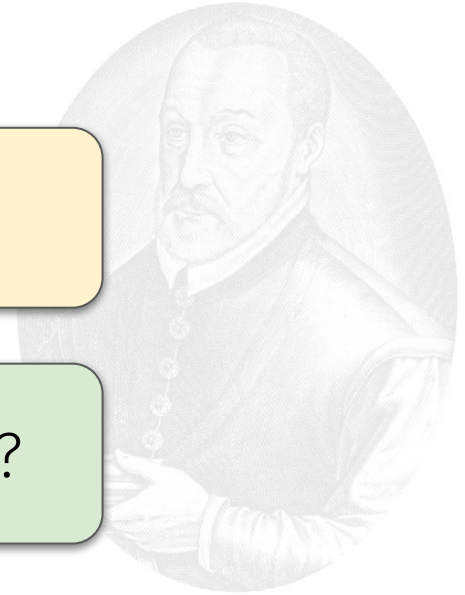
- Encrypts successfully
ciphers determined

Can you **brute-force** it?

- For an n -letter keyword k and a plaintext p
 - Encryption: $c_i := (p_i + k_{i \bmod n}) \bmod 26$
 - Decryption: $p_i := (c_i - k_{i \bmod n}) \bmod 26$

- Example for $k = \text{bcbcd}$
 - Plain: `blue`
 - +Key: `012012 012012`
 - =Cipher: `bcdbcd anczzp`

What about **cryptanalysis**?



Vigènere Cipher Cryptanalysis

- **Observation:** a Vigènere cipher is merely $N=|k|$ Caesar ciphers
 - Break it down into groups of letters—**grouped by column** (i.e., key-shift position)

```
ELFMASBQDXISZMMHIBFEFQIMEUVNGLMLRETHAZAQPDPOTEGDEDONLYVZJNWHCKBLPPQWD  
QZZGFFUKDWCIWPKKSIDYBGBATBUMOWFMYGFBPKYVELFHRHBMDEMESJLQMZVHSXMCPIOW  
KJLQVFGWCBSIOOPAYVDEZJYKORVFGOAYVHMMZJGDAEORWYKQYWUYQOKQEXISZNMNCI  
UINFCEZLJGWHKZEQFBPORMCIVDKFOVEISWJYKVOGMCQAXOELFRKGBPPMTDNGWKEPZUNSLJ  
PHCMPOYUPRXVXYZNIWIAXBZXISXSDPCSPAWFNASSWSDACPPPEWJLRMSJZALDPPCESIS  
XLXSOSUGGMOXPXWUQXSSAZYWBMEFQBSEUHDWNCOTITKXOJFROMYDKQXIEVAOKARSPR  
BGBQEFTTKJOWYIPTPZBSYHQQJSVLXFGKFDPPHVRABCRWBMEFQMGISHDMCBZHFQZTOIEW  
MSXGGAVMCCSAVLPVFRPZOLFFHQKFFQYGFQGPZOUELBHPZOGSEWSPZOECSOULWBAZRBGWSA  
YXNONJSMOEOERYXSBASTGETVGASTGAKCBSIBAKMBZJNCJWIBSIZOCPWCPCGAXOLVPIJV  
DPPJJFOLDPFKSSWDSHPWUVAQRIGINOZWKUTWUOGWKVOQVGPZKDPXISSJYVRFPFKOCSTVFU  
WJNTPWTHDWIJCIBYKQWQLJGXSDDPCSPAPAVMDFFTJOTPEWJYDPPHVRUKTWBTPWBBSI  
NGWQSVREUZASCBTENVKMCMVPAFDPPHVRAEQMEWVDSADPSMTPKOVQYKUSWEKBELFKUKT  
LPMSUSXLEEMYOLYBSINOXGEBSMTJEGVMYXFBYGEVEISKWDDMCWPPYZKSCIBQPKGQELBBCW  
BIYHWSJYOIYGFJCSAXMORKXDMYQSWCSVRSGVEKDQXITSNOLTRWALXIXPFADKBXPXP  
DWSADYFGHGGETXUSZLRMZHPFVYVLPKRFKXGVISOXSDAZWPTWXYXFFEFQKZRWSNKKBTS  
OGDSVNHEZHDJYCLQWLQWQYFVHEKZZZQHHQDWWHZCQSBMZUCBQYCCIMSIWXBMXOHLQZ
```

column1

Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC
- Shift: #####

column2

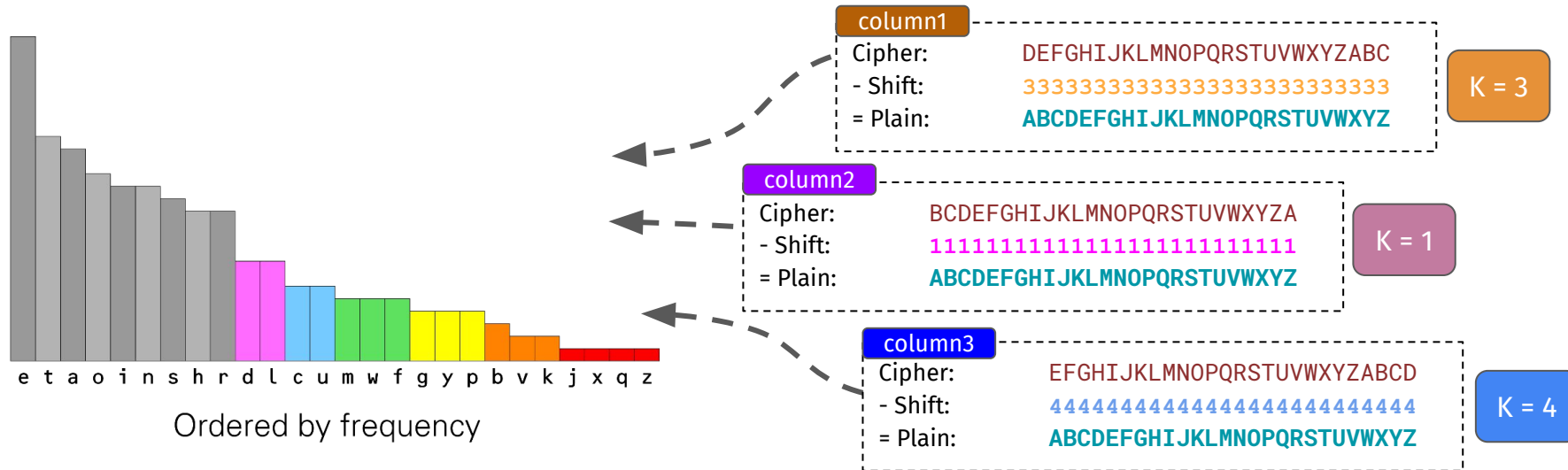
Cipher: BCDEFGHIJKLMNOPQRSTUVWXYZA
- Shift: #####

column3

Cipher: EFGHIJKLMNOPQRSTUVWXYZABCD
- Shift: #####

Vigènere Cipher Cryptanalysis

- **Observation:** a Vigènere cipher is merely $N=|k|$ Caesar ciphers
 - 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**



Vigènere Cipher Cryptanalysis

- How to find **key length**? Use **Kasiski's method**
 - Published 1863 by Kasiski
 - **Observation 1:** in a long plaintext, **repeated strings** often are coincidentally encrypted with **the same key letters**
 - **Observation 2:** when this happens, **distances** between a **repeating cipher string** often are **a multiple of key length**



Vigènere Cipher Cryptanalysis

- How to find **key length**? Use **Kasiski's method**
 - Published 1863 by Kasiski
 - **Observation 1:** in a long plaintext, **repeated strings** often are coincidentally encrypted with **the same key letters**
 - **Observation 2:** when this happens, **distances** between a **repeating cipher string** often are **a multiple of key length**

- **Example:**

- Plain: **CRYPTO**ISSHORTFOR**CRYPTO**GRAPHY
- +Key: **ABCD**ABCDABCDABCD**ABCD**ABCDABCD
- =Cipher: **CSASTP**KVSIQUTGQU**CSASTP**IUAQJB

Distance between **CSASTP** = **16**



Possible key lengths
= **16, 8, 4, 2, or 1**

Finding the Key Length: Kasiski's 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

Finding the Key Length: Kasiski's Method

■ Let's visualize a larger 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

Finding the Key Length: Kasiski's Method

■ Let's visualize a larger example:

p	THERE ARE	TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
key	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	
key	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	
key	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
key	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD

Finding the Key Length: Kasiski's Method

■ Let's visualize a larger example:

p	THERE ARE	TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
key	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
key	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
key	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
key	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD





Finding the Key Length: Kasiski's Method

■ Let's visualize a larger example:

p	THERE ARE	TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	AREDE	SIGNO	NEWAY
key	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
key	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
key	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
key	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD

Finding the Key Length: Kasiski's Method

■ Let's visualize a larger 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	EM SY	TEMSY	STEMS	YSTEM	SYSTE	MSYST	
c	ALXAE	YCXMF	KMKBQ	BDCLA	E LFW	KIMJC	GUZUG	SKECZ	GBWYM	OACFV	
p	IESAN	DTH EO	THERW	AYIST	OMAKE	ITSOC	OMPLI	CATED	THATT	HEREA	
	EMSYS	TEMS Y	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	
c	MQKYF	WXT WM	LAIDO	YQBWF	GKSDI	ULQGV	SYHJA	VEFWB	LAEFL	FWKIM	
p	RENOO	BVIOU	SDEFI	CIENC	IESTH	EFIRS	TM ETH	ODISF	ARMOR	EDIFF	
	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SY STE	MSYST	EMSYS	TEMSY	
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LK WML	AVGKY	EDEMJ	XHUXD	

Finding the Key Length: Kasiski's Method

■ Let's visualize a larger example:

p	THERE ARE	TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	ARE	DE	SIGNO	NEWAY
key	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	
key	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	
key	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	
key	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY	
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD	

Finding the Key Length: Kasiski's Method

■ Let's visualize a larger example:

p	THERE	ARE	TW	OWAYS	OFCON	STRUC	TINGA	SOFTW	ARE	DE	SIGNO	NEWAY
key	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY		
c	LFWKI	MJCLP	SISWK	HJOGL	KMVGU	RAGKM	KMXMA	MJCVX	WUYLG	GI	ISW	
p	ISTOM	AKEIT	SOSIM	PLETH	ATTHE	REARE	OBVIO	USLYN	ODEFI	CIENC		
key	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		
key	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		
key	SYSTE	MSYST	EMSYS	TEMSY	STEMS	YSTEM	SYSTE	MSYST	EMSYS	TEMSY		
c	JCFHS	NNGGN	WPWDA	VMQFA	AXWFZ	CXBVE	LKWML	AVGKY	EDEMJ	XHUXD		

Finding the Key Length: Kasiski's Method

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

Substring	Length	Starting Positions within Ciphertext
LWFKIMJC	8	
WMLA	4	
MJC	3	
ISW	3	
KMK	3	
VMQ	3	

Finding the Key Length: Kasiski's Method

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

Substring	Length	Starting Positions within Ciphertext	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)	

Finding the Key Length: Kasiski's Method

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

Substring	Length	Starting Positions within Ciphertext	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

Finding the Key Length: Kasiski's Method

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

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

Finding the Key Length: Kasiski's Method

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

Substring	Length	Factors of Distances	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

Finding the Key Length: Kasiski's Method

- To visualize outliers, make a table of **occurrences of distance factors**

	Factors of Distances and their Occurrences																		
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 the Key Length: Kasiski's Method

- Cull **outlier** distance factors...

	Factors of Distances and their Occurrences																		
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 the Key Length: Kasiski's Method

- Cull **outlier** distance factors... as well as **unrealistically small** ones

	Factors of Distances and their Occurrences																		
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 the Key Length: Kasiski's Method

- Cull **outlier** distance factors... as well as **unrealistically small** ones

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

Finding the Key Length: Kasiski's Method

- Compute the **greatest common factor** of remaining 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

Finding the Key Length: Kasiski's Method

- Compute the **greatest common factor** of remaining 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

Our **key length** is likely **6**!

After Key Length Found: Solving the Cipher

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

123456	123456	123456	123456	123456	123456	123456	123456	123456
LFWKIM	JCLPSI	SWKHJO	GLKMVG	URAGKM	KMXMAM	JCVXWU	YLGII	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		

After Key Length Found: Solving the Cipher

- Then, **group letters by columns**—if key length right, they received equal shifts!

123456	123456	123456	123456	123456	123456	123456	123456	123456
LFWKIM	JCLPSI	SWKHJO	GLKMVG	URAGKM	KMXMAM	JCVXWU	YLGII	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		

After Key Length Found: Solving the Cipher

- Then, **group letters by columns**—if key length right, they received equal shifts!

123456	123456	123456	123456	123456	123456	123456	123456	123456
LFWKIM	JCLPSI	SWKHJO	GLKMVG	URAGKM	KMXMAM	JCVXWU	YLGII	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		

After Key Length Found: Solving the Cipher

- Then, **group letters by columns**—if key length right, they received equal shifts!

123456	123456	123456	123456	123456	123456	123456	123456	123456
LFWKIM	JCLPSI	SWKHJO	GLKMVG	URAGKM	KMXMAM	JCVXWU	YLGII	SWALXA
123456	123456	123456	123456	123456	123456	123456	123456	123456
EYCMF	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		

After Key Length Found: Solving the Cipher

■ Then, g

Observation 1: we've broken a Vigenere Cipher down into **$N=|k|$ Caesar Ciphers**

equal shifts!

Observation 2: for a large plaintext, (e.g., a book), **slices of it** will uphold **its language's letter frequencies**

Now, use **cryptanalysis on all N slices** to recover the **N -length Vigenere key!**

Questions?



Next time on CS 4440...

One-time Pads, Transposition and Block Ciphers