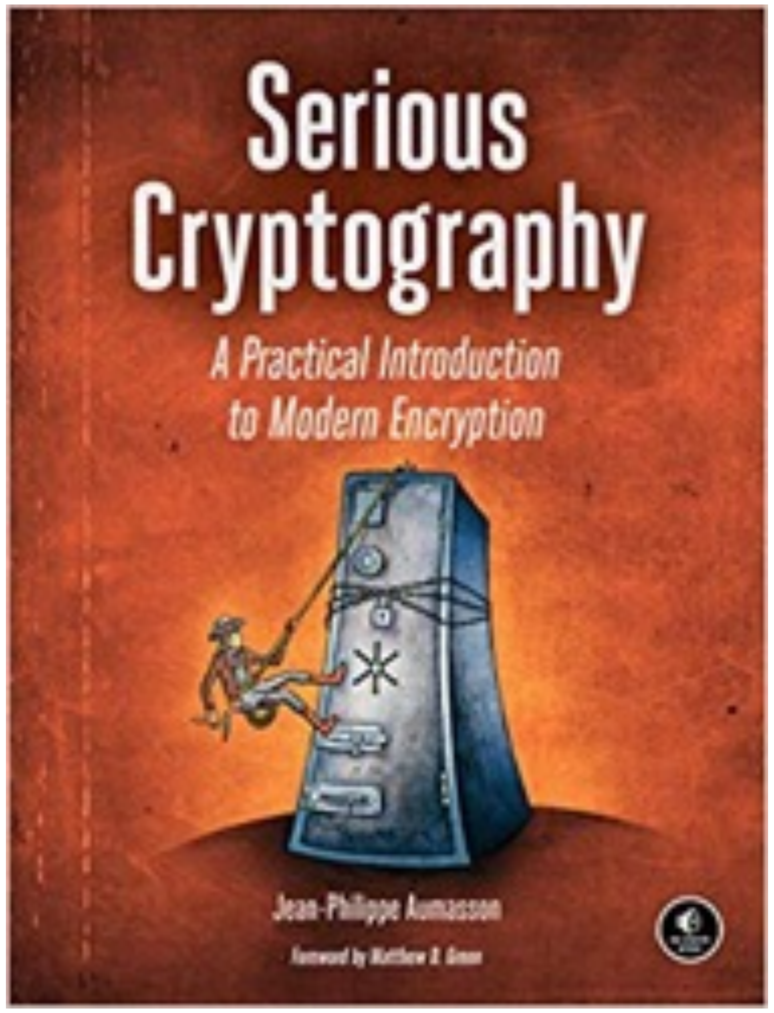
CNIT 141
Cryptography for Computer Networks


Serious Cryptography
A Practical Introduction to Modern Encryption
Jean-Philippe Aumasson
Foreword by Matthew D. Green

1. Encryption

Updated on 8-25-2021

Topics

- The Basics
- Classical Ciphers
- How Ciphers Work
- Perfect Encryption: The One-Time Pad
- Encryption Security
- Asymmetric Encryption
- When Ciphers Do More Than Encryption
- How Things Can Go Wrong

The Basics

- P: Plaintext
- K: Key
- C: Ciphertext
- E: Encryption via cipher
- D: Decryption via cipher

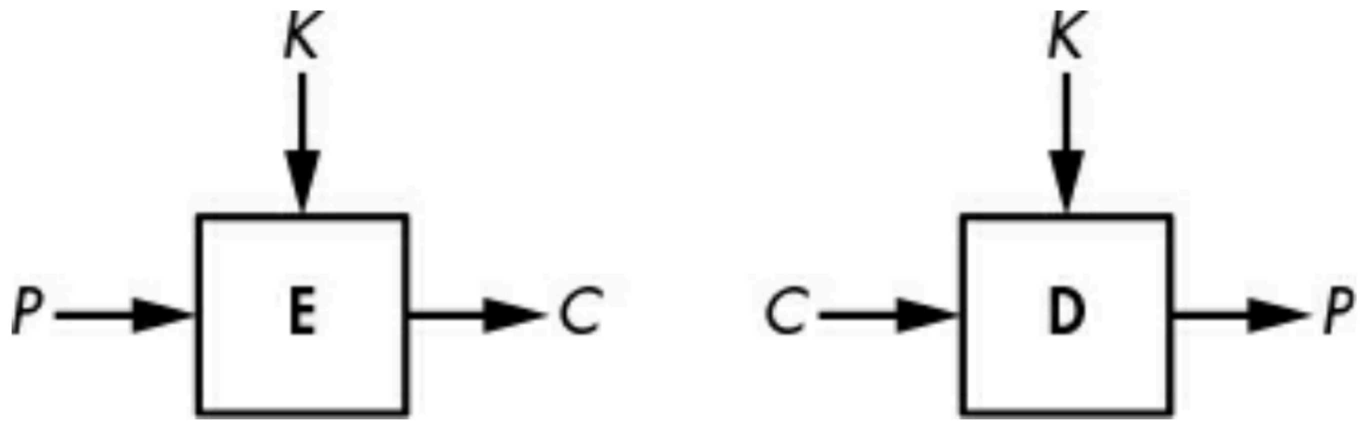


Figure 1-1: Basic encryption and decryption

Classical Ciphers

Caesar Cipher

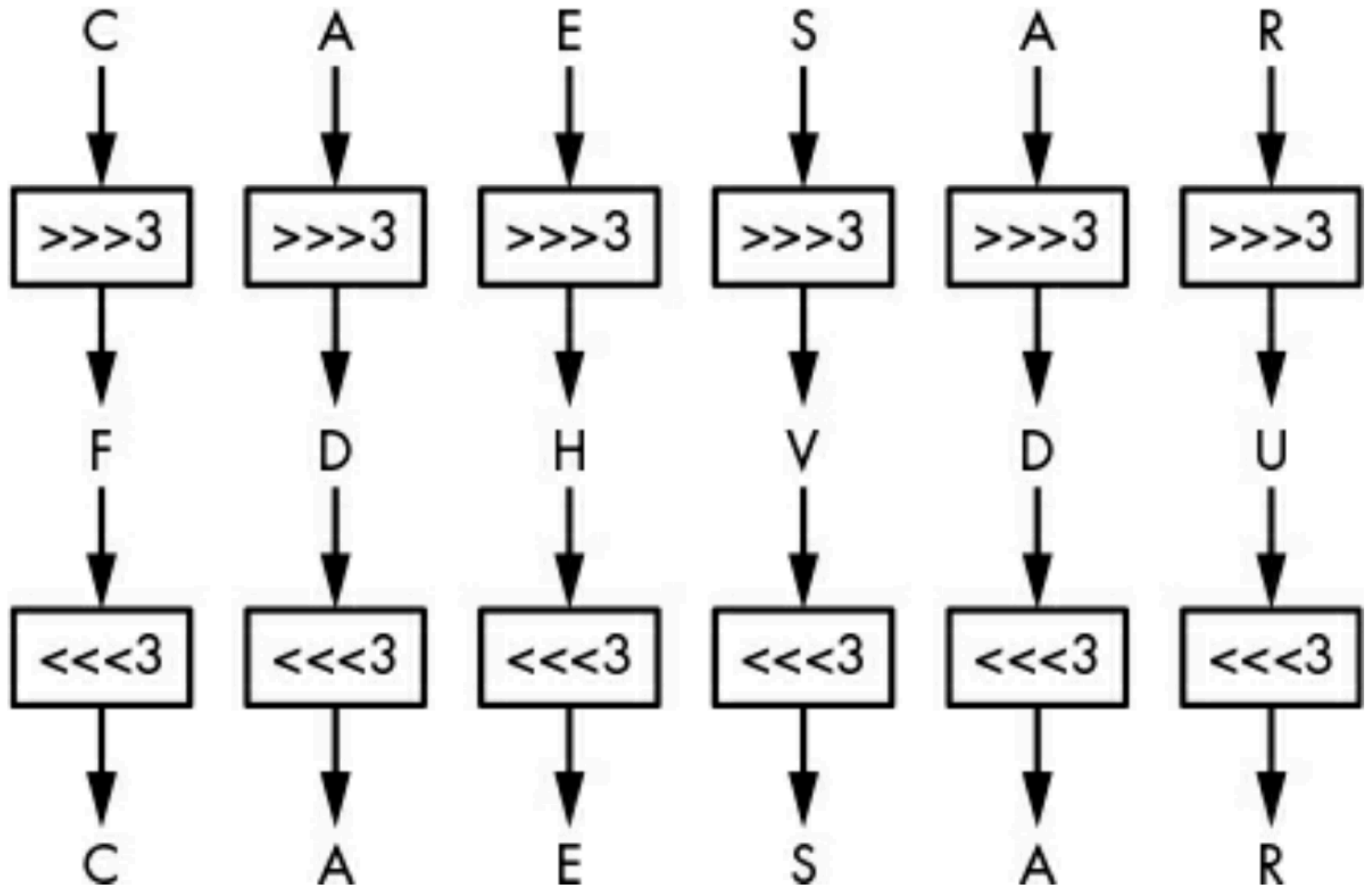
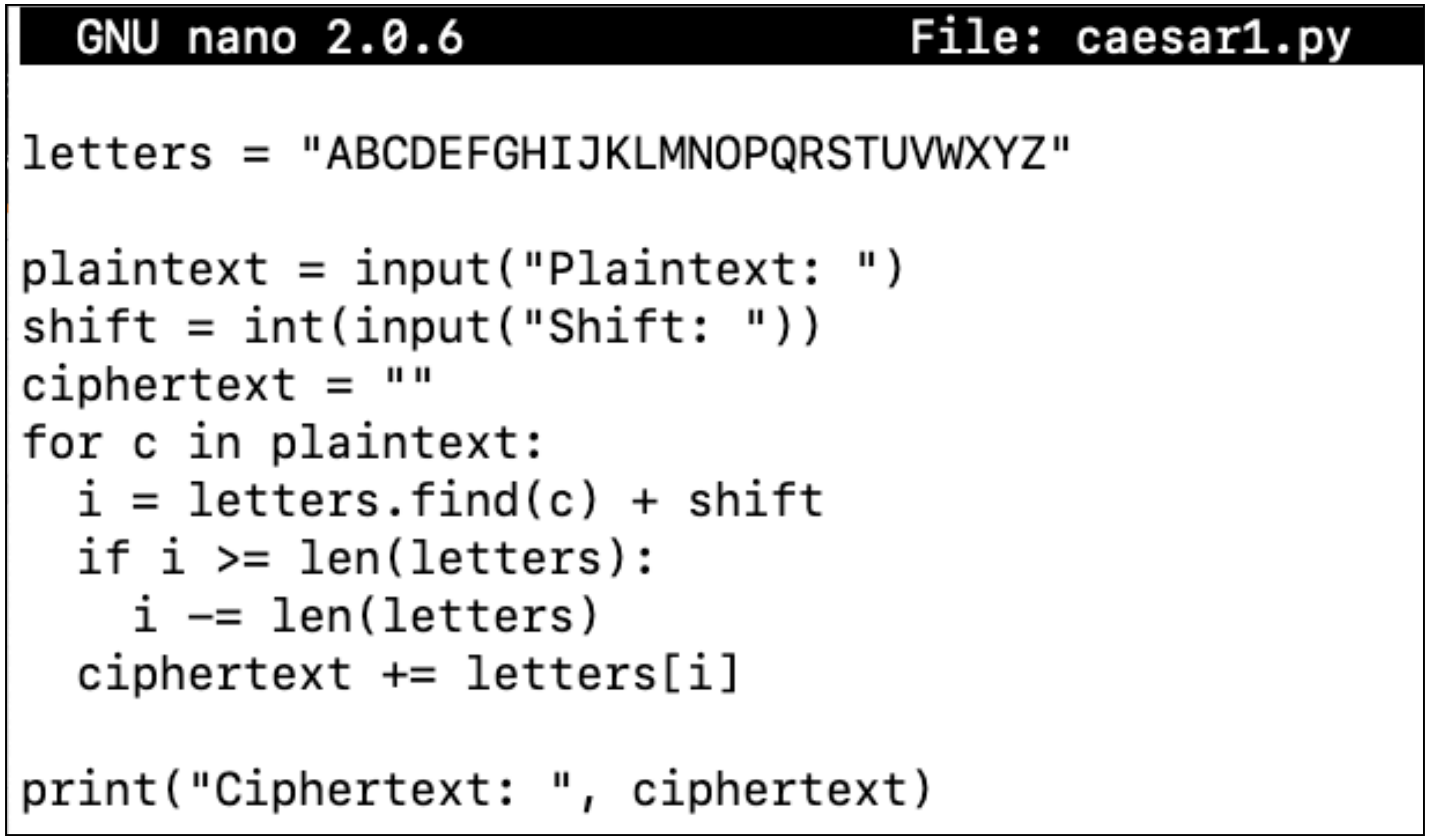
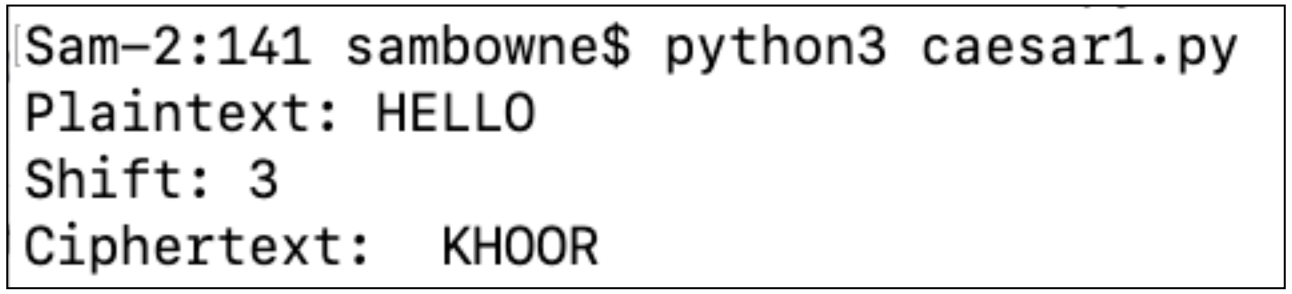


Figure 1-2: The Caesar cipher

Caesar Cipher in Python

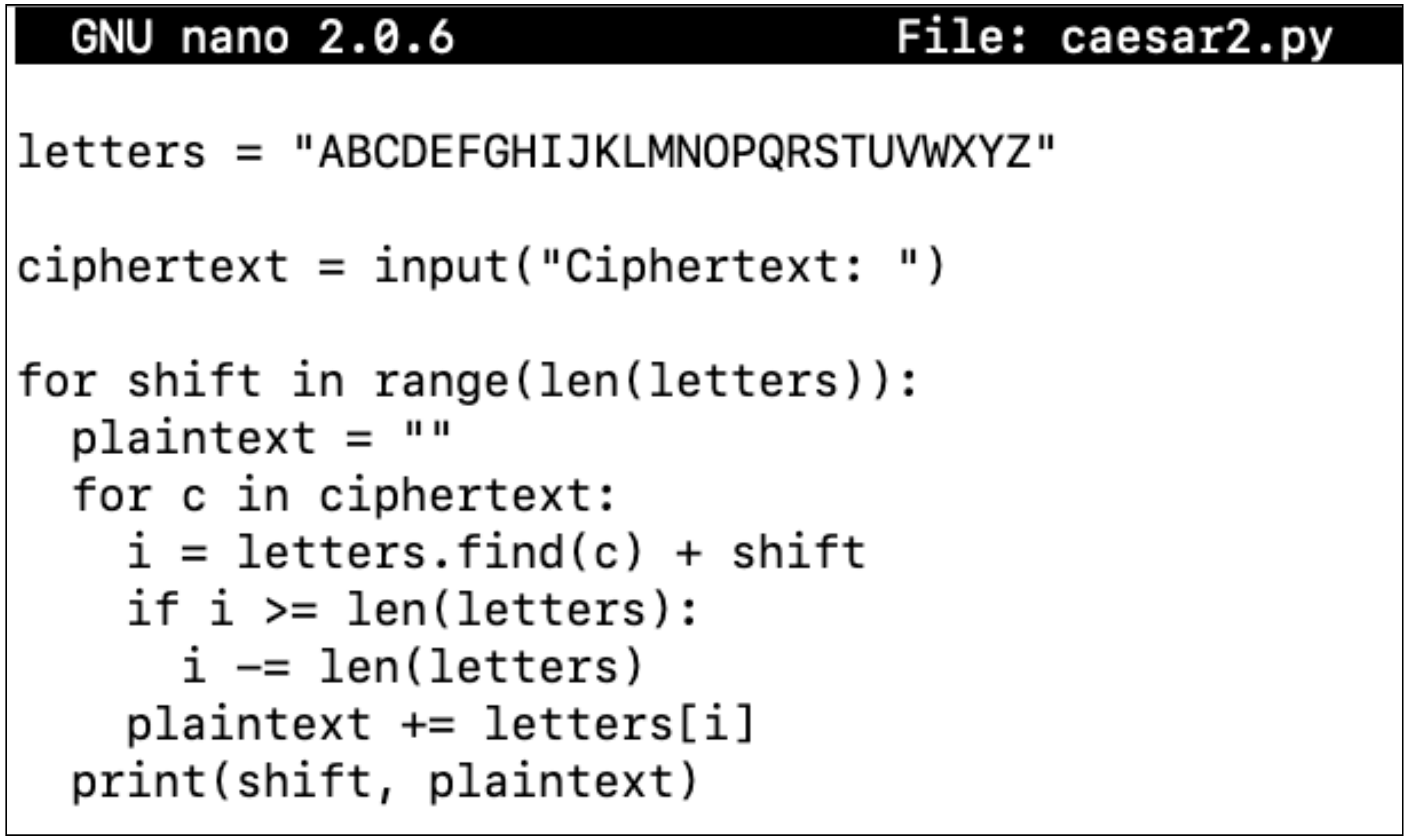


```
  GNU nano 2.0.6                      File: caesar1.py

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

plaintext = input("Plaintext: ")
shift = int(input("Shift: "))
ciphertext = ""
for c in plaintext:
  i = letters.find(c) + shift
  if i >= len(letters):
    i -= len(letters)
  ciphertext += letters[i]

print("Ciphertext: ", ciphertext)
```

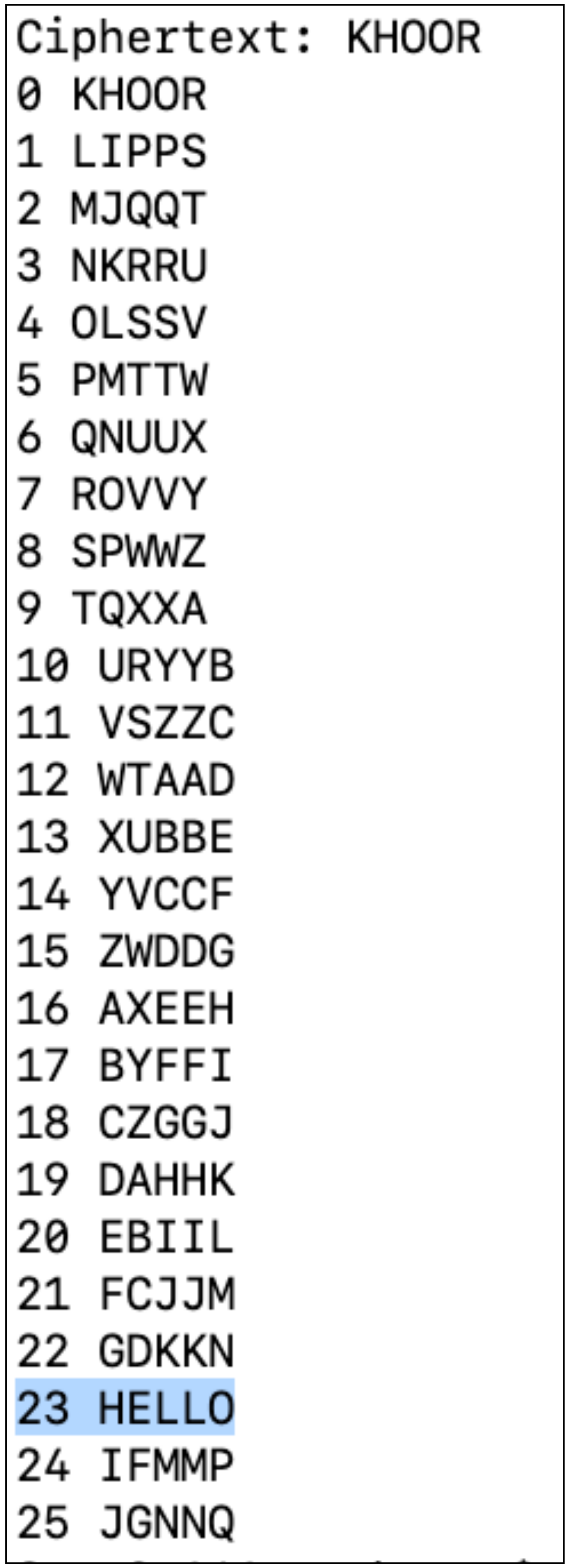


```
Sam-2:141 sambowne$ python3 caesar1.py
Plaintext: HELLO
Shift: 3
Ciphertext:  KHOOR
```

Brute Force Attack



```
  GNU nano 2.0.6                File: caesar2.py

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

ciphertext = input("Ciphertext: ")

for shift in range(len(letters)):
  plaintext = ""
  for c in ciphertext:
    i = letters.find(c) + shift
    if i >= len(letters):
      i -= len(letters)
    plaintext += letters[i]
  print(shift, plaintext)
```



```
Ciphertext: KHOOR
0 KHOOR
1 LIPPS
2 MJQQT
3 NKRRU
4 OLSSV
5 PMTTW
6 QNUUX
7 ROVVY
8 SPWWZ
9 TQXXA
10 URYYB
11 VSZZC
12 WTAAD
13 XUBBE
14 YVCCF
15 ZWDDG
16 AXEEH
17 BYFFI
18 CZGGJ
19 DAHHK
20 EBIIL
21 FCJJM
22 GDKKN
23 HELLO
24 IFMMP
25 JGNNQ
```

Vigenere Cipher

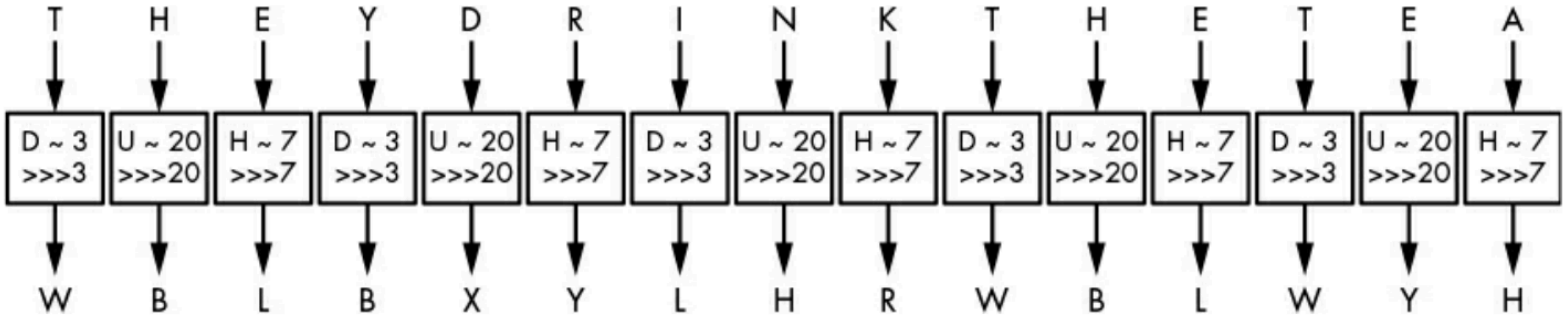


Figure 1-3: The Vigenère cipher

- Shift varies with a repeated keyword
- Combine several Caesar ciphers together

Breaking the Vigenere Cipher

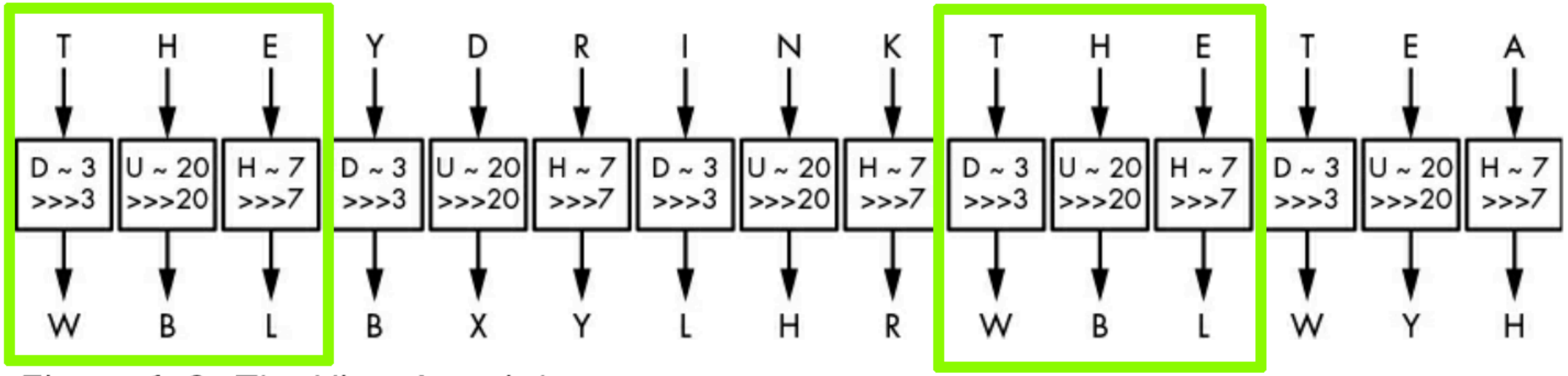
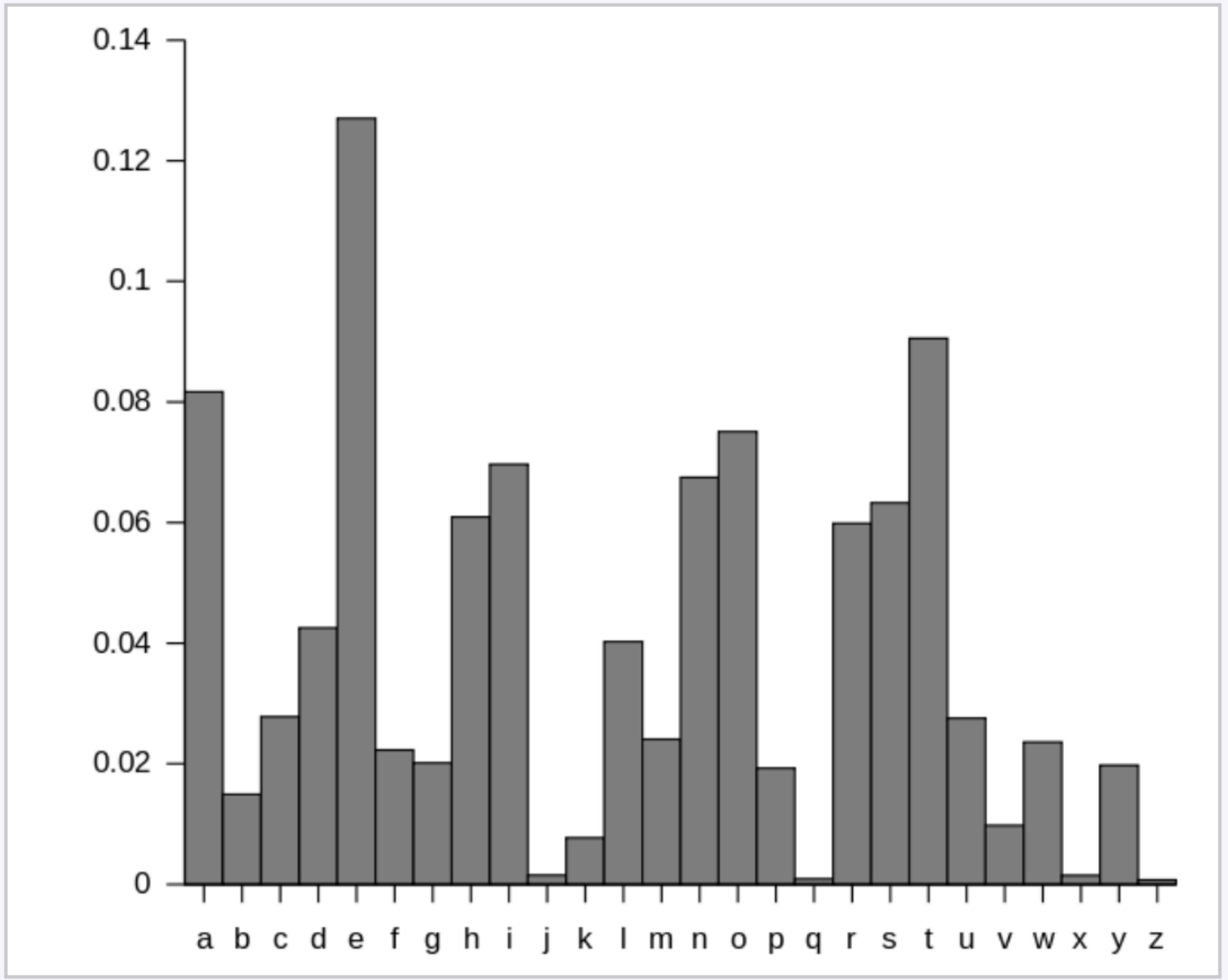


Figure 1-3: The Vigenère cipher

- Find repeating ciphertext to deduce key length
- Use ***frequency analysis***

Frequency Analysis

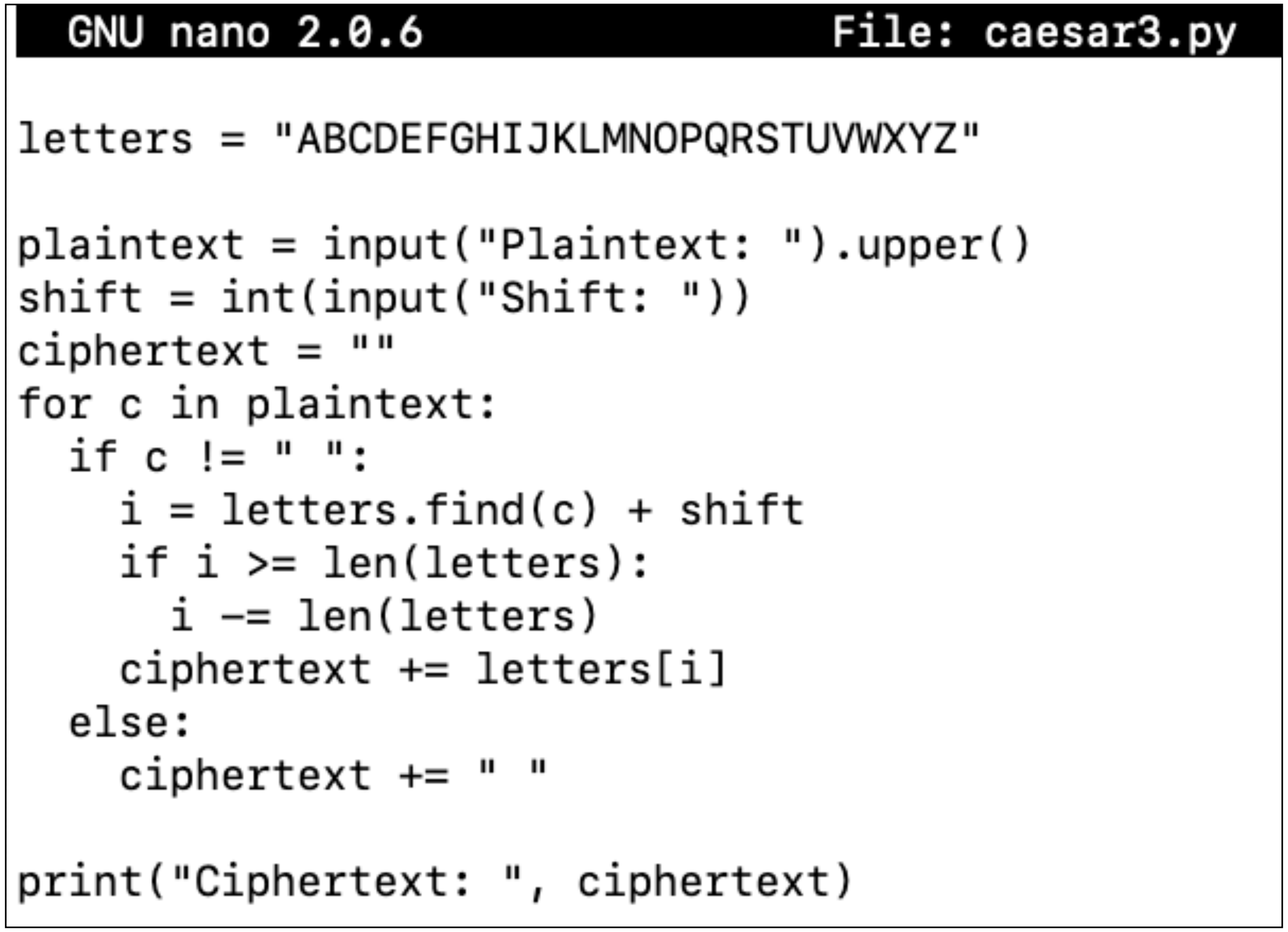


A typical distribution of letters in English language text.

- From Wikipedia

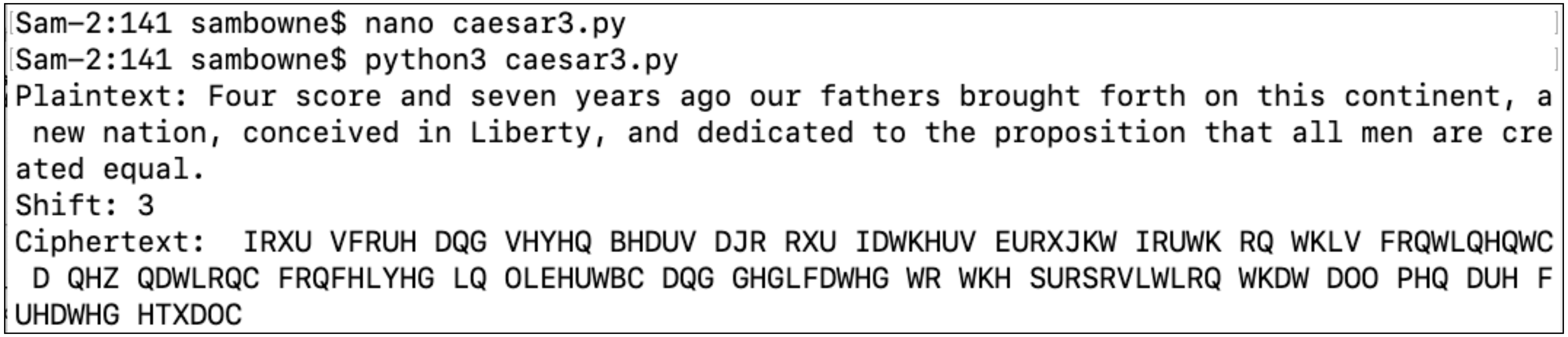
Modified Caesar Program

- Converts to uppercase
- Preserves spaces



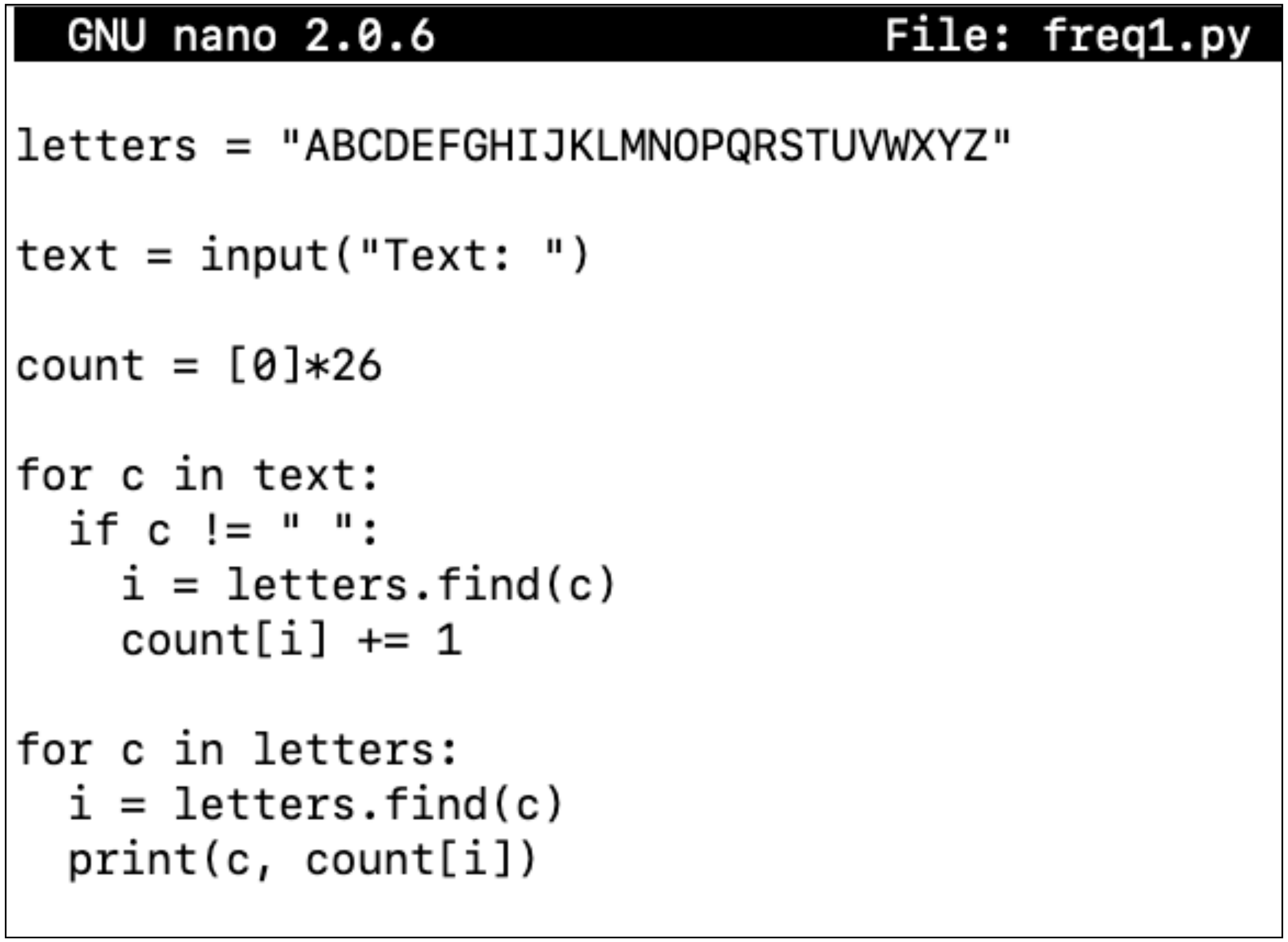
```
  GNU nano 2.0.6                     File: caesar3.py

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

plaintext = input("Plaintext: ").upper()
shift = int(input("Shift: "))
ciphertext = ""
for c in plaintext:
  if c != " ":
    i = letters.find(c) + shift
    if i >= len(letters):
      i -= len(letters)
    ciphertext += letters[i]
  else:
    ciphertext += " "

print("Ciphertext: ", ciphertext)
```

Encrypt a Paragraph

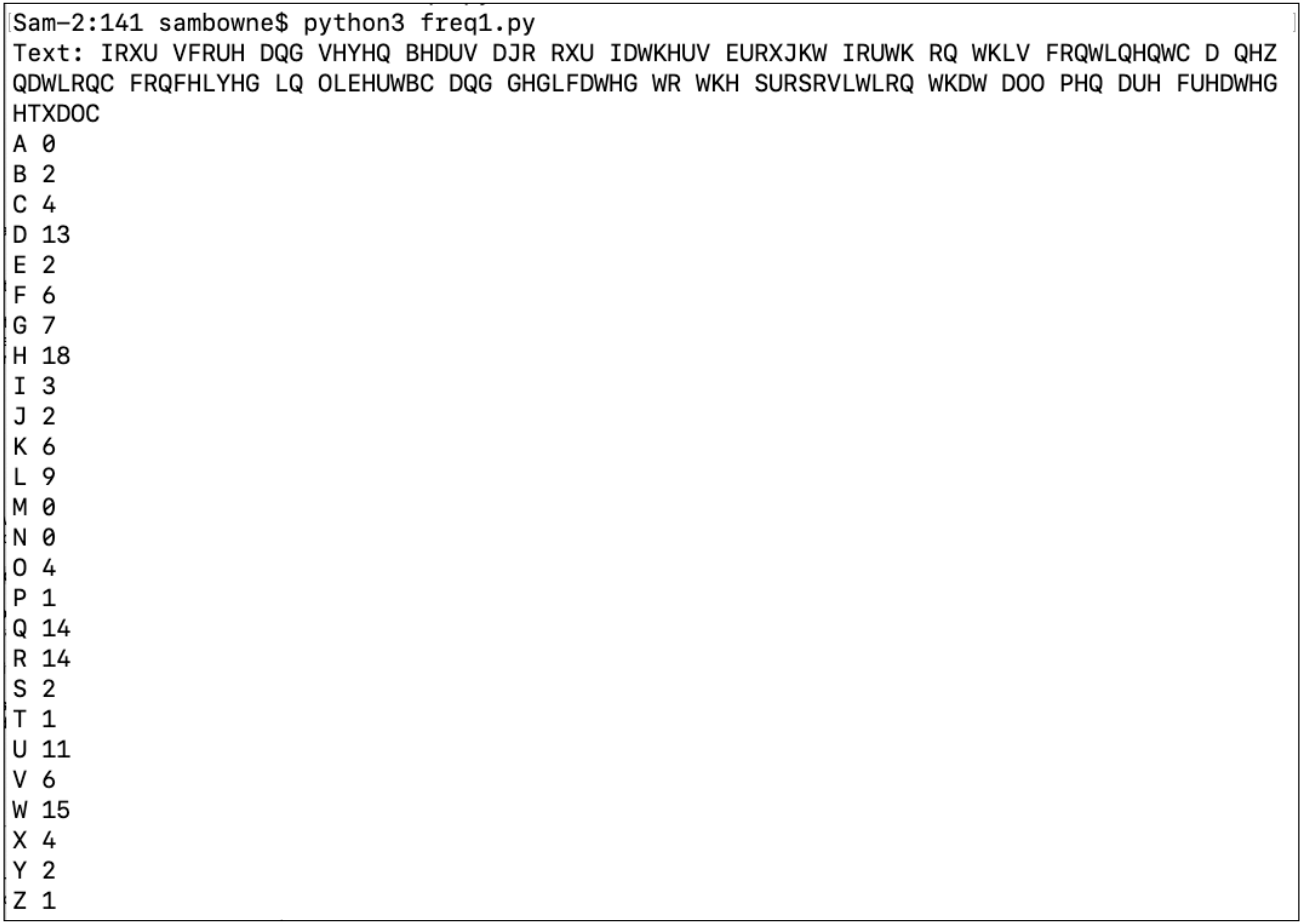


```
Sam-2:141 sambowne$ nano caesar3.py
Sam-2:141 sambowne$ python3 caesar3.py
Plaintext: Four score and seven years ago our fathers brought forth on this continent, a
 new nation, conceived in Liberty, and dedicated to the proposition that all men are cre
ated equal.
Shift: 3
Ciphertext:  IRXU VFRUH DQG VHYHQ BHDUV DJR RXU IDWKHUV EURXJKW IRUWK RQ WKLV FRQWLQHQWC
 D QHZ QDWLRQC FRQFHLYHG LQ OLEHUWBC DQG GHGLFDWHG WR WKH SURSRVLWLRQ WKDW DOO PHQ DUH F
UHDWHG HTXDOC
```

Frequency Counter



```
  GNU nano 2.0.6                      File: freq1.py

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

text = input("Text: ")

count = [0]*26

for c in text:
  if c != " ":
    i = letters.find(c)
    count[i] += 1

for c in letters:
  i = letters.find(c)
  print(c, count[i])
```



```
Sam-2:141 sambowne$ python3 freq1.py
Text: IRXU VFRUH DQG VHYHQ BHDUV DJR RXU IDWKHUV EURXJKW IRUWK RQ WKLV FRQWLQHQWC D QHZ
QDWLRQC FRQFHLYHG LQ OLEHUWBC DQG GHGLFDWHG WR WKH SURSRVLWLRQ WKDW DOO PHQ DUH FUHDWHG
HTXDOC
A 0
B 2
C 4
D 13
E 2
F 6
G 7
H 18
I 3
J 2
K 6
L 9
M 0
N 0
O 4
P 1
Q 14
R 14
S 2
T 1
U 11
V 6
W 15
X 4
Y 2
Z 1
```

How Ciphers Work

Two Components

- **Permutation**
  - Transforms one letter to another letter
  - In Caesar cipher, shift letter three places
- **Mode of Operation**
  - Algorithm to handle messages of arbitrary size
  - In Caesar cipher, process each letter independently

Permutation Security

- **Permutation should be determined by the key**
  - If key is secret, attacker can’t easily decrypt
- **Different keys should result in different permutations**
- **Permutation should look random**
  - No pattern in ciphertext

Mode of Operation

- Caesar cipher encrypts letters one at a time
  - Double letters remain doubled
  - HE**LL**O -> KH**OO**R
- Patterns in plaintext are preserved in ciphertext
- Insecure (now called “Electronic Code Book” mode)
- More secure modes encrypt repeated text differently each time

Perfect Encryption: The One-Time Pad

XOR

- XOR combines two bits
    - 0 ^ 0 = 0
    - 0 ^ 1 = 1
    - 1 ^ 0 = 1
    - 1 ^ 1 = 0

Encrypting a Stream of Bits

- Plain: ABC = `0100 0001 0100 0010 0100 0011`
- Key: `0110 0110 0110 0101 1010 1110`
- Cipher: `0010 0111 0010 0111 1110 1101`

- Key must be random and never re-used
- Key must be longer than all the plaintexts you want to send

Unbreakable

- If an attacker uses a brute-force attack
- Trying all possible keys
- They get all possible letter sequences
- No way to identify the correct decryption

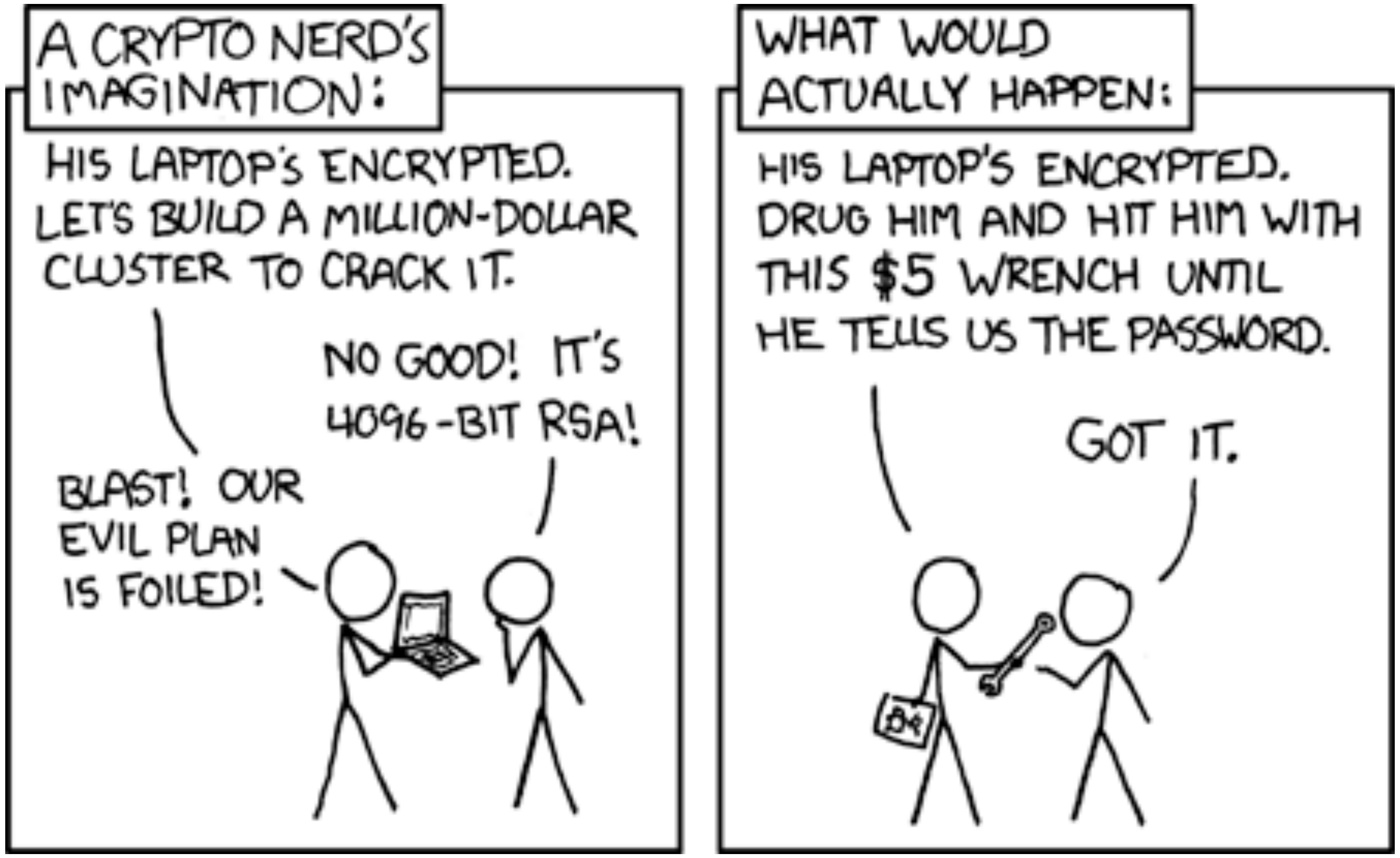
Kahoot!

Encryption Security

Attack Models

- Set requirements for cryptographers who design ciphers
  - So they know what attacks to prevent
- Give guidelines to users
  - Whether a cipher is safe in their environment
- Provide clues for cryptanalysts who attempt to break ciphers
  - Is an attack doable in the model considered?

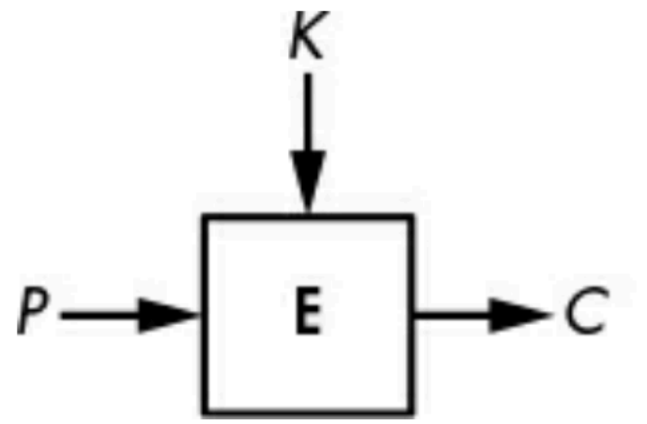
Attack Models


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Kerckhoff’s Principle

- The **key** is secret
- The **cipher** is not secret

Black-Box Models



No knowledge of cipher operation

- Ciphertext-Only Attack (COA)
  - Attacker sees only C
- Known-Plaintext Attack (KPA)
  - Attacker knows P and C
- Chosen-Plaintext Attack (CPA)
  - Attacker can perform encryption for any P
- Chosen-Ciphertext Attack (CCA)
  - Attacker can perform encryption and decryption

Gray-Box Models

- Attacker has access to the implementation
  - Can tamper with the system’s internals
- **Side-channel attack**
  - Attacker measures something else about the cipher’s operation
  - Such as timing or power consumption
  - **Noninvasive** — does not alter integrity of system

Gray-Box Models

- **Invasive attacks**
  - Modify system
  - Examples
    - Using acid to dissolve parts of a microchip
    - Injecting faults with lasers

Security Goals

- **Indistinguishability**
  - Ciphertext should be indistinguishable from a random string
- **Non-malleability**
  - Ciphertext cannot be altered and produce meaningful plaintext

Security Notions

- **IND-CPA**
  - Indistinguishability against a Chosen-Plaintext Attack
  - Also called **semantic security**
  - Two identical plaintext strings must result in different ciphertexts
  - Accomplished by adding “random” bits each time you encrypt

Asymmetric Encryption

- Uses two keys
- Also called Public-Key encryption
- **Public key** freely published to everyone
- **Private key** held secret
- Will be covered in later chapters—everything in this chapter is about **symmetric encryption**

When Ciphers Do More Than Encryption

Authenticated Encryption

- Returns an **authentication tag** with the ciphertext
- Tag ensures integrity of the message and also authenticates the author
- **Authenticated Encryption with Associated Data (AEAD)**
  - Another variant

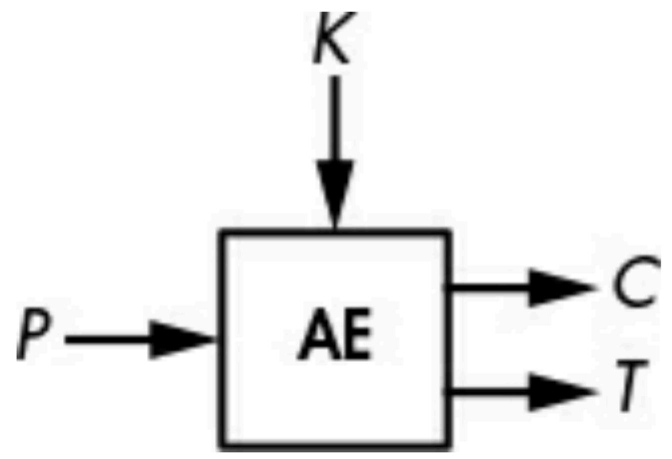


Figure 1-4: Authenticated encryption

Format-Preserving Encryption

- Normally encryption takes inputs as bits and returns outputs as bits
- Could be written as hex, base64, etc.
- Format-Preserving Encryption returns ciphertext in the same format as the plaintext
  - Zip code -> Zip code
  - IP address -> IP address

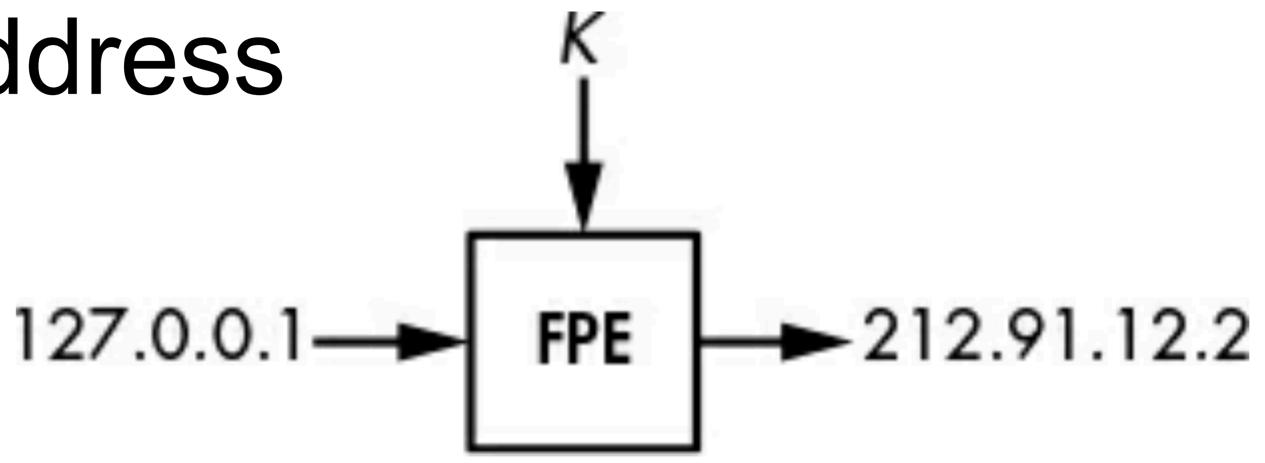


Figure 1-5: Format-preserving encryption for IP addresses

Fully Homomorphic Encryption

- Allows modification of encrypted data without decrypting it
- The first FHE scheme was created in 2009
- Very slow

Searchable Encryption

- Searches encrypted data without decrypting it
- Using an encrypted search string
- Protects privacy of search engine users

Enveil is a pioneering data security company protecting Data in Use.

Never Decrypt.

We use homomorphic encryption techniques to solve this problem in a way no one else can. Whether you're performing searches or analytics on data you own, seeking information from a third-party data provider, or driving revenue by securely monetizing your data assets, **Enveil ensures nothing is ever revealed during the entire processing lifecycle.**

Tweakable Encryption

- Adds a "tweak" parameter to normal encryption
  - Such as a unique customer number
  - Acts like an Initialization Vector in CBC
- Main application is disk encryption

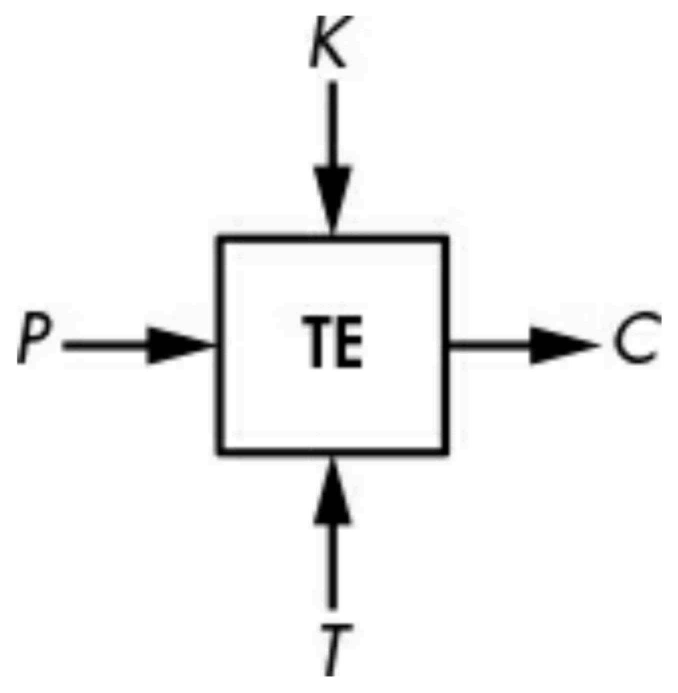


Figure 1-6: Tweakable encryption

How Things Can Go Wrong

Weak Cipher

- 2G phone networks used the A5/1 cipher
- Vulnerable to a **time-memory trade-off** attack
  - Using large lookup tables to speed up an attack

Wrong Model

- Padding Oracle attack
- If a user submitted data that decrypted to a valid string, that was taken as authentication
  - Even if the string contained nonsense
- Server provided error messages for incorrect padding
- Those errors can be used to find valid ciphertext without knowing the key

Kahoot!