Classical Ciphers

MAT364 - Cryptography Course

Instructor: Adil Akhmetov

University: SDU

Week 2

Press Space for next page →

What Are Classical Ciphers?

Definition

Classical ciphers are historical encryption methods used before the computer age. They rely on simple mathematical transformations of text.

Why Study Them?

- **Historical context** - Understanding evolution of crypto
- **Basic principles** - Core concepts still relevant
- **Attack methods** - Learn how to break ciphers
- **Modern applications** - Some concepts still used today

Key Characteristics

- **Manual implementation** - Can be done by hand
- **Simple algorithms** - Easy to understand
- **Weak security** - Vulnerable to modern attacks
- **Educational value** - Foundation for modern crypto

Remember: Classical ciphers are NOT secure for modern applications, but they teach us fundamental principles!

Types of Classical Ciphers

Substitution Ciphers

- **Monoalphabetic** - One alphabet mapping
- **Polyalphabetic** - Multiple alphabet mappings
- **Examples:** Caesar, Atbash, Vigenère

Transposition Ciphers

- **Columnar** - Rearrange columns
- **Rail Fence** - Write in zigzag pattern
- **Route** - Follow specific path

Hybrid Ciphers

- **Combination** of substitution and transposition
- **Multiple steps** for increased security
- **Examples:** ADFGVX cipher

Modern Relevance

- **Stream ciphers** - Use substitution principles
- **Block ciphers** - Use transposition concepts
- **Key scheduling** - Derived from classical methods

Substitution Ciphers

Caesar Cipher Deep Dive

How It Works

- **Shift each letter** by a fixed amount
- **Wrap around** alphabet (Z → A)
- **Same shift** for encryption and decryption
- **Key space:** 25 possible shifts (0-25)

Mathematical Formula

- **Encryption:** `E(x) = (x + k) mod 26`
- **Decryption:** `D(y) = (y - k) mod 26`
- **Where:** x = letter position, k = shift amount

Example Walkthrough

```
Original: "HELLO" (7,4,11,11,14)
Key: 3
Encrypt: (7+3,4+3,11+3,11+3,14+3) mod 26
Result: (10,7,14,14,17) = "KHOOR"

Decrypt: "KHOOR" (10,7,14,14,17)
Key: 3
Decrypt: (10-3,7-3,14-3,14-3,17-3) mod 26
Result: (7,4,11,11,14) = "HELLO"
```

Caesar Cipher Implementation

Python Code

```python
def caesar_cipher(text, shift, mode='encrypt'):
    """Caesar cipher implementation"""
    result = ""
    for char in text:
        if char.isalpha():
            # Determine case
            ascii_offset = 65 if char.isupper() else 97
            # Get letter position (0-25)
            letter_pos = ord(char) - ascii_offset

            if mode == 'encrypt':
                new_pos = (letter_pos + shift) % 26
            else:  # decrypt
                new_pos = (letter_pos - shift) % 26

            result += chr(new_pos + ascii_offset)
```

Why It's Weak

- **Small key space** - Only 25 possible keys
- **Brute force** - Try all 25 shifts
- **Pattern preservation** - Letter frequencies unchanged
- **No confusion** - Each letter always maps to same letter

Breaking Caesar Cipher

1. **Brute force** - Try all 25 shifts
2. **Frequency analysis** - Most common letters
3. **Pattern recognition** - Look for common words
4. **Statistical analysis** - Compare to English letter frequencies

Caesar Cipher Theory

Mathematical Properties

Modular Arithmetic:

- `(a + b) mod n = ((a mod n) + (b mod n)) mod n`
- `(a - b) mod n = ((a mod n) - (b mod n)) mod n`
- `(a + k - k) mod n = a mod n = a`

Why it's reversible:

- Addition and subtraction are inverse operations
- Modular arithmetic preserves this property
- Same key works for both operations

Security Analysis

Key space size: 26 (including identity) **Effective keys:** 25 (excluding no shift) **Time to break:** O(26) = O(1) - constant time **Information leakage:** High - preserves all patterns

Attack complexity:

- **Brute force:** 25 operations maximum
- **Frequency analysis:** O(1) with known language
- **Pattern matching:** O(1) with common words

Security Lesson: Caesar cipher demonstrates why large key spaces and pattern disruption are essential for security!

Monoalphabetic Substitution

How It Works

- **One-to-one mapping** between alphabets
- **Each letter** maps to exactly one other letter
- **Key:** Complete permutation of alphabet
- **Key space:** $26! \approx 4 \times 10^{26}$ possible keys

Example

```
Plain:  A B C D E F G H I J K L M N O P Q R S T U V W
Cipher: Z Y X W V U T S R Q P O N M L K J I H G F E D
```

Implementation

```python
def monoalphabetic_cipher(text, key):
    """Monoalphabetic substitution cipher"""
    result = ""
    for char in text:
        if char.isalpha():
            ascii_offset = 65 if char.isupper() else 9
            letter_pos = ord(char) - ascii_offset
            result += key[letter_pos]
        else:
            result += char
    return result

# Usage
key = "ZYXWVUTSRQPONMLKJIHGFEDCBA"
encrypted = monoalphabetic_cipher("HELLO", key)
```

Breaking Monoalphabetic Ciphers

Frequency Analysis

English letter frequencies:

- E: 12.7%, T: 9.1%, A: 8.2%, O: 7.5%
- I: 7.0%, N: 6.7%, S: 6.3%, H: 6.1%
- R: 6.0%, D: 4.3%, L: 4.0%, C: 2.8%

Method:

1. Count letter frequencies in ciphertext
2. Match to known English frequencies
3. Use common words and patterns
4. Refine mapping iteratively

Common Patterns

Digraphs (two letters):

- TH, HE, IN, ER, AN, RE, ED, ND, ON, EN

Trigraphs (three letters):

- THE, AND, FOR, ARE, BUT, NOT, YOU, ALL, CAN, HER

Word patterns:

- Single letters: A, I
- Common words: THE, AND, OR, TO, OF, IN, IS, IT, AS, BE

Historical Note: Frequency analysis was first described by Al-Kindi in the 9th century - one of the most important breakthroughs in cryptanalysis!

Vigenère Cipher

How It Works

- **Polyalphabetic substitution** - Multiple Caesar ciphers
- **Keyword determines** which Caesar cipher to use
- **Cycles through** keyword letters
- **Key space:** 26^k where k = keyword length

Example

```
Keyword: "KEY" (10, 4, 24)
Message: "HELLO" (7, 4, 11, 11, 14)
Encrypt: (7+10, 4+4, 11+24, 11+10, 14+4) mod 26
Result: (17, 8, 9, 21, 18) = "RIJVS"
```

Implementation

```python
def vigenere_cipher(text, key, mode='encrypt'):
    """Vigenère cipher implementation"""
    result = ""
    key_index = 0
    for char in text:
        if char.isalpha():
            ascii_offset = 65 if char.isupper() else 9
            letter_pos = ord(char) - ascii_offset
            key_shift = ord(key[key_index % len(key)].

            if mode == 'encrypt':
                new_pos = (letter_pos + key_shift) % 2
            else:
                new_pos = (letter_pos - key_shift) % 2

            result += chr(new_pos + ascii_offset)
```

Vigenère Cipher Theory

Why It's Stronger

Key space explosion:

- Caesar: 25 keys
- Vigenère (3 chars): $26^3 = 17{,}576$ keys
- Vigenère (10 chars): $26^{10} \approx 1.4 \times 10^{14}$ keys

Pattern disruption:

- Same letter can map to different ciphertext letters
- Depends on position in keyword cycle
- Breaks frequency analysis for short keywords

Mathematical Foundation

Encryption formula: `C_i = (P_i + K_{i mod |K|}) mod 26`

Decryption formula: `P_i = (C_i - K_{i mod |K|}) mod 26`

Where:

- C_i = ciphertext character at position i
- P_i = plaintext character at position i
- K = keyword
- |K| = keyword length

Historical Impact: Vigenère cipher was considered unbreakable for 300 years until Kasiski's method in 1863!

Transposition Ciphers

Columnar Transposition

How It Works

1. **Write message** in rows of fixed width
2. **Read columns** in specific order
3. **Key determines** column reading order
4. **Example:** Key "KEY" → columns 2,4,1,3

Example

```
Key: "KEY" (2,4,1,3)
Message: "HELLO WORLD"
Width: 4

H E L L
O   W O
R L D

Read columns 2,4,1,3:
Column 2: E, ,L → "EL"
Column 4: L,O,  → "LO"
Column 1: H,O,R → "HOR"
Column 3: L,W,D → "LWD"
```

Implementation

```python
def columnar_transposition(text, key, mode='encrypt'):
    """Columnar transposition cipher"""
    # Remove spaces and pad if necessary
    text = text.replace(' ', '').upper()

    if mode == 'encrypt':
        # Create matrix
        width = len(key)
        rows = (len(text) + width - 1) // width
        matrix = [[''] * width for _ in range(rows)]

        # Fill matrix
        for i, char in enumerate(text):
            row, col = divmod(i, width)
            matrix[row][col] = char
```

Rail Fence Cipher

How It Works

- **Write message** in zigzag pattern
- **Number of rails** determines the pattern
- **Read row by row** to get ciphertext
- **Example with 3 rails:**

```
Message: "HELLO WORLD"
Rails: 3

H . . . O . . . R . .
. E . L . W . L . D .
. . L . . . O . . . .

Ciphertext: "HOREWLDLO"
```

Implementation

```
def rail_fence_cipher(text, rails, mode='encrypt'):
    """Rail fence cipher implementation"""
    text = text.replace(' ', '').upper()

    if mode == 'encrypt':
        # Create rail pattern
        pattern = []
        direction = 1
        rail = 0

        for i in range(len(text)):
            pattern.append((rail, i))
            rail += direction
            if rail == rails - 1 or rail == 0:
                direction = -direction
```

Transposition Cipher Theory

Why Transposition Works

Positional scrambling:

- **Changes letter positions** without changing letters
- **Preserves letter frequencies** - same letters, different order
- **Creates confusion** through position changes
- **Reversible** - can reconstruct original order

Mathematical properties:

- **Permutation** of positions
- **Bijective mapping** - one-to-one correspondence
- **Inverse exists** - can reverse the transformation

Security Analysis

Strengths:

- **Large key space** - n! permutations for n positions
- **Pattern disruption** - breaks word boundaries
- **Frequency preservation** - harder to analyze

Weaknesses:

- **Anagram attacks** - rearrange letters
- **Pattern recognition** - common letter combinations
- **Statistical analysis** - digraph/trigraph frequencies

Key Insight: Transposition ciphers show that changing position can be as important as changing content for security!

Breaking Classical Ciphers

Cryptanalysis Techniques

Brute Force Attacks

Caesar Cipher:

- Try all 25 possible shifts
- Time complexity: O(26)
- Check for readable text

Monoalphabetic:

- Try common substitution patterns
- Use frequency analysis
- Time complexity: O(26!)

Statistical Attacks

Frequency Analysis:

- Compare letter frequencies
- Use known language statistics
- Look for common patterns

Digraph/Trigraph Analysis:

- Analyze two/three letter combinations
- Use known language patterns
- Refine substitution mapping

Kasiski Method (Breaking Vigenère)

The Method

1. **Find repeated patterns** in ciphertext
2. **Measure distances** between repetitions
3. **Find GCD** of distances
4. **Estimate keyword length**

Example

```
Ciphertext: "WICVQRXWICVQRXWICVQRX"
Pattern "WICVQRX" repeats every 7 characters
GCD of distances: 7
Estimated keyword length: 7
```

Implementation

```python
def kasiski_method(ciphertext):
    """Estimate keyword length using Kasiski method"""
    # Find repeated patterns
    patterns = {}
    for i in range(len(ciphertext) - 2):
        pattern = ciphertext[i:i+3]
        if pattern in patterns:
            patterns[pattern].append(i)
        else:
            patterns[pattern] = [i]

    # Calculate distances
    distances = []
    for pattern, positions in patterns.items():
        if len(positions) > 1:
            for i in range(1, len(positions)):
```

Modern Relevance of Classical Ciphers

Concepts Still Used

Substitution principles:

- **S-boxes** in block ciphers
- **Stream ciphers** - XOR with key stream
- **Hash functions** - substitution operations

Transposition principles:

- **P-boxes** in block ciphers
- **Bit shuffling** operations
- **Round functions** in Feistel networks

Educational Value

Learning objectives:

- **Understand basic principles** of encryption
- **Learn attack methods** and their evolution
- **Appreciate security requirements** for modern crypto
- **Develop intuition** for cryptographic design

Historical context:

- **Evolution of cryptography** over centuries
- **Importance of key management** and distribution
- **Need for formal security analysis**

Modern Lesson: Classical ciphers teach us that security requires both confusion (substitution) and diffusion (transposition)!

Practical Implementation

Complete Cipher Suite

Caesar Cipher Class

```
class CaesarCipher:
    def __init__(self, shift):
        self.shift = shift % 26

    def encrypt(self, text):
        return self._transform(text, self.shift)

    def decrypt(self, text):
        return self._transform(text, -self.shift)

    def _transform(self, text, shift):
        result = ""
        for char in text:
            if char.isalpha():
                ascii_offset = 65 if char.isupper() el
                letter_pos = ord(char) - ascii_offset
```

Vigenère Cipher Class

```
class VigenereCipher:
    def __init__(self, key):
        self.key = key.upper()

    def encrypt(self, text):
        return self._transform(text, 1)

    def decrypt(self, text):
        return self._transform(text, -1)

    def _transform(self, text, direction):
        result = ""
        key_index = 0
        for char in text:
            if char.isalpha():
                ascii_offset = 65 if char.isupper() el
```

Frequency Analysis Tool

Implementation

```
def frequency_analysis(text):
    """Analyze letter frequencies in text"""
    text = text.upper().replace(' ', '')
    frequencies = {}
    total_chars = len(text)

    for char in text:
        if char.isalpha():
            frequencies[char] = frequencies.get(char,

    # Convert to percentages
    for char in frequencies:
        frequencies[char] = (frequencies[char] / total

    return dict(sorted(frequencies.items(),
                       key=lambda x: x[1], reverse=True
```

English Letter Frequencies

```
ENGLISH_FREQUENCIES = {
    'E': 12.7, 'T': 9.1, 'A': 8.2, 'O': 7.5, 'I': 7.0,
    'N': 6.7, 'S': 6.3, 'H': 6.1, 'R': 6.0, 'D': 4.3,
    'L': 4.0, 'C': 2.8, 'U': 2.8, 'M': 2.4, 'W': 2.4,
    'F': 2.2, 'G': 2.0, 'Y': 2.0, 'P': 1.9, 'B': 1.3,
    'V': 1.0, 'K': 0.8, 'J': 0.2, 'X': 0.2, 'Q': 0.1,
    'Z': 0.1
}

# Usage
ciphertext = "KHOOR ZRUOG"
freq = frequency_analysis(ciphertext)
correlation = compare_frequencies(freq, ENGLISH_FREQUE
print(f"Correlation: {correlation:.2f}")
```

Practical Tasks

Task 1: Cipher Implementation

Create a comprehensive cipher suite:

1. **Caesar Cipher** with brute force attack
2. **Vigenère Cipher** with keyword analysis
3. **Frequency Analysis** tool
4. **Interactive menu** for cipher selection

Task 2: Cryptanalysis

Implement attack methods:

1. **Brute force** Caesar cipher
2. **Frequency analysis** for monoalphabetic
3. **Kasiski method** for Vigenère
4. **Pattern recognition** for transposition

Requirements

- **Clean, documented code**
- **Error handling** for edge cases
- **Interactive interface** for testing
- **Statistical analysis** tools
- **Git repository** with proper commits

Learning Goals

- **Understand** how classical ciphers work
- **Learn** basic cryptanalysis techniques
- **Appreciate** why modern crypto is needed
- **Develop** intuition for security design

Goal: Build a complete classical cipher toolkit and learn to break them using historical methods!

Common Mistakes to Avoid

Implementation Errors

- ❌ **Case sensitivity** - Handle upper/lower case properly
- ❌ **Non-alphabetic characters** - Preserve spaces, punctuation
- ❌ **Key validation** - Check for valid keys
- ❌ **Edge cases** - Empty strings, single characters

Best Practices

- ✅ **Use established libraries** for real cryptography
- ✅ **Implement for learning** only
- ✅ **Document security limitations**
- ✅ **Test with known examples**
- ✅ **Handle errors gracefully**

Security Mistakes

- ❌ **Using classical ciphers** for real security
- ❌ **Weak key generation** - Use proper randomness
- ❌ **Key reuse** - Don't reuse keys across messages
- ❌ **Ignoring patterns** - Consider frequency analysis

Important: Classical ciphers are for educational purposes only. Never use them for real security applications!

Questions?

Let's discuss classical ciphers! 💬

Next Week: We'll explore modern stream ciphers and learn why they're much more secure!

Assignment: Implement and break classical ciphers to understand their weaknesses!