Kasiski Method Deep Dive

Breaking the Vigenère Cipher

Instructor: Adil Akhmetov

University: SDU

Course: MAT364 - Cryptography

Press Space for next page →

Historical Context

The Vigenère Cipher

- **Invented:** 1553 by Giovan Battista Bellaso
- **Popularized:** 1586 by Blaise de Vigenère
- **Considered unbreakable** for 300+ years
- **Used by:** Military, diplomats, spies

Friedrich Kasiski (1805-1881)

- **Prussian officer** and cryptanalyst
- **Published method** in 1863
- **First systematic attack** on Vigenère cipher
- **Revolutionary breakthrough** in cryptanalysis

Why It Was "Unbreakable"

- **Multiple alphabets** - not just one substitution
- **Key-dependent** - different shifts for each position
- **Breaks frequency analysis** - same letter maps to different ciphertext
- **Large key space** - 26^k possible keys

The Breakthrough

- **Pattern recognition** in repeated sequences
- **Mathematical analysis** of distances
- **Key length estimation** using GCD
- **Foundation** for modern cryptanalysis

Historical Impact: Kasiski's method ended the era of "unbreakable" Vigenère cipher and revolutionized cryptanalysis!

The Core Insight

Why Patterns Repeat

Key insight: When the same plaintext pattern appears at positions where the keyword repeats, it produces the same ciphertext pattern.

Example:

- Plaintext: "THE" appears at positions 0 and 6
- Keyword: "KEY" (length 3)
- At position 0: T + K, H + E, E + Y
- At position 6: T + K, H + E, E + Y (same!)

Mathematical Foundation

Distance between repetitions:

- If pattern repeats at positions i and j
- Distance = j - i
- If keyword length divides this distance
- Same keyword letters are used
- Same ciphertext pattern results

Key length estimation:

- Find all distances between repetitions
- Calculate GCD of all distances
- GCD is likely the keyword length

Key Insight: Repeated patterns in ciphertext reveal the keyword length through mathematical analysis of distances!

Step-by-Step Process

1 Find Repeated Patterns

1. **Scan ciphertext** for repeated sequences
2. **Minimum length** of 3 characters
3. **Record positions** of each occurrence
4. **Ignore single occurrences**

2 Calculate Distances

1. **For each pattern** with multiple occurrences
2. **Calculate distances** between all pairs
3. **Record all distances** found
4. **Note frequency** of each distance

3 Find GCD

1. **List all distances** found
2. **Calculate GCD** of all distances
3. **Consider common factors** of distances
4. **Estimate keyword length**

4 Verify and Refine

1. **Test estimated length** by grouping letters
2. **Apply frequency analysis** to each group
3. **Refine estimate** if needed
4. **Proceed to key recovery**

Detailed Example Walkthrough

Given Information

Ciphertext: "WICVQRXWICVQRXWICVQRX"

Plaintext: "ATTACKATDAWN"

Keyword: "BATTLE" (length 6)

Step 1: Find Patterns

Looking for repeated sequences of length 3+:

Pattern "WIC":

- Position 0: **WIC**
- Position 7: **WIC**
- Position 14: **WIC**

Pattern "VQR":

- Position 3: **VQR**

Step 2: Calculate Distances

For pattern "WIC":

- Distance 1: 7 - 0 = **7**
- Distance 2: 14 - 7 = **7**
- Distance 3: 14 - 0 = **14**

For pattern "VQR":

- Distance 1: 10 - 3 = **7**
- Distance 2: 17 - 10 = **7**
- Distance 3: 17 - 3 = **14**

All distances: [**7** , **7** , **14** , **7** , **7** , **14**]

GCD Calculation

Finding the GCD

Distances found: [7, 7, 14, 7, 7, 14]

GCD calculation:

- GCD(7, 7) = **7**
- GCD(7, 14) = **7**
- GCD(7, 7) = **7**
- GCD(7, 14) = **7**
- GCD(7, 7) = **7**

Result: GCD = **7**

Verification

Keyword length: 6 (actual)

Estimated length: 7 (from GCD)

Why the difference?

Refined Analysis

Common factors of distances:

- 7: appears 4 times
- 14: appears 2 times
- 2: factor of 14
- 1: factor of all

Most likely candidates:

- 7 (most frequent)
- 2 (factor of 14)
- 1 (too small)

Best estimate: 7 (closest to actual 6)

Python Implementation

Basic Kasiski Method

```python
import math
from collections import defaultdict

def kasiski_method(ciphertext, min_pattern_length=3):
    """Estimate keyword length using Kasiski method"""

    # Step 1: Find repeated patterns
    patterns = defaultdict(list)

    for i in range(len(ciphertext) - min_pattern_lengt
        pattern = ciphertext[i:i + min_pattern_length]
        patterns[pattern].append(i)

    # Filter patterns with multiple occurrences
    repeated_patterns = {p: pos for p, pos in patterns
                         if len(pos) > 1}
```

Distance Calculation

```python
def calculate_distances(repeated_patterns):
    """Calculate distances between pattern occurrences
    all_distances = []

    for pattern, positions in repeated_patterns.items(
        print(f"\nPattern '{pattern}':")

        # Calculate all pairwise distances
        for i in range(len(positions)):
            for j in range(i + 1, len(positions)):
                distance = positions[j] - positions[i]
                all_distances.append(distance)
                print(f"  Distance: {positions[j]} - {

    return all_distances

[illegible]
```

Complete Implementation

Full Kasiski Analysis

```python
def complete_kasiski_analysis(ciphertext, min_pattern_
    """Complete Kasiski analysis with detailed output"

    print("=== KASISKI METHOD ANALYSIS ===")
    print(f"Ciphertext: {ciphertext}")
    print(f"Length: {len(ciphertext)}")
    print()

    # Step 1: Find repeated patterns
    patterns = kasiski_method(ciphertext, min_pattern_

    if not patterns:
        print("No repeated patterns found!")
        return None

    # Step 2: Calculate distances
```

Usage Example

```python
# Example usage
ciphertext = "WICVQRXWICVQRXWICVQRX"
estimated_length = complete_kasiski_analysis(ciphertex

print(f"\n=== RESULT ===")
print(f"Estimated keyword length: {estimated_length}")

# Test with different minimum pattern lengths
for min_len in [2, 3, 4, 5]:
    print(f"\n--- Testing with min pattern length {min
    result = complete_kasiski_analysis(ciphertext, min
    if result:
        print(f"Estimated length: {result}")
```

Advanced Analysis

Statistical Refinement

```python
def statistical_kasiski(ciphertext, min_pattern_length
    """Enhanced Kasiski with statistical analysis"""

    patterns = kasiski_method(ciphertext, min_pattern_
    distances = calculate_distances(patterns)

    if not distances:
        return None

    # Statistical analysis
    from collections import Counter
    distance_counts = Counter(distances)

    print("Distance frequency analysis:")
    for distance, count in distance_counts.most_common
        print(f"  Distance {distance}: {count} occurre
```

Multiple Pattern Lengths

```python
def multi_length_kasiski(ciphertext):
    """Test multiple pattern lengths for better accura

    results = {}

    for min_len in range(2, min(8, len(ciphertext) //
        print(f"\n=== Testing pattern length {min_len}
        result = statistical_kasiski(ciphertext, min_l
        if result:
            results[min_len] = result

    # Find consensus
    if results:
        length_counts = Counter(results.values())
        consensus_length = length_counts.most_common(1
```

Visualization of the Method

Pattern Visualization

```
Ciphertext: WICVQRXWICVQRXWICVQRX
Position:   012345678901234567890 1

Pattern "WIC":
  Position 0:  WIC
  Position 7:       WIC
  Position 14:            WIC
  Distances: 7, 7, 14

Pattern "VQR":
  Position 3:    VQR
  Position 10:        VQR
  Position 17:             VQR
  Distances: 7, 7, 14
```

Distance Analysis

```
All distances: [7, 7, 14, 7, 7, 14]

Factor analysis:
  Factor 1: appears 6 times
  Factor 7: appears 6 times
  Factor 2: appears 2 times
  Factor 14: appears 2 times

GCD calculation:
  GCD(7, 7) = 7
  GCD(7, 14) = 7
  GCD(7, 7) = 7
  GCD(7, 14) = 7
  GCD(7, 7) = 7

Result: 7
```

Limitations and Improvements

Limitations

Short ciphertext:

- **Insufficient patterns** for reliable analysis
- **Random coincidences** may mislead
- **Need longer text** for accuracy

Short keywords:

- **Fewer repetitions** expected
- **Harder to detect** patterns
- **Less reliable** estimates

Noisy data:

- **Transmission errors** can break patterns
- **Character substitutions** affect analysis
- **Need error correction**

Improvements

Friedman's Index of Coincidence:

- **Statistical measure** of pattern repetition
- **More reliable** than simple pattern matching
- **Works better** with shorter texts

Modern enhancements:

- **Machine learning** approaches
- **Probabilistic analysis** of patterns
- **Combined methods** for better accuracy

Practical considerations:

- **Multiple tests** with different parameters
- **Cross-validation** with other methods
- **Human verification** of results

Practical Application

Real-World Example

Ciphertext: "QPWKA LVRXC QZIKR VWRKA TTTZT ZDOKR XHIXA VFOCL QFZFC RHG"

Analysis:

1. **Find patterns** of length 3+
2. **Calculate distances** between repetitions
3. **Find GCD** of all distances
4. **Estimate keyword length**

Expected result: Keyword length around 6-8

Complete Workflow

```python
def break_vigenere(ciphertext):
    """Complete Vigenère breaking workflow"""

    # Step 1: Estimate keyword length
    key_length = multi_length_kasiski(ciphertext)
    print(f"Estimated keyword length: {key_length}")

    # Step 2: Group letters by position
    groups = [[] for _ in range(key_length)]
    for i, char in enumerate(ciphertext):
        if char.isalpha():
            groups[i % key_length].append(char)

    # Step 3: Apply frequency analysis to each group
    for i, group in enumerate(groups):
        print(f"Group {i}: {''.join(group)}")
```

Mathematical Foundation

Probability Theory

Expected repetitions:

- **Probability** of pattern repetition
- **Depends on** pattern length and text length
- **Higher probability** for longer patterns
- **Lower probability** for shorter patterns

Statistical significance:

- **Chi-square test** for pattern significance
- **Z-score** for distance analysis
- **Confidence intervals** for estimates

Information Theory

Entropy analysis:

- **Measure randomness** in ciphertext
- **Compare with** expected entropy
- **Identify patterns** that reduce entropy
- **Quantify information** gained

Mutual information:

- **Measure dependence** between positions
- **Identify key length** through dependencies
- **More robust** than simple GCD

Modern Extensions

Machine Learning Approaches

Neural networks:

- **Learn patterns** automatically
- **Handle noise** better than traditional methods
- **Adapt to** different cipher types
- **Improve accuracy** with training

Deep learning:

- **CNN for** pattern recognition
- **RNN for** sequence analysis
- **Transformer** for attention-based analysis
- **State-of-the-art** performance

Quantum Cryptanalysis

Quantum algorithms:

- **Grover's algorithm** for key search
- **Quantum Fourier transform** for pattern analysis
- **Quantum machine learning** for cryptanalysis
- **Future-proof** methods

Hybrid approaches:

- **Classical preprocessing** + quantum analysis
- **Quantum pattern recognition**
- **Quantum statistical analysis**
- **Next-generation** cryptanalysis

Hands-On Exercise

Exercise: Break This Cipher

Ciphertext: "QPWKA LVRXC QZIKR VWRKA TTTZT ZDOKR XHIXA VFOCL QFZFC RHG"

Tasks:

1. **Implement** Kasiski method
2. **Find repeated patterns** of length 3+
3. **Calculate distances** between repetitions
4. **Estimate keyword length** using GCD
5. **Verify** with frequency analysis

Solution Steps

```
# Step 1: Clean and prepare ciphertext
ciphertext = "QPWKA LVRXC QZIKR VWRKA TTTZT ZDOKR XHIX
clean_text = ciphertext.replace(" ", "")

# Step 2: Apply Kasiski method
key_length = complete_kasiski_analysis(clean_text)

# Step 3: Group by estimated key length
groups = [[] for _ in range(key_length)]
for i, char in enumerate(clean_text):
    groups[i % key_length].append(char)

# Step 4: Analyze each group
for i, group in enumerate(groups):
    print(f"Group {i}: {''.join(group)}")
    # Apply frequency analysis to find key letter
```

Common Pitfalls

Implementation Errors

❌ **Insufficient pattern length:**

- Using patterns too short (1-2 characters)
- High chance of random coincidences
- **Solution:** Use patterns of length 3+

❌ **Ignoring factor analysis:**

- Only using GCD without considering factors
- Missing better estimates
- **Solution:** Analyze all factors and their frequencies

Analysis Mistakes

❌ **Single test reliance:**

- Using only one pattern length
- Not cross-validating results
- **Solution:** Test multiple pattern lengths

❌ **Ignoring statistical significance:**

- Not considering probability of random patterns
- **Solution:** Use statistical tests for significance

Best Practices

Implementation Tips

✅ **Use multiple pattern lengths:**

- Test patterns of length 2, 3, 4, 5+
- Find consensus among results
- Increase confidence in estimate

✅ **Statistical validation:**

- Use chi-square tests for significance
- Calculate confidence intervals
- Verify with other methods

Analysis Guidelines

✅ **Cross-validation:**

- Compare with Friedman’s method
- Use frequency analysis to verify
- Test with known examples

✅ **Error handling:**

- Handle cases with no patterns
- Provide fallback methods
- Give confidence measures

Questions?

Let's discuss the Kasiski method! 💬

Key Takeaway: The Kasiski method is a brilliant example of how mathematical analysis can break seemingly unbreakable ciphers!

Next Steps: Practice implementing the method and try it on various Vigenère ciphers!