Mandelbrot Fractal Encryption (MFE)

Technical Documentation

Version 1.0

Timur Gabaidulin

December 20, 2024

Contents

1 Overview **2**

2 Technical Specifications **2**
2.1 Core Components 2
2.1.1 Fractal Generation Parameters 2
2.1.2 Character Mapping 2

3 Encryption Process **2**
3.1 Input Processing 2
3.2 Encryption Algorithm 2
3.3 Output Format 2

4 Decryption Process **3**
4.1 Ciphertext Processing 3
4.2 Character Recovery Algorithm 3

5 Security Considerations **3**
5.1 Strengths 3
5.2 Limitations 3

6 Implementation Requirements **3**
6.1 Dependencies 3
6.2 Memory Requirements 3
6.3 Performance Considerations 4

7 GUI Application Features **4**
7.1 Message Input 4
7.2 Encryption Controls 4
7.3 Decryption Controls 4
7.4 Visualization 4

8 Usage Examples **4**

9 Error Handling **4**
9.1 Input Validation 4
9.2 Runtime Protection 5

10 Future Improvements **5**
10.1 Security Enhancements 5
10.2 Performance Optimization 5
10.3 Feature Additions 5

1 Overview

Mandelbrot Fractal Encryption (MFE) is a novel encryption method that leverages the mathematical properties of the Mandelbrot set to create a unique encryption scheme. The algorithm uses fractal geometry to map plaintext characters to specific points in the complex plane, generating ciphertext based on the iterative behavior of these points under the Mandelbrot set formula.

2 Technical Specifications

2.1 Core Components

2.1.1 Fractal Generation Parameters

- Complex plane boundaries: $x \in [-2, 1]$, $y \in [-1.5, 1.5]$
- Resolution: 500×500 pixels
- Maximum iterations: 100
- Default key point: $(-0.5, 0)$

2.1.2 Character Mapping

- Each character is mapped to a unique point in the complex plane
- Position calculation: $unique\_point = key\_point + (char\_value \times 0.1) + (index \times 0.001)$
- Character value range: 0-255 (ASCII)

3 Encryption Process

3.1 Input Processing

The encryption process begins with sequential processing of each character in the plaintext. The character's position within the message influences its mapping to ensure uniqueness.

3.2 Encryption Algorithm

The core encryption algorithm can be expressed as follows:

```
For each character:
    1. Calculate unique complex point (c) based on:
       - Character ASCII value
       - Character position in message
       - Base key point
    2. Perform Mandelbrot iteration: z = z   + c
    3. Track iteration count until |z| > 2 or max iterations reached
    4. Store (iteration_count, final_z_value) as ciphertext
```

3.3 Output Format

- Ciphertext consists of tuples: $(iteration\_count, complex\_number)$
- Each character generates one tuple
- Tuples are concatenated to form complete ciphertext

4 Decryption Process

4.1 Ciphertext Processing

The decryption process involves:

- Parsing input tuples containing iteration counts and z-values
- Reconstructing character mapping using key point

4.2 Character Recovery Algorithm

```
For each encrypted tuple:
    1. Extract target iteration count and z-value
    2. Test all possible ASCII values (0-255)
    3. For each test value:
       - Calculate unique point using same formula as encryption
       - Perform Mandelbrot iteration
       - Compare resulting iteration count and z-value
    4. Select character with closest match
```

5 Security Considerations

5.1 Strengths

- Non-linear transformation using fractal mathematics
- Position-dependent character encoding
- Complex number precision requirements
- Multiple parameters affecting output (iteration count, z-value)

5.2 Limitations

- Deterministic mapping may be vulnerable to frequency analysis
- Computational complexity scales with message length
- Sensitive to floating-point precision errors
- Key space limited by viable Mandelbrot set points

6 Implementation Requirements

6.1 Dependencies

- Python 3.x
- NumPy: Mathematical operations
- Matplotlib: Fractal visualization
- Tkinter: GUI implementation

6.2 Memory Requirements

- Base memory: ~10MB
- Runtime memory: $O(width \times height)$ for fractal generation
- Message processing: $O(n)$ where n is message length

6.3 Performance Considerations

- Encryption time: $O(n \times max\_iterations)$
- Decryption time: $O(n \times 256 \times max\_iterations)$
- Visualization memory: $O(width \times height)$

7 GUI Application Features

7.1 Message Input

- Text entry field for plaintext
- Support for ASCII character set

7.2 Encryption Controls

- One-click encryption button
- Automatic parameter handling
- Error checking for invalid input

7.3 Decryption Controls

- Ciphertext input field
- Automatic format validation
- Error handling for invalid ciphertext

7.4 Visualization

- Real-time Mandelbrot set display
- Heat map coloring for iteration counts
- Interactive fractal viewing

8 Usage Examples

```
# Encryption Example
message = "Hello, World!"
key_point = (-0.5, 0)
ciphertext = encrypt_message(message, key_point, -2, 1, -1.5, 1.5, 500, 500, 100)

# Decryption Example
decrypted = decrypt_message(ciphertext, key_point, -2, 1, -1.5, 1.5, 500, 500, 100)
```

9 Error Handling

9.1 Input Validation

- Empty message checking
- Character range validation
- Key point boundary verification

9.2 Runtime Protection

- Exception handling for invalid ciphertext
- Floating-point overflow prevention
- Memory allocation safety checks

10 Future Improvements

10.1 Security Enhancements

- Multiple fractal key points
- Random padding addition
- Iteration count randomization

10.2 Performance Optimization

- Parallel processing support
- GPU acceleration
- Optimized character mapping

10.3 Feature Additions

- File encryption support
- Custom parameter configuration
- Extended character set support