STEGANOGRAPHY ANALYZER - Complete Project Documentation

🎯 Project Overview

Project Name: Steganography
Purpose: A comprehensive desktop application for hiding secret messages inside image and audio files using LSB (Least Significant Bit) steganography techniques
Date: November 2025
Programming Language: Python 3.12

📋 What is Steganography?

Steganography is the practice of concealing messages or information within other non-secret data. Unlike encryption (which scrambles data), steganography hides the very existence of the message.

Real-world Applications:
- Digital watermarking for copyright protection
- Covert communication channels
- Data integrity verification
- Confidential data transmission
- Secure authentication systems

🎨 Project Features

1. IMAGE STEGANOGRAPHY
What it does: Hides secret text messages inside PNG images without visible changes

Technical Implementation:
- Uses LSB (Least Significant Bit) substitution technique
- Modifies the last bit of each RGB color channel
- Supports encryption with AES-256 for added security
- Capacity: (Image Width × Height × 3 channels) ÷ 8 bytes
- Example: 1920×1080 image can hide ~777 KB of data

Why PNG?
- Lossless compression preserves LSB modifications
- JPG uses lossy compression that destroys hidden data

Process:
1. Load PNG image as pixel array (RGB values)
2. Convert secret message to binary
3. Replace LSB of each color channel with message bits
4. Save modified image (visually identical to original)
5. Decode by extracting LSBs and converting back to text

2. AUDIO STEGANOGRAPHY

What it does: Embeds secret messages in WAV audio files with two modes

Technical Implementation:
- **Sequential Mode:** Hides data in consecutive audio samples
- **Key-Based Mode:** Randomizes embedding positions using a secret key
- Uses LSB substitution on 16-bit audio samples
- Inaudible to human ear (changes < 0.0015% of sample value)
- Capacity: (Number of Samples) ÷ 8 bytes
- Example: 60-second audio at 44.1kHz can hide ~330 KB

Key-Based Security:
- Uses SHA-256 hash of user key to generate pseudo-random positions
- Same key required for encoding and decoding
- Without correct key, message appears as noise

Process:
1. Load WAV file as sample array
2. Convert message to binary
3. Embed bits into LSB of samples (sequential or random positions)
4. Save modified audio (sounds identical to original)
5. Decode by extracting LSBs from same positions

3. ENCRYPTION SUPPORT
What it does: Encrypts messages before embedding for double-layer security

Technical Implementation:
- AES-256 encryption in CBC mode
- PBKDF2 key derivation with 100,000 iterations
- Random IV (Initialization Vector) for each encryption
- Base64 encoding for text representation

Security Layers:
1. Encrypt message with password
2. Embed encrypted message in media
3. To recover: Extract from media → Decrypt with password

🛠️ Technologies & Tools Used

Programming Language
- **Python 3.12** - Modern, high-level programming language
  - Reason: Extensive libraries for image/audio processing
  - Cross-platform compatibility (Windows, Mac, Linux)

Core Python Libraries

1. **Pillow (PIL) 10.0.0+**
- **Purpose:** Image processing and manipulation
- **Used For:**
  - Loading PNG images as NumPy arrays
  - Converting between image formats

  - Pixel-level access and modification
  - Saving modified images
- **Key Functions:** `Image.open()`, `Image.save()`, `numpy.array()`

2. **NumPy 1.24.0+**
- **Purpose:** Numerical computing and array operations
- **Used For:**
  - Efficient pixel array manipulation
  - Binary operations (bit shifting, masking)
  - Fast mathematical computations
  - Array reshaping and slicing
- **Key Functions:** `np.array()`, `np.frombuffer()`, bit operations

3. **SciPy 1.11.0+**
- **Purpose:** Scientific computing (specifically audio I/O)
- **Used For:**
  - Reading WAV audio files
  - Writing modified audio data
  - Sample rate handling
  - Audio format conversions
- **Key Functions:** `scipy.io.wavfile.read()`,
`scipy.io.wavfile.write()`

4. **PyCryptodome 3.19.0+**
- **Purpose:** Cryptographic operations
- **Used For:**
  - AES-256 encryption/decryption
  - PBKDF2 key derivation
  - Random IV generation
  - Secure password hashing
- **Key Functions:** `AES.new()`, `PBKDF2()`, `get_random_bytes()`

5. **Tkinter (Built-in)**
- **Purpose:** GUI (Graphical User Interface) framework
- **Used For:**
  - Creating desktop application window
  - Tabbed interface for image/audio modes
  - File selection dialogs
  - Text input/output areas
  - Buttons and controls
- **Key Components:** `Tk()`, `ttk.Notebook`, `ScrolledText`,
`messagebox`

6. **Wave (Built-in)**
- **Purpose:** WAV audio file handling
- **Used For:**
  - Reading WAV file metadata
  - Sample rate and bit depth information
  - Audio file validation

7. **Threading (Built-in)**
- **Purpose:** Concurrent execution
- **Used For:**
  - Background processing to prevent GUI freezing

```
  - Responsive UI during encoding/decoding
  - Progress indication

### **Development Tools**

#### **IDE/Editor**
- **Visual Studio Code** - Primary development environment
  - Python extension for IntelliSense
  - Integrated terminal
  - Git integration
  - Debugging tools

#### **Version Control**
- **Git** - Source code management
  - Track changes and history
  - Backup and recovery

#### **Package Management**
- **pip** - Python package installer
  - Install dependencies from requirements.txt
  - Manage library versions

#### **Virtual Environment**
- **venv** - Isolated Python environment
  - Prevents dependency conflicts
  - Project-specific package versions

---

## 📁 Project Structure

```
STAGNOGRAPHY ANALIZER/
│
├── main_gui.py   # Main application entry point (619
lines)
│   └── SteganographyGUI class with tabbed interface
│
├── modules/   # Core steganography modules
│   ├── __init__.py   # Package initialization
│   ├── image_steg.py   # Image steganography implementation (289
lines)
│   │   ├── ImageSteganography class
│   │   ├── encode_image() - Hide message in PNG
│   │   ├── decode_image() - Extract message from PNG
│   │   └── calculate_capacity() - Determine max message size
│   │
│   └── audio_steg.py   # Audio steganography implementation (312
lines)
│   ├── AudioSteganography class
│   ├── encode_audio() - Hide message in WAV
│   ├── decode_audio() - Extract message from WAV
│   ├── _generate_positions() - Key-based randomization
│   └── calculate_capacity() - Determine max message size
```
```

```
│
├── utils/                      # Utility functions
│   ├── __init__.py
│   └── helpers.py              # Helper classes (198 lines)
│       ├── EncryptionHelper - AES encryption/decryption
│       ├── FileHelper - File validation
│       └── BinaryConverter - Text ↔ Binary conversion
│
├── tests/                      # Unit tests
│   ├── test_image.py           # Image steganography tests
│   ├── test_audio.py           # Audio steganography tests
│   └── test_utils.py           # Utility function tests
│
├── examples.py                 # Example usage demonstrations
├── verify_installation.py      # Dependency checker
├── requirements.txt            # Python dependencies
├── run.bat                     # Windows launcher
├── run.sh                      # Linux/Mac launcher
│
└── Documentation/
    ├── README.md               # Project overview
    ├── QUICKSTART.md           # Getting started guide
    ├── ARCHITECTURE.md         # System design diagrams
    └── PROJECT_SUMMARY.md      # Detailed feature list
```

Total Lines of Code: ~1,500+ lines (excluding tests and documentation)

🔬 Technical Deep Dive

LSB Steganography Algorithm

Image LSB Process:
```
Original RGB Pixel: (Red=11010110, Green=10110011, Blue=11001100)
Message to hide: "A" = 01000001 (8 bits in binary)

Encoding:
1. Take first 3 bits of message: 010
2. Replace LSBs:
   Red:   11010110 → 11010110 (LSB=0, no change)
   Green: 10110011 → 10110010 (LSB=1 → 0)
   Blue:  11001100 → 11001101 (LSB=0 → 1)
3. Move to next pixel for next 3 bits
4. Continue until message fully embedded

Result: Modified pixel: (Red=214, Green=178, Blue=205)
Difference: Invisible to human eye (changes ≤1 per channel)
```

Audio LSB Process:

```
Original Sample: 0101101011010110 (16-bit signed integer = 23254)
Message bit: 1

Encoding:
1. Clear LSB: 0101101011010110 AND 1111111111111110 = 0101101011010110
2. Set new LSB: 0101101011010110 OR 0000000000000001 = 0101101011010111
3. Result: 23255 (difference of 1 out of 65536 possible values)
4. Inaudible: 0.0015% change in amplitude

Key-Based Enhancement:
- SHA-256 hash of key generates random seed
- Pseudo-random number generator selects positions
- Example: If key="secret", positions might be [17, 423, 891, 1205...]
```

Encryption Flow

```
1. User Input:
   - Message: "Secret data"
   - Password: "mypassword123"

2. Key Derivation (PBKDF2):
   - Salt: Random 16 bytes
   - Iterations: 100,000
   - Output: 32-byte AES key

3. Encryption (AES-256 CBC):
   - Generate random 16-byte IV
   - Encrypt message with key and IV
   - Output: Encrypted bytes

4. Encoding:
   - Format: IV + Salt + Encrypted_Data
   - Base64 encode for text representation
   - Result: "U2FsdGVkX1..." (looks like random text)

5. Embedding:
   - Encrypted text embedded in image/audio using LSB
   - Double protection: hidden + encrypted

6. Decoding:
   - Extract encrypted text from media
   - Base64 decode
   - Split into IV, Salt, Encrypted_Data
   - Derive key from password + salt
   - Decrypt with AES using key + IV
   - Recover original message
```

💻 How to Run the Project

Installation Steps:

1. **Install Python 3.8+**
   ```bash
   python --version  # Verify installation
   ```

2. **Install Dependencies**
   ```bash
   pip install -r requirements.txt
   ```
   This installs:
   - Pillow (image processing)
   - NumPy (numerical operations)
   - SciPy (audio I/O)
   - PyCryptodome (encryption)

3. **Verify Installation**
   ```bash
   python verify_installation.py
   ```
   Checks all dependencies and modules

4. **Run Application**
   ```bash
   python main_gui.py
   ```
   Or double-click `run.bat` (Windows) / `run.sh` (Linux/Mac)

Usage Example:

Hiding a Message in an Image:
1. Launch `main_gui.py`
2. Go to "Image Steganography" tab
3. Click "Select Cover Image" → Choose a PNG file
4. Enter secret message in text box
5. (Optional) Enable encryption and enter password
6. Click "Encode Message" → Save as new PNG
7. Result: Visually identical image with hidden message

Recovering the Message:
1. Click "Decode Message"
2. Select the stego-image (encoded PNG)
3. (If encrypted) Enter the same password
4. Click "Decode" → Message appears in output box

Audio Example:
1. Go to "Audio Steganography" tab
2. Select WAV file
3. Enter message
4. (Optional) Enter embedding key for security
5. Encode → Save modified WAV
6. Decode with same key to retrieve message

🧪 Testing & Validation

Test Coverage:

1. Image Tests (`test_image.py`)
- Basic encoding/decoding
- Large message handling
- Capacity limits
- Format validation (PNG vs JPG)
- Empty message handling
- Unicode character support

2. Audio Tests (`test_audio.py`)
- Sequential embedding
- Key-based embedding
- Wrong key detection
- Capacity limits
- Sample rate variations
- Bit depth handling

3. Utility Tests (`test_utils.py`)
- Encryption/decryption
- Binary conversion accuracy
- File validation
- Error handling

Manual Testing Performed:
- ✅ Image: 800×600 PNG with 100-character message
- ✅ Audio: 60-second WAV with 50-character message
- ✅ Encryption: 256-bit AES with various passwords
- ✅ Key-based: Random position embedding
- ✅ GUI: All buttons, tabs, and dialogs
- ✅ Error handling: Invalid files, wrong keys, oversized messages

---

🎓 Key Concepts Demonstrated

1. Object-Oriented Programming (OOP)
- Classes: `ImageSteganography`, `AudioSteganography`, `EncryptionHelper`
- Encapsulation: Private methods (e.g., `_text_to_binary()`)
- Modularity: Separate modules for different functionalities
- Reusability: Helper functions used across modules

2. Binary Operations
- Bit manipulation (AND, OR, XOR, bit shifting)
- LSB extraction and modification
- Binary-to-decimal conversion
- Byte array handling

3. File I/O Operations
- Reading/writing image files (Pillow)
- Reading/writing audio files (SciPy, Wave)
- Binary file handling
- File validation and error checking

4. Cryptography
- Symmetric encryption (AES-256)
- Key derivation (PBKDF2)
- Cipher modes (CBC)
- Salt and IV generation

5. GUI Development
- Event-driven programming
- Multi-threaded applications (background processing)
- User input validation
- File dialogs and message boxes
- Tab-based interfaces

6. Algorithm Design
- LSB steganography algorithm
- Pseudo-random position generation
- Capacity calculation
- Delimiter-based message termination

7. Error Handling
- Try-except blocks
- Input validation
- Graceful failure messages
- Edge case handling

8. Software Engineering Practices
- Modular code structure
- Documentation and comments
- Unit testing
- Version control (Git)
- Requirements management

📊 Capacity & Performance

Storage Capacity:

Media Type	Example	Capacity	Calculation
Small Image	640×480 PNG	~115 KB	(640×480×3)÷8
HD Image	1920×1080 PNG	~777 KB	(1920×1080×3)÷8
4K Image	3840×2160 PNG	~3.1 MB	(3840×2160×3)÷8
30s Audio	WAV 44.1kHz	~165 KB	(44100×30)÷8
60s Audio	WAV 44.1kHz	~330 KB	(44100×60)÷8
3min Audio	WAV 44.1kHz	~1.98 MB	(44100×180)÷8

Performance:
- Image encoding: < 1 second (1920×1080)
- Audio encoding: < 2 seconds (60-second WAV)
- Encryption overhead: < 0.5 seconds
- GUI response: Instant (with background threading)

🔒 Security Analysis

Security Features:
1. **LSB Steganography**
   - Advantages: Invisible to human perception
   - Limitations: Detectable by statistical analysis tools

2. **AES-256 Encryption**
   - Military-grade encryption standard
   - 2^256 possible keys (practically unbreakable)
   - PBKDF2 protects against brute force

3. **Key-Based Embedding**
   - Adds security through obscurity
   - Without key, message appears as random noise
   - SHA-256 hashing for position generation

Security Considerations:
- ⚠️ LSB data destroyed by re-compression (JPG, MP3)
- ⚠️ Steganalysis tools can detect presence of hidden data
- ⚠️ Use encryption for sensitive information
- ⚠️ Secure key distribution required
- ✅ Good for: Low-risk data hiding, watermarking
- ❌ Not suitable for: High-security military/government use

🌟 Project Highlights

What Makes This Project Special:

1. **Dual-Mode Steganography**
   - Both image and audio support
   - Different techniques optimized for each media type

2. **User-Friendly GUI**
   - No command-line knowledge required
   - Clear visual feedback
   - Simple tabbed interface

3. **Optional Encryption**
   - Double-layer security (hiding + encryption)
   - Industry-standard AES-256

4. **Key-Based Security**
   - Unique feature for audio steganography
   - Random position embedding

5. **Cross-Platform**
   - Works on Windows, Mac, Linux
   - Pure Python implementation

6. **Well-Documented**
   - Extensive comments in code
   - Multiple documentation files
   - Usage examples included

7. **Tested & Validated**
   - Unit tests for all major functions
   - Manual testing performed
   - Error handling throughout

📚 Learning Outcomes

Skills Developed:

Technical Skills:
- Python programming (intermediate to advanced)
- Image processing with Pillow/NumPy
- Audio processing with SciPy
- Cryptography implementation
- GUI development with Tkinter
- Binary operations and bit manipulation
- File I/O operations
- Multi-threading concepts

Computer Science Concepts:
- Steganography algorithms
- LSB substitution technique
- Encryption/decryption
- Hash functions
- Pseudo-random number generation
- Data encoding/decoding

Software Development:
- Object-oriented design
- Modular code architecture
- Error handling and validation
- Unit testing
- Documentation writing
- Version control

Problem Solving:
- Algorithm implementation
- Capacity calculations
- Security considerations

- User experience design

🚀 Future Enhancements (Possible Improvements)

1. **Additional Media Types:**
   - PDF files
   - Text files
   - More image formats (BMP, TIFF)

2. **Advanced Steganography:**
   - DCT-based methods (more robust)
   - Spread spectrum techniques
   - Adaptive LSB (varies by image content)

3. **Enhanced Security:**
   - Public-key cryptography (RSA)
   - Digital signatures
   - Stealth mode (anti-steganalysis)

4. **User Interface:**
   - Drag-and-drop file selection
   - Batch processing
   - Preview before/after
   - Capacity visualization

5. **Additional Features:**
   - File compression before embedding
   - Password strength meter
   - Embedding history
   - Export/import settings

📖 References & Resources

Steganography Theory:
- LSB Steganography: Replacing least significant bits to hide data
- Cover media: Original file used for hiding
- Stego media: Modified file containing hidden data
- Payload: The secret message being hidden

Libraries Documentation:
- **Pillow:** https://pillow.readthedocs.io/
- **NumPy:** https://numpy.org/doc/
- **SciPy:** https://docs.scipy.org/
- **PyCryptodome:** https://pycryptodome.readthedocs.io/
- **Tkinter:** https://docs.python.org/3/library/tkinter.html

Algorithms:
- AES (Advanced Encryption Standard) - NIST FIPS 197
- PBKDF2 (Password-Based Key Derivation Function 2) - RFC 2898
- SHA-256 (Secure Hash Algorithm) - NIST FIPS 180-4

🎤 Presentation Tips

Key Points to Emphasize:

1. **Problem Statement:**
   - Need for covert communication
   - Hiding data in plain sight
   - Digital watermarking applications

2. **Solution:**
   - LSB steganography technique
   - User-friendly desktop application
   - Multiple media type support

3. **Technical Achievement:**
   - Binary manipulation at bit level
   - Cryptographic security integration
   - Responsive GUI with threading

4. **Demonstration:**
   - Live encoding/decoding demo
   - Show visual similarity of images
   - Prove message recovery works

5. **Challenges Overcome:**
   - Understanding binary operations
   - Audio sample manipulation
   - GUI responsiveness with threading

6. **Results:**
   - Functional application
   - ~1,500 lines of code
   - Comprehensive testing
   - Complete documentation

Demo Script:
1. Show original image/audio
2. Encode message with your name
3. Compare original vs stego (looks identical)
4. Decode to recover message
5. Show encryption feature (optional)
6. Explain capacity limits

✅ Project Checklist

- ✅ Functional image steganography
- ✅ Functional audio steganography
- ✅ AES-256 encryption support

- ✅ Key-based embedding
- ✅ Graphical user interface
- ✅ Error handling
- ✅ Input validation
- ✅ Unit tests
- ✅ Documentation
- ✅ Cross-platform compatibility
- ✅ Code comments
- ✅ Example scripts
- ✅ Installation guide

📞 Project Summary

In One Sentence:
A Python desktop application that uses LSB steganography to hide encrypted messages inside PNG images and WAV audio files with a user-friendly GUI.

Core Technologies:
Python 3.12, Pillow (images), SciPy (audio), PyCryptodome (encryption), Tkinter (GUI), NumPy (binary ops)

Lines of Code: ~1,500+

Key Features: Image/Audio LSB embedding, AES-256 encryption, key-based randomization, capacity calculation, tabbed GUI

Time Investment: Multiple weeks of development and testing

Difficulty Level: Intermediate to Advanced Python

End of Documentation
Prepared for teacher presentation by Paula, November 2025