File Functionalities

File	Purpose / Functionality
experiments.py	Main script to run experiments. Handles message input, PQC encryption, chunk splitting, RL allocation, embedding into images/audio, extraction, recombination, and decryption.
utils.py	Contains utility functions for: - Post-Quantum Encryption (Kyber512) <pq_encrypt> / <pq_decrypt> - Optional AES encryption <aes_encrypt> / <aes_decrypt> - Chunk fragmentation <split_chunks> / <recombine_chunks> - Placeholder RL allocation <rl_allocate_chunks>
stego_image.py	Implements **image steganography** using LSB technique: - <embed_image_lsb>: Embed a byte chunk into an image. - <extract_image_lsb>: Extract chunk from an image.
stego_audio.py	Implements **audio steganography** using LSB technique: - <embed_audio_lsb>: Embed a byte chunk into a WAV audio file. - <extract_audio_lsb>: Extract chunk from a WAV audio file.
stego_manager.py	Higher-level interface to manage multiple carriers and coordinate embedding/extraction across images and audio files.
pqc.py	Wrapper functions for post-quantum encryption using Kyber: - Handles encryption/decryption operations for messages and returns ciphertext/shared keys.
fragmenter.py	Handles splitting and recombining data chunks for embedding.
rl_agent.py	Implements the **reinforcement learning agent**: - Learns optimal chunk allocation strategy. - Interacts with multi_carrier_env.py to select best carriers based on reward.
metrics.py	Computes metrics to evaluate embedding quality and RL agent performance, e.g., distortion, payload distribution, or recovery success.
multi_carrier_env.py	Custom **RL environment** for multi-carrier allocation: - Defines **state**, **action**, **reward** for chunk allocation. - Simulates embedding into multiple images/audio files. - Works with rl_agent.py to train allocation policies before actual embedding.
envs/	Folder containing environment definitions. Currently contains multi_carrier_env.py.

End-to-end pipeline

Step 0: Setup & Environment

- **envs/**
  - Contains environment definitions for RL.
  - Currently has multi_carrier_env.py, which simulates multiple carriers for chunk allocation.

Step 1: Input Message

- **experiments.py**
  - Prompts the user for a secret message.
  - Converts the message into bytes for encryption.

Step 2: Post-Quantum Encryption

- **utils.py / pqc.py**
  - pq_encrypt(message: bytes) (in utils/pqc)
    - Encrypts the message using **Kyber512** key encapsulation.
    - Returns ciphertext, shared secret, public key, and secret key.
  - pq_decrypt(ciphertext, secret_key)
    - Decrypts the ciphertext using the secret key to recover the original message.
  - Optional AES encryption functions (aes_encrypt, aes_decrypt) are also here, used if you want an extra symmetric layer on the message.

Flow:
experiments.py -> utils.pq_encrypt -> Kyber encryption -> ciphertext returned

Step 3: Chunking the Ciphertext

- **utils.py / fragmenter.py**
  - split_chunks(ciphertext, chunk_size)

        - Splits ciphertext into fixed-size byte chunks (e.g., 16 bytes).
    - recombine_chunks(chunks)
        - Reassembles chunks back into the original ciphertext after extraction.

Flow:
ciphertext -> split_chunks -> list of chunks

Step 4: Load Carrier Files

- **experiments.py**
    - Loads all images (.png) and audio (.wav) from data/images and data/audio.
    - These files act as carriers for the steganography process.

Step 5: Allocate Chunks (RL Placeholder)

- **utils.py / rl_agent.py / multi_carrier_env.py**
    - rl_allocate_chunks(chunks, image_paths, audio_paths)
        - Currently **round robin allocation but** designed to be replaced by RL.
    - multi_carrier_env.py
        - Defines the RL environment: states, actions, rewards for allocating chunks to carriers.
    - rl_agent.py
        - Contains logic for training RL policies to optimize chunk placement.

Flow:
chunks + carrier files -> RL allocation -> mapping of chunk -> carrier

Step 6: Embed Chunks into Media

- **stego_image.py**
    - embed_image_lsb(image_path, output_path, chunk)
        - Embeds a byte chunk into an image using LSB steganography.

    - extract_image_lsb(image_path, chunk_size)
        - Extracts chunk from the stego image.
- **stego_audio.py**
    - embed_audio_lsb(audio_path, output_path, chunk)
        - Embeds a byte chunk into a WAV audio file.
    - extract_audio_lsb(audio_path, chunk_size)
        - Extracts chunk from the stego audio.
- **stego_manager.py**
    - Coordinates embedding/extraction across multiple carriers (images + audio).

Flow:
allocation -> embed_image_lsb / embed_audio_lsb -> stego files created

Step 7: Extract Chunks & Recombine

- **experiments.py** + **stego_image.py / stego_audio.py / fragmenter.py**
    - Extracts chunks from all carriers.
    - Uses recombine_chunks to reconstruct the full ciphertext.

Flow:
stego files -> extract chunks -> recombine -> recovered ciphertext

Step 8: Decrypt Message

- **utils.py** / **pqc.py**
    - pq_decrypt(recovered_ciphertext, secret_key)
        - Converts the ciphertext back into the original message.

Flow:
recovered ciphertext -> pq_decrypt -> original message

Step 9: Metrics / Evaluation

- **metrics.py**
    - Optional: evaluates embedding quality, distortion, and recovery success.

- Can be used to provide feedback for RL agent training.

Flow:
original chunks + stego files -> metrics.py -> evaluation scores

Full Flow Summary

1. **Input message** → experiments.py
2. **Encrypt message** → pqc.py / utils.py
3. **Split ciphertext** → fragmenter.py / utils.py
4. **Load carriers** → experiments.py
5. **Allocate chunks** → utils.py (RL placeholder) / rl_agent.py / multi_carrier_env.py
6. **Embed chunks** → stego_image.py / stego_audio.py / stego_manager.py
7. **Extract & recombine** → stego_image.py / stego_audio.py / fragmenter.py
8. **Decrypt message** → pqc.py / utils.py
9. **Compute metrics** → metrics.py