Papers Downloaded from ACM and their research findings

Paper 1: P2 (Encryption Standard)

Title: "Content-Aware Selective Encryption for H.265/HEVC Using Deep Hashing Network and Steganography"

Source: ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), Vol 21, Issue 1, Jan 2025.

- What they did (The 2025 Standard):

They argue that encrypting all data is inefficient, but simple rules (like "encrypt every 10th block") are insecure. They proposed "Content-Aware Selection". They used a hashing network to measure the "sensitivity" of a data block. If a block contains sensitive features (high importance), it gets heavy encryption. If it's background noise, it gets light encryption.

- The Research Gap in Your Code:

Your FileEncryptor.java uses Length-Based Selection (if length <= 2). This is "Content-Agnostic." It ignores the meaning or importance of the word. A trivial word like "is" and a code "X9" might both be short, but one is critical and the other is not.

- ✅ **The Upgrade to Implement: "Entropy-Based Cipher Selection"**
    - **Goal:** Make your encryption "Content-Aware" like P2.
    - **Implementation:** instead of word.length(), you will calculate **Shannon Entropy**.
    - **New Logic:**
        - Entropy > threshold (Complex/Random string) ->**AES-256** (Sensitive).
        - Entropy < threshold (Standard English) -> **Vigenère** (Non-Sensitive).
    - **Result:** You align with P2's methodology of "Selective Encryption based on Feature Importance."

Paper 2: P4 (Steganography Standard)

Title: "Enhancing Adversarial Embedding based Image Steganography via Clustering Modification Directions"

Source: ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), Vol 20, Issue 1, Jan 2024.

- What they did (The 2025 Standard):

They addressed the problem of "Embedding Clustering." When you hide data pixels next to each other sequentially, it creates a "cluster" of artifacts that AI detectors (Steganalysis) can

easily spot. They proposed spreading the changes out or "clustering" them in specific high-texture directions where they are invisible.

- The Research Gap in Your Code:

Your ImageStego.java embeds data Sequentially (block[0][0], block[0][1]...). This creates a predictable "trail" of modified blocks that AI models (like SRNet) can detect easily because the modifications are clumped together.

- ✅ **The Upgrade to Implement: "Variance-Based Sparse Embedding"**
    - **Goal:** Break the "clustering" of artifacts like P4.
    - **Implementation:** Before embedding in a block, calculate the **Pixel Variance** (Texture/Roughness).
    - **New Logic:**
        - If Variance < Threshold (Smooth area, Sky) -> **SKIP** the block.
        - If Variance > Threshold (Textured area, Grass) -> **EMBED**.
    - **Result:** This forces the data to "scatter" (become sparse) across the image, hiding only in the noisy parts where detection is hardest.

Paper 3: P8 (The Steganography Robustness Standard)

Title: *"SparSamp: Efficient Provably Secure Steganography Based on Sparse Sampling"*
Source: *USENIX Security Symposium* (Evaluated Artifact Available), **2024/2025 Cycle**.

- **What they did (The 2025 Standard):** They attacked the concept of "Dense Embedding." If you hide data in *every* possible slot (e.g., every 8x8 block), you alter the global statistical distribution of the image, making it trivial for an AI to say "This image is mathematically too perfect/noisy." They proved that **Sparse Sampling** (skipping blocks based on a pseudo-random interval) preserves the original distribution, making the message statistically undetectable.
- **The Research Gap in Your Code:** Your ImageStego.java currently runs a nested loop: for(x=0; x<width; x+=N). This is **Dense & Sequential**. You are filling blocks 1, 2, 3, 4... in a straight line. This creates a "stego-snake" artifact that P8 explicitly warns against.
- ✅ **The Upgrade to Implement: "Randomized Interval Embedding (Sparse Mode)"**
    - **Goal:** Break the sequential pattern.
    - **Implementation:** Instead of x += 8, you use a shared **Random Seed** (derived from the Vigenère Key).
    - **New Logic:**

1. Seed = Hash(VigenèreKey)
2. Interval = Random(Seed) (e.g., next block is 3 hops away, then 1 hop, then 5 hops).
3. **Embed Data**.

- **Result:** The message is scattered randomly across the image. Even if an attacker analyzes the first 100 blocks, they find nothing because the data might start at block 500.

Paper 4: P6 (The Crypto-Architecture Standard)

Title: *"Testing Side-channel Security of Cryptographic Implementations against Future Microarchitectures"* **Source:** *ACM CCS 2024* (Proceedings of the 2024 ACM SIGSAC Conference).

- **What they did (The 2025 Standard):** This paper is a wake-up call for "Hardcoded Crypto." They demonstrated that even if an algorithm (like Kyber) is mathematically secure, specific CPU architectures (Intel vs. AMD) leak data via "Side Channels" (timing/power) if the implementation isn't tuned. They mandate **"Crypto-Agility"**—the system *must* be able to swap underlying algorithms if a specific implementation is found to be leaky on the deployment hardware.
- **The Research Gap in Your Code:** Your LatticeManager.java has this line: keyGen.initialize(KyberParameterSpec.kyber768);. **This is "Protocol Ossification."** You are married to Kyber-768. If Paper P6 reveals that Kyber-768 has a side-channel leak on your specific laptop processor, your entire project is defenseless.
- ✅ **The Upgrade to Implement: "Configuration-Driven Crypto-Agility"**
  - **Goal:** Decouple the "Logic" from the "Algorithm."
  - **Implementation:** Create a config.properties file.
  - **New Logic:**
    1. Read PQC_ALGO from config.
    2. If config == "KYBER1024", load KyberParameterSpec.kyber1024.
    3. If config == "DILITHIUM", load Dilithium.
  - **Result:** You can claim in your report: *"In response to side-channel risks identified in ACM CCS 2024, we architected a modular crypto-provider that allows hot-swapping of PQC primitives without code refactoring."*

Paper 5: P7 (The Architecture Standard)

Title: "Split Unlearning"

Source: ACM CCS 2025 (Proceedings of the 2025 ACM SIGSAC Conference).

- What they did (The 2025 Standard):

This paper addresses privacy in AI by "Splitting" the learning process. Instead of keeping all data on one server (centralized risk), they split the model and data between clients and servers. This ensures that if one node is compromised or needs to "unlearn" data, it can be done without affecting the whole system. The core concept is "Security via Partitioning."

- The Research Gap in Your Code:

Your current project is Monolithic. You take the whole message, encrypt it, and hide it linearly in one image.

  - *Risk:* If the attacker cracks that one specific image (or the header), they get **100%** of the data. There is no "Partitioning" or "Fault Tolerance."

- ✅ **The Upgrade to Implement: "Shamir's Secret Sharing (Split-Payload Steganography)"**
  - **Goal:** Apply the "Split" philosophy of P7 to your hidden message.
  - **Implementation:** Instead of embedding the message directly, you first split it into N "shares" using **Shamir's Secret Sharing Scheme (SSSS)**.
  - **New Logic:**
    1. Split Message M into 5 parts.
    2. You only need *any 3 parts* to recover the message.
    3. Embed Part 1 in the Red Channel, Part 2 in Green, Part 3 in Blue (or in different frequency bands).
  - **Why DRDO needs this:** This adds **Fault Tolerance**. If the image is compressed and the "Blue" channel data is lost, the receiver can still reconstruct the message using the Red and Green channels. This is military-grade reliability.

The Final "Novel Implementation" for DRDO

Title: *"Quantum-Resistant Split-Steganography with Entropy-Adaptive Sparse Masking"*

The Workflow (How the code will run):

1. **Step 1: Intelligent Analysis (P2)**
    - The system scans the input text.
    - It calculates **Shannon Entropy**.
    - *High Entropy (Coordinates)* -> Flagged for **AES-256**.
    - *Low Entropy (Chatter)* -> Flagged for **Vigenère**.
2. **Step 2: Distributed Security (P7 & P6)**
    - The encrypted data is **Split** into 3 Shares (using Shamir's Secret Sharing).
    - The Session Keys are encapsulated using **Agile PQC** (Kyber or Dilithium, loaded via Config).
3. **Step 3: Adaptive Embedding (P4 & P8)**
    - The system scans the image blocks.
    - **Variance Check (P4):** Is this block "Rough" enough? If no -> Skip.
    - **Sparse Check (P8):** Is this the randomized "Hop" interval? If no -> Skip.
    - Only when *both* checks pass, the system embeds **one share** of the data.

Why this is "Novel" & "Gap-Filling":

- **Novelty:** Most steganography is either "Secure" (Crypto) or "Robust" (Stego). Yours is **both**, plus it is **Fault Tolerant** (due to P7's split logic) and **Quantum-Safe** (P6).
- **Research Gap Filled:** You have moved from "Static, Linear, Monolithic" code to "Adaptive, Distributed, Agile" architecture.

FILE UPDATES

1. Target File: FileEncryptor.java

Feature to Implement: Entropy-Based Cipher Selection

- **Reference Paper:** *P2 (ACM TOMM 2025) - "Content-Aware Selective Encryption"*
- **The Upgrade:** Replace the "naive" length-based check with a statistical entropy check to decide "Importance."
- **Coding Tasks:**
    1. **Add Helper Method:** Create public static double calculateEntropy(String text) that returns the Shannon Entropy score.
    2. **Update Logic:**
        - *Old:* if (word.length() <= 2)
        - *New:* if (calculateEntropy(word) > threshold (Find out about it)) -> **AES-256** (High Security).
        - *Else:* **Vigenère** (Standard Security).

2. Target File: ImageStego.java

Feature A: Texture-Adaptive Masking (The "Invisibility" Layer)

- **Reference Paper:** *P4 (ACM TOMM 2024) - "Enhancing Adversarial Embedding"*
- **The Upgrade:** Stop embedding in "Smooth" areas (like blue sky) where artifacts are visible.
- **Coding Tasks:**
    1. **Add Helper Method:** Create private static double getBlockVariance(double[][] dctBlock).
    2. **Update Loop:** Inside the nested x, y loop, add a check:
        - if (getBlockVariance(dct) < THRESHOLD) continue; (Skip smooth blocks).

Feature B: Sparse Randomized Sampling (The "Anti-AI" Layer)

- **Reference Paper:** *P8 (USENIX Security 2025) - "SparSamp"*
- **The Upgrade:** Stop embedding sequentially (Block 1, 2, 3...). Hop around the image randomly to break statistical patterns.
- **Coding Tasks:**

1. **Add PRNG:** Initialize a Random object seeded with the **Vigenère Key** (so the receiver can replicate the sequence).
2. **Update Loop:** Instead of x += 8, use x += (8 * (rand.nextInt(3) + 1)). This creates random "hops" between data points.

3. Target File: LatticeManager.java

Feature to Implement: Configuration-Driven Crypto-Agility

- **Reference Paper:** *P6 (ACM CCS 2024) - "Testing Side-channel Security"*
- **The Upgrade:** Remove hardcoded algorithms to prevent "Protocol Ossification."
- **Coding Tasks:**
    1. **Create Config:** Add a config.properties file with PQC_ALGORITHM=Kyber768.
    2. **Update KeyGen:** Modify generateLatticeKeyPair() to read this config.
        - If config is "Kyber1024", load KyberParameterSpec.kyber1024.
        - If config is "Dilithium", load the Dilithium spec.

4. Target File: Main.java

Feature to Implement: Split-Payload Orchestration

- **Reference Paper:** *P7 (ACM CCS 2025) - "Split Unlearning"*
- **The Upgrade:** Avoid a "Monolithic" payload. Split the data for fault tolerance.
- **Coding Tasks:**
    1. **Payload Splitting:** Before calling ImageStego.encode, split the encrypted string into **3 Logic Chunks** (e.g., Header, Body, Metadata).
    2. **Distributed Calls:**
        - Embed Chunk 1 in **Red Channel** (or low freq).
        - Embed Chunk 2 in **Green Channel** (or mid freq).
        - Embed Chunk 3 in Blue Channel (or high freq).

(Note: This requires a small update to ImageStego to accept a "Channel" parameter).

WORK DISTRIBUTION FOR PUNAV AND GOKUL

Punav (The Core Implementer)

Role: Implement the "Adaptive & Agile" Defense System.

Goal: Upgrade the codebase from "Static" to "Dynamic."

1. Feature: Entropy-Based Adaptive Logic (Ref: Paper P2)

- **Encryption Task (FileEncryptor.java):**
    - **Implement calculateEntropy(String).**

  Update the encryption logic:

    - **If Entropy > threshold (Find the right threshold)** -> Use **AES-256**.
    - **If Entropy < threshold** (e.g., "the", "attack"): Use **Vigenère**.

Decryption Task (FileDecryptor.java):

- **The Upgrade: The Receiver cannot calculate entropy of the *plaintext* (because they don't have it yet!).**
- **New Logic: You must distinguish based on Ciphertext Format.**
- ***Why:* AES always outputs fixed block sizes (padding), while Vigenère preserves the original short length. This implicitly solves the "Adaptive" problem.**

2. Feature: Texture-Adaptive Masking (Ref: Paper P4)

- **Encryption Task (ImageStego.java - encode):**
    - Implement getBlockVariance().
    - Loop through blocks: If Variance < Threshold (Smooth) -> SKIP. Else -> EMBED.
- **Decryption Task (ImageStego.java - decode):**
    - CRITICAL: The Receiver must run the exact same check.
    - Loop through blocks: Calculate Variance of the *Stego Block*.
    - If Variance < Threshold -> SKIP READING (Assume no data is there).
    - If Variance > Threshold -> EXTRACT DATA.
    - **Note:** Ensure the Threshold is forgiving enough that the embedded data doesn't push a block from "Smooth" to "Rough" (though usually, embedding increases roughness, so it's safe).

3. Feature: Sparse Randomized Embedding (Ref: Paper P8)

- **Encryption Task (ImageStego.java - encode):**
    - Seed a PRNG with the VigenereKey.

    - Calculate nextHop = random.nextInt().
    - Jump nextHop blocks before embedding.
- **Decryption Task (ImageStego.java - decode):**
    - Synchronization is Key.
    - Initialize the PRNG with the same VigenereKey (recovered from keys.enc).
    - Calculate nextHop = random.nextInt().
    - Jump that many blocks before reading.
    - ***Warning:*** If you read 1 wrong block, the whole message breaks. The sequence must be identical.

4. Feature: Split-Payload Reassembly (Ref: Paper P7)

- **Encryption Task (Main.java):**
    - Split ciphertext into 3 chunks. Embed in Red, Green, Blue channels.
- **Decryption Task (ReceiverMain.java):**
    - Call ImageStego.decode(..., RED_CHANNEL).
    - Call ImageStego.decode(..., GREEN_CHANNEL).
    - Call ImageStego.decode(..., BLUE_CHANNEL).
    - Concatenate string 1 + 2 + 3 to get the full ciphertext.

5. Feature: Crypto-Agility Configuration (Ref: Paper P6)

- **The Task:** Remove hardcoded "Kyber-768" to prevent "Protocol Ossification."
- **Action Items:**
    - Create a config.properties file: PQC_ALGO=Kyber768.
    - **Update LatticeManager.java** to read this file. If the config changes to Dilithium, the code should load the Dilithium parameter spec dynamically.

IMPORTANT FOR PUNAV: For every if (condition) you add in the Encoder, you must add the exact same if (condition) in the Decoder.

1. Texture Masking: If the *Sender* skips a smooth block, the *Receiver* must also calculate variance and skip that same block.
2. Sparse Hopping: The *Receiver* must initialize the Random Number Generator with the exact same seed (the recovered Vigenère Key) to replicate the 'hops' correctly. If these are not perfectly mirrored, the decryption will fail."

Gokul (The AI Validator)

Role: Steganalysis & Adversarial Validation.

Goal: Prove that Punav's new system defeats modern AI detection.

1. Feature: The "New" Data Collection Pipeline

- **The Task:** Generate the dataset specifically to test **Punav's new features**.
- **Action Items:**
    - **Dataset A (Baseline):** Use the *Old* ImageStego (Sequential embedding) to generate 500 images. **Label: Old_Method.**
    - **Dataset B (Novel):** Use Punav's *New* ImageStego (Adaptive/Sparse) to generate 500 images. **Label: New_Method.**
    - ***Why:*** You need to compare them to show improvement.

2. Feature: Adversarial Model Training

- **The Task:** Train a CNN (Steganalysis Model) to detect hidden data.
- **Action Items:**
    - Use a standard architecture like **Xu-Net** or **SRNet** (Python/TensorFlow).
    - Train it first on **Dataset A**. It should achieve **High Accuracy (~90%)** (detecting the old method easily).
    - *Key Research Step:* Now, test that *same* model on **Dataset B**.

3. Feature: The "Proof of Success" (The Graph)

- **The Task:** Generate the specific graph for the final report.
- **Action Items:**
    - Run the model on Punav's new images.
    - **The Goal:** The model accuracy should **DROP** significantly (e.g., to ~50%, which is random guessing).
    - **Why:** This proves that **"Sparse Sampling + Texture Masking"** successfully fooled the AI. This is your project's "Result."