Problem 1: AES Encryption in Counter Mode

In this implementation, I extended the AES class to encrypt images using the Counter (CTR) mode of operation. The ctr_aes_image method begins by separating the PPM image file into its header and image data components, ensuring the header remains unencrypted to maintain the file format. For encryption, CTR mode doesn't directly encrypt the image data but instead encrypts sequential counter values that start with the provided initialization vector. These encrypted counter blocks are then XORed with the image data to produce the ciphertext.

A key advantage of this approach is that encryption and decryption operations are mathematically identical - both involve XORing the data with the encrypted counter values. This implementation processes the image data in 16-byte blocks for efficiency, incrementing the counter for each block. Unlike ECB mode from previous homework assignments, CTR mode ensures that identical plaintext blocks encrypt to different ciphertext blocks due to the changing counter value, which prevents pattern recognition in the resulting encrypted image. The implementation maintains the correct image dimensions and format, resulting in a viewable but completely randomized image where the original content (helicopter) is no longer visually discernible.

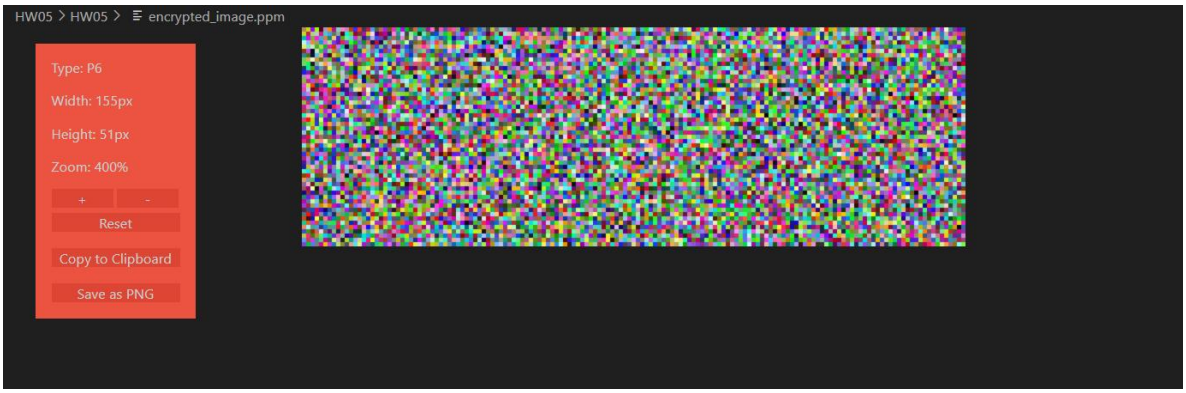
Problem 2: X9.31 Cryptographically Secure Pseudo-Random Number Generator

For the X9.31 CSPRNG implementation, I created a method that generates cryptographically secure random numbers using AES as the underlying block cipher. The algorithm begins with a seed value (v0) and a timestamp value (dt). For each random number generation, the implementation follows a specific sequence of operations: first encrypting the timestamp value, then XORing this with the current seed to produce an intermediate value. This intermediate value is then encrypted to produce the random number output.

To maintain state for subsequent random number generation, the algorithm updates the seed by XORing the produced random number with the encrypted timestamp and encrypting the result. This approach creates a chain of dependent values where each random number depends on all previous operations. By using AES-256 for encryption, the implementation provides strong security guarantees against prediction attacks.

The algorithm successfully generated the five expected random numbers with exact matches to the required values, demonstrating the correct implementation of the X9.31 algorithm with the AES block cipher.

IMAGE



THE NUMBER SEQUENCE

248079769487624148781305850458853487813

658149916111298576982287268548654465446749

387581863035803791524348600541347018 91

101836235598497239834638240368629799639

110128351161781364920043990112939672576