# Secure Data Transfer Over Internet Using Image Steganography

This project is a Java program that is focused on implementing form of steganography using the Least Significant Bit (LSB) method for hiding a message within an image file with the purpose of secure data transfer over internet. It includes methods for ecrypting the text, embedding the text into the image, extracting the text from the resulting picture and decrypting it.

## Research:

• Steganography is a technique of hiding a message or data within another non-secret message or data (image, audio file, or video). Cryptography makes a message unreadable by encoding it, steganography hides the existence of the message.

• If we want to transfer data over the internet securely using image steganography, that means that we need to embed sensitive information into a digital image in such a way that the presence of the hidden data is not detectable to unauthorized users.

• Even if the image is hacked by a hacker, we wouldn't want him to be able to read the data. For that purpose, we need to encrypt the data before embedding it using a cryptographic algorithm. [1]

• The encrypted data needs to be embedded into the pixels of the image using steganographic techniques. The resulting image appears identical or very similar to the original image to the human eye, but it contains the encrypted data hidden within its pixels.

• Upon receiving the steganographically altered image, the recipient can extract the hidden data. But the recipient also needs to know the encryption key to decrypt the data after extracting it.

• The hidden data should remain intact and recoverable even if the cover medium undergoes compression, cropping, or noise addition.

### Types and techniques of image steganography:

Steganographic techniques that modify image files for hiding information are:

• Spatial domain;

- JPEG compression
- JPEG Steganography
- Wavelet transform method
- Least Significant Bit (LSB) embedding falls under the category of spatial domain steganographic techniques.

• Transform domain;

• Spread spectrum;

• Statistical techniques; and

• Distortion techniques.

Steganographic techniques that modify the image file format involve:

• Fle embedding and

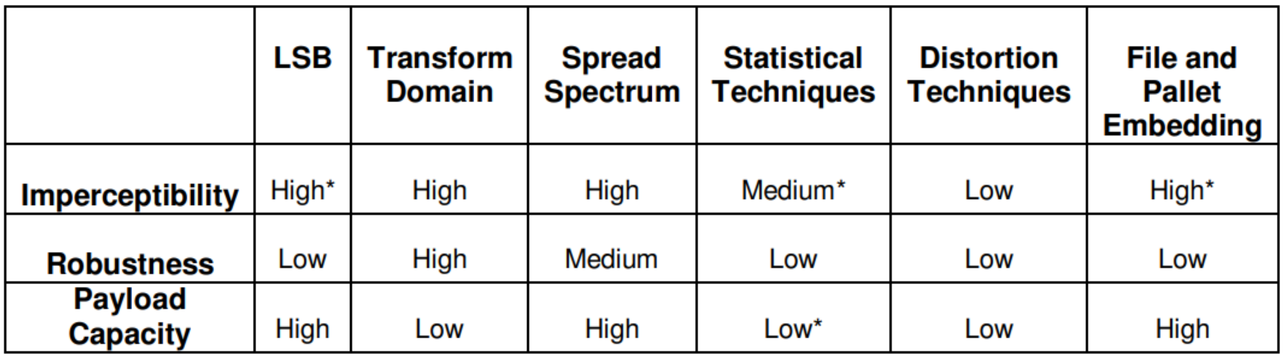
• Palette embedding.

Techniques that modify the elements in the visual image:

• Image generation technique and

• Image element modification technique.

There is also a special type of the spatial and transform domain techniques called the adaptive steganography technique. [2]



| | LSB | Transform Domain | Spread Spectrum | Statistical Techniques | Distortion Techniques | File and Pallet Embedding |
|---|---|---|---|---|---|---|
| **Imperceptibility** | High* | High | High | Medium* | Low | High* |
| **Robustness** | Low | High | Medium | Low | Low | Low |
| **Payload Capacity** | High | Low | High | Low* | Low | High |

*Table from: https://www.cscjournals.org/manuscript/Journals/IJCSS/Volume6/Issue3/IJCSS-670.pdf*

## Least Significant Bit (LSB):

LSB (Least Significant Bit) image steganography – we change the least significant bit of a pixel value, so that it has minimal impact on the image's visual appearance. LSB steganography is one of the simplest methods for hiding information within digital media, such as images.

Each pixel is defined by 3 channels: Red, Green, Blue (RGB), each of them represented by 1 byte. If we change the last one bit of the pixel (the Least Significant Bit), the two colors will be pretty similar. Even if they're joined, we won't be able to see the difference, meaning the final image will look much similar to the original one.

If we want to encode a single letter, we need to get the decimal value for the letter from the ASCII table and convert it into binary form. The binary form of letters has 8 bits, and we can store 3 bits per pixel, meaning to encode one letter in a picture, we will need 3 pixels, only changing the last bit. [3]

## Sources:

•[1] https://nevonprojects.com/secure-data-transfer-over-internet-using-image-steganography/#:~:text=What%20this%20system%20does%20is,be%20able%20to%20read%20the

• [2] https://www.cscjournals.org/manuscript/Journals/IJCSS/Volume6/Issue3/IJCSS-670.pdf

• [3] https://medium.com/@renantkn/lsb-steganography-hiding-a-message-in-the-pixels-of-an-image-4722a8567046

## About the project:

This project consists of the following:

• A **Java program** with the following components:

- **User input**: The user can choose if they want to embed a secret message into an image or if they want to read a secret message from an image. They can write a secret message which they would like to embedd into an image, the location path of the image which they would like to use and the destination path of the image which will contain the embedded message. They can also write the secret key and the cipher algorithm if they choose to read a secret message from an image.
- **Data Encryption**: After the user writes the message that they would like to hide, the message is encrypted using symmetric encryption (AES) and encoded using Base64. This step ensures security so that even if someone is able to extract the hidden data from the image, they won't be able to read it.
- **Data Embedding**: After encrypting the data, the encrypted data is embedded into the pixels of the image using LSB method.
- **Data Extraction**: The user can also extract a secret message from an image, but after extraction the message is in encrypted form. It needs to be decrypted for the user to be able to read it.
- **Data Decryption**: After extraction the next step is decryption, but to do that, the user needs to know the secret key which was used to encrypt the data. After providing the secret key and the cipher algorithm used, the user is now able to read the secret message which is displayed in the console.

• **Documentation**: The documentation is the file that you are currently reading. It includes the research that I've done, how the project works, how to use it and sources I have used to create it.

I **tested** the project if it would work if the image with the secret message was transmitted via mail. After sending the image through mail, I downloaded it and I attempted to extract the message and was able to do that without any loss or corruption. I also tried compressing the image and decompressing it and was again able to extract the message without any loss or corruption.

## Programming Language and Libraries:

Programming language: **Java**, JDK version 17

IDE used: IntelliJ IDEA

Libraries:

1. **Java Standard Library**.
2. **javax.imageio**: API for reading and writing images in various formats.
3. **java.awt**: Classes for creating and managing graphical user interface components.
4. **java.io**: This package provides classes for input and output operations.
5. **java.util.ArrayList**
6. **java.lang**

7. **javax.crypto.Cipher**: Functionality for encryption and decryption using cryptographic algorithms.
8. **javax.crypto.KeyGenerator**: This class is used to generate secret keys for symmetric encryption algorithms (AES).
9. **javax.crypto.SecretKey**: This interface represents a secret symmetric key used for encryption and decryption.
10. **java.nio.charset.StandardCharsets**: This class provides constants for charset names defined by the IANA character set registry.
11. **java.security.Signature**: This class provides the functionality for generating and verifying digital signatures.
12. **java.util.Base64**: This class provides methods for encoding and decoding data in Base64 format.
13. **java.util.Scanner**

## Installation and starting the project:

To set up the project, follow these steps:

1.Clone the project repository from GitHub: https://github.com/MonikaJov/Image-Steganography-using-LSB

2.Open the file as IntelliJ IDEA project

3.Run the project

3.Write in console

## System Architecture:

### class Main:

When we run the main program, we are given a choice whether we want to write a hidden message to an image or read a hidden message from an image.

If we chose that we want to write a hidden message to an image, then we need to enter our message in console, the image location and the destination path. Then the method Generate_image is called where the message is encrypted with the encrypt method from the TextEncryptor class and then the method Embed from the EmbedLSB class is called, which embeds a message into the pixels of the image.

If we chose that we want to read a hidden message from an image, then we need to enter our secret keys in console, the image location. Then the method Extract from the ExtractLSB class is called which extracts the hidden message and then decrypts it using the decrypt method from the TextDecryptor class.

## class EmbedLSB:

This class embeds a message into an image using LSB steganography, where the message is hidden within the least significant bits of the images pixel data.

- **Embed:** Is the entry point for embedding a message into an image. It takes three parameters: the original image file, the message to be embedded, and the file path for the new image with the embedded message.
- **GetImage**: Creates a copy of the original image.
- **GetPixelArray**: Extracts the pixel data from the buffered image into an array of Pixel objects.
- **ConvertMessageToBinary**: Converts the original message in binary format.
- **ConvertAsciiToBinary**: Converts ASCII values to binary strings.
- **LeftPadZeros**: Pads binary strings to ensure they are 8 bits long.
- **EncodeMessageBinaryInPixels**: Embeds the binary message into the pixel data.
- **ChangePixelsColor**: Modifies the RGB values of pixels to encode the message.
- **GetPixelsRGBBinary**: Converts pixel RGB values to binary strings.
- **ChangePixelBinary**: Modifies the least significant bit of a binary string.
- **GetNewPixelColor**: Constructs a new Color object from modified RGB binary strings.
- **ReplacePixelsInNewBufferedImage**: Replaces original pixels in a new buffered image with modified pixels.
- **SaveNewFile**: Saves the new image.

## class ExtractLSB:

This class to extracts the hidden message from an image that has been encoded using the LSB (least significant bit) steganography method.

- **Extract**: This method is the entry point for extracting a message from an image. It takes three parameters: the file path of the image containing the hidden message, a secret key, and a cipher object for decryption.
- **GetPixelArray**: Extracts the pixel data from the buffered image into an array of Pixel objects, similar to the Embed method in the EmbedLSB class.
- **ExtractMessageFromPixels**: Extracts the hidden message from the least significant bit of the RGB values of the pixels.
- **ConvertPixelsToCharacter**: Converts groups of three pixels into a character by extracting the least significant bit of their RGB values.
- **TurnPixelIntegersToBinary**: Converts the RGB integer values of a pixel into binary strings.

- **IsEndOfMessage**: Checks if the end of the message has been reached.
- **ConvertBinaryValuesToCharacter**: Converts a list of binary strings into a character.
- **RemovePaddedZeros**: Removes any padding from the binary representation of a character.

## class TextEncryptor:

This class encrypts text using symmetric encryption (AES) and encode the encrypted data using Base64.

- **generateAESKey**: This method generates a secret key for AES encryption.
- **getCipher**: This method gets an instance of the AES cipher.
- **encrypt**: This method encrypts the input text using the secret key and cipher object.
- **CreatingASignatureObject**: It's meant to demonstrate the creation of a signature object using SHA256 with RSA. However, it's not used.

## class TextDecryptor:

This class decrypts text that has been encrypted using symmetric encryption (AES) and encoded using Base64.

**decrypt**: This method decrypts the input Base64-encoded cipher text using the provided secret key and cipher object.

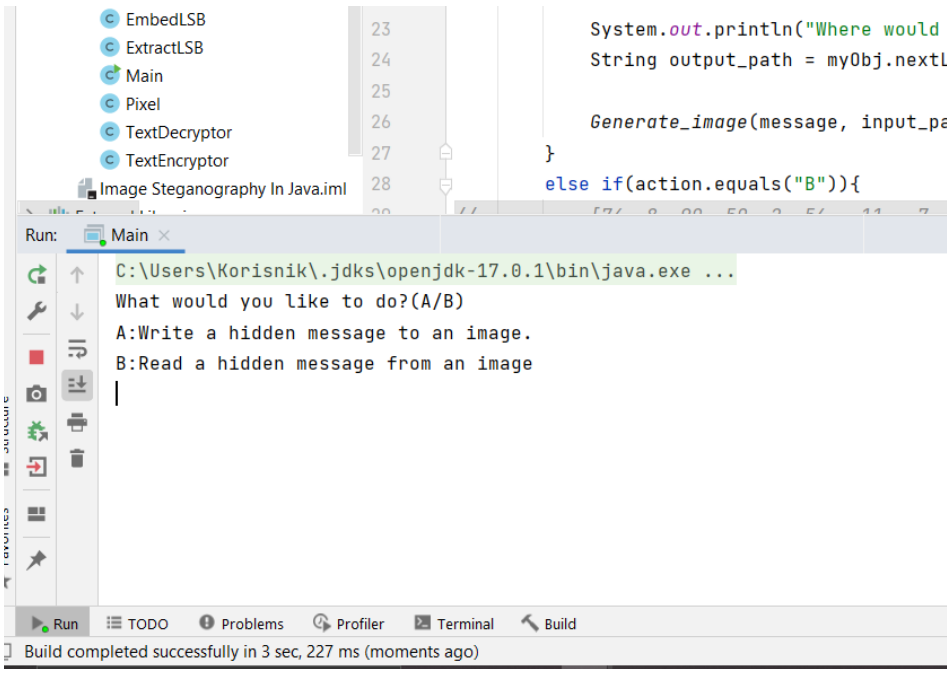
## Sources:

• Tutorial for the embedding text in images and extractiong text from images using LSB:
https://www.youtube.com/watch?v=e4xphUB0Ptg

• Tutorial for encryption of text:
https://www.tutorialspoint.com/java_cryptography/java_cryptography_encrypting_data.htm

• Tutorial for decryption of text:
https://www.tutorialspoint.com/java_cryptography/java_cryptography_decrypting_data.htm

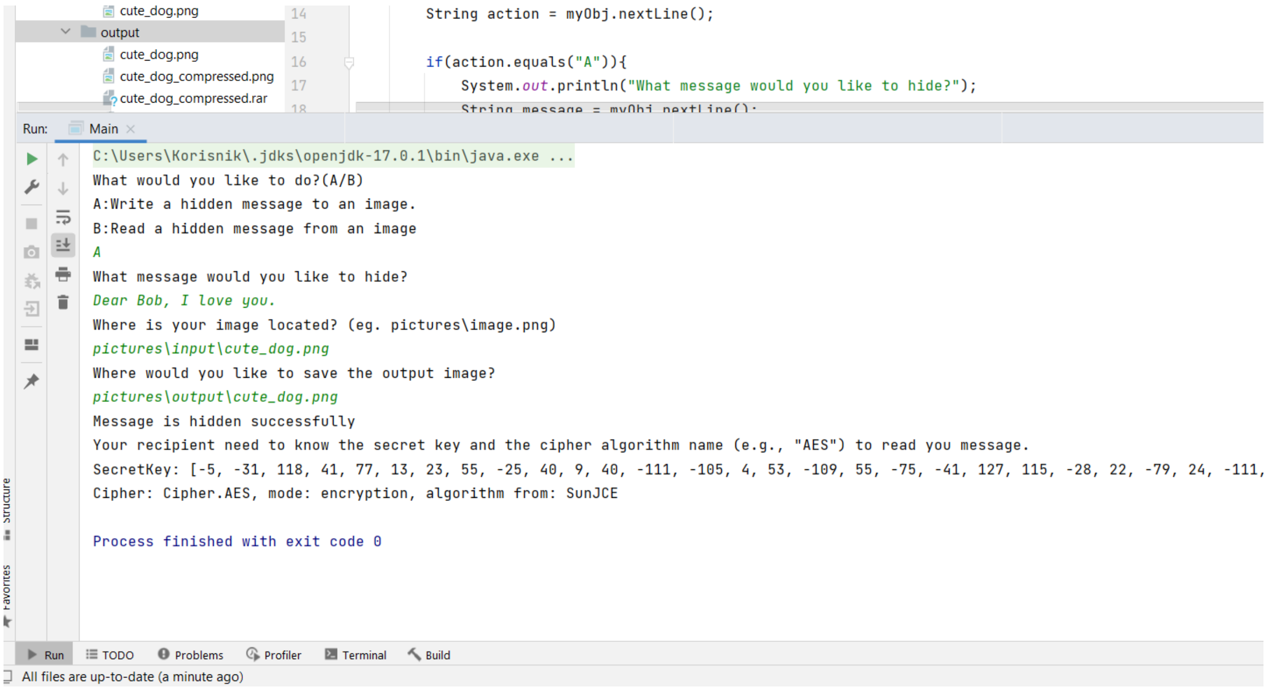
# Screenshots:


EmbedLSB
ExtractLSB
Main
Pixel
TextDecryptor
TextEncryptor
Image Steganography In Java.iml
Run: Main
Run TODO Problems Profiler Terminal Build

```
        System.out.println("Where would
        String output_path = myObj.nextL

        Generate_image(message, input_pa
    }
    else if(action.equals("B")){
```

```
C:\Users\Korisnik\.jdks\openjdk-17.0.1\bin\java.exe ...
What would you like to do?(A/B)
A:Write a hidden message to an image.
B:Read a hidden message from an image
```

Build completed successfully in 3 sec, 227 ms (moments ago)

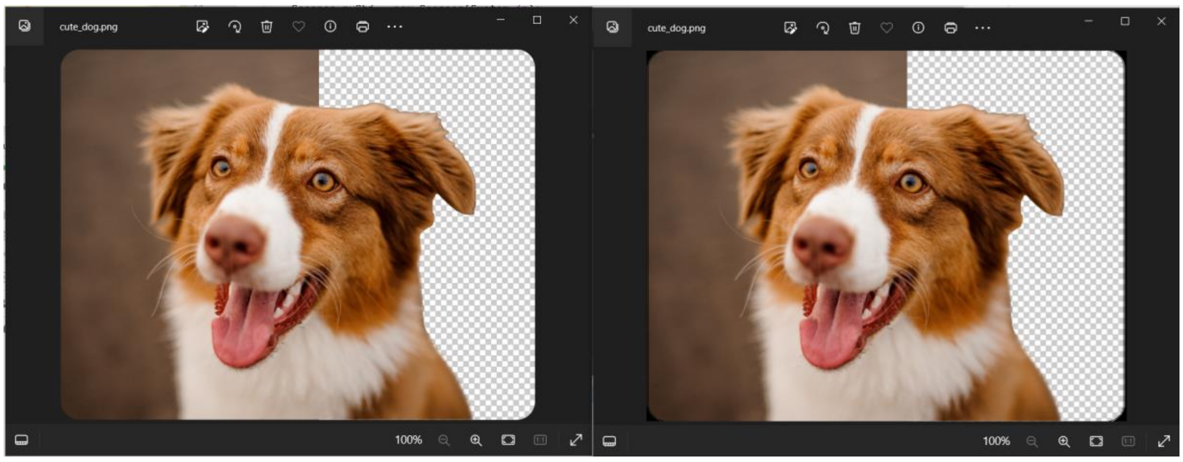


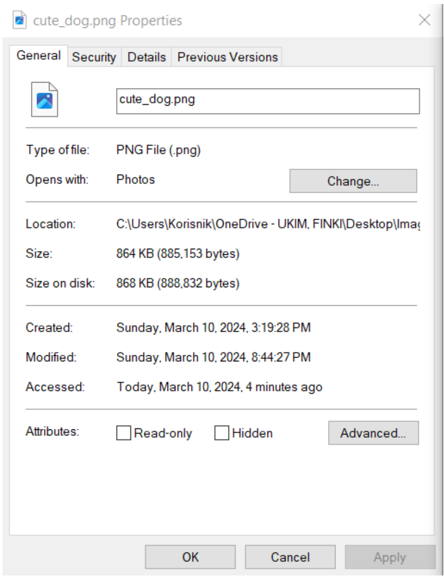
```
String action = myObj.nextLine();

if(action.equals("A")){
    System.out.println("What message would you like to hide?");
```

```
C:\Users\Korisnik\.jdks\openjdk-17.0.1\bin\java.exe ...
What would you like to do?(A/B)
A:Write a hidden message to an image.
B:Read a hidden message from an image
A
What message would you like to hide?
Dear Bob, I love you.
Where is your image located? (eg. pictures\image.png)
pictures\input\cute_dog.png
Where would you like to save the output image?
pictures\output\cute_dog.png
Message is hidden successfully
Your recipient need to know the secret key and the cipher algorithm name (e.g., "AES") to read you message.
SecretKey: [-5, -31, 118, 41, 77, 13, 23, 55, -25, 40, 9, 40, -111, -105, 4, 53, -109, 55, -75, -41, 127, 115, -28, 22, -79, 24, -111,
Cipher: Cipher.AES, mode: encryption, algorithm from: SunJCE

Process finished with exit code 0
```

All files are up-to-date (a minute ago)

## Original image vs image with hidden message



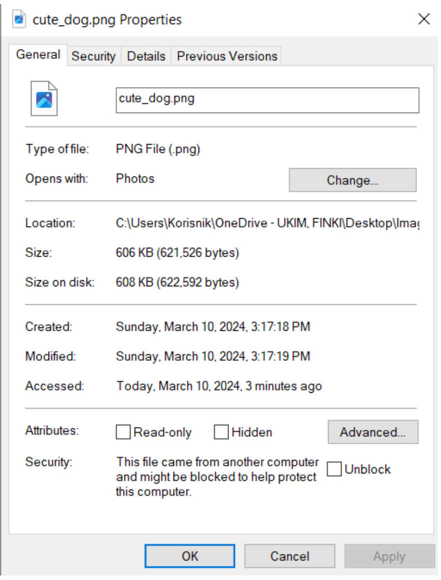


cute_dog.png Properties

General | Security | Details | Previous Versions

cute_dog.png

Type of file: PNG File (.png)
Opens with: Photos
Location: C:\Users\Korisnik\OneDrive - UKIM, FINKI\Desktop\Imag
Size: 864 KB (885,153 bytes)
Size on disk: 868 KB (888,832 bytes)
Created: Sunday, March 10, 2024, 3:19:28 PM
Modified: Sunday, March 10, 2024, 8:44:27 PM
Accessed: Today, March 10, 2024, 4 minutes ago
Attributes: Read-only Hidden



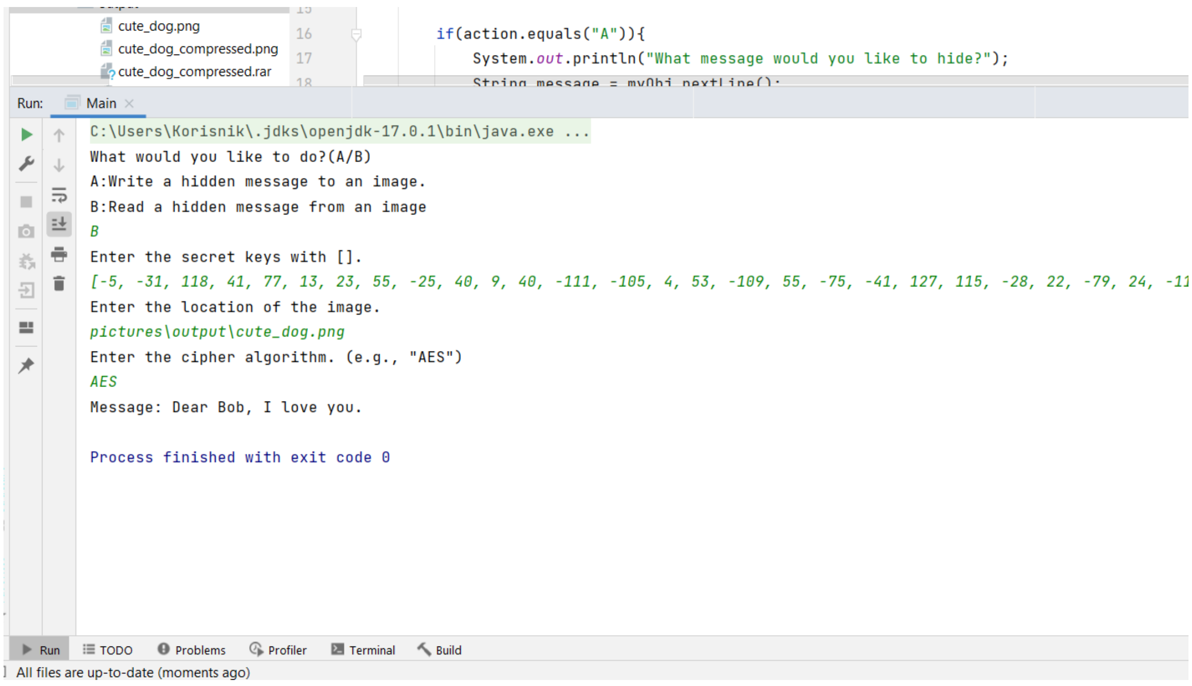
cute_dog.png Properties

General | Security | Details | Previous Versions

cute_dog.png

Type of file: PNG File (.png)
Opens with: Photos
Location: C:\Users\Korisnik\OneDrive - UKIM, FINKI\Desktop\Imag
Size: 606 KB (621,526 bytes)
Size on disk: 608 KB (622,592 bytes)
Created: Sunday, March 10, 2024, 3:17:18 PM
Modified: Sunday, March 10, 2024, 3:17:19 PM
Accessed: Today, March 10, 2024, 3 minutes ago
Attributes: Read-only Hidden
Security: This file came from another computer and might be blocked to help protect this computer. Unblock

The image with a hidden message is a little bigger than the orifginal image.



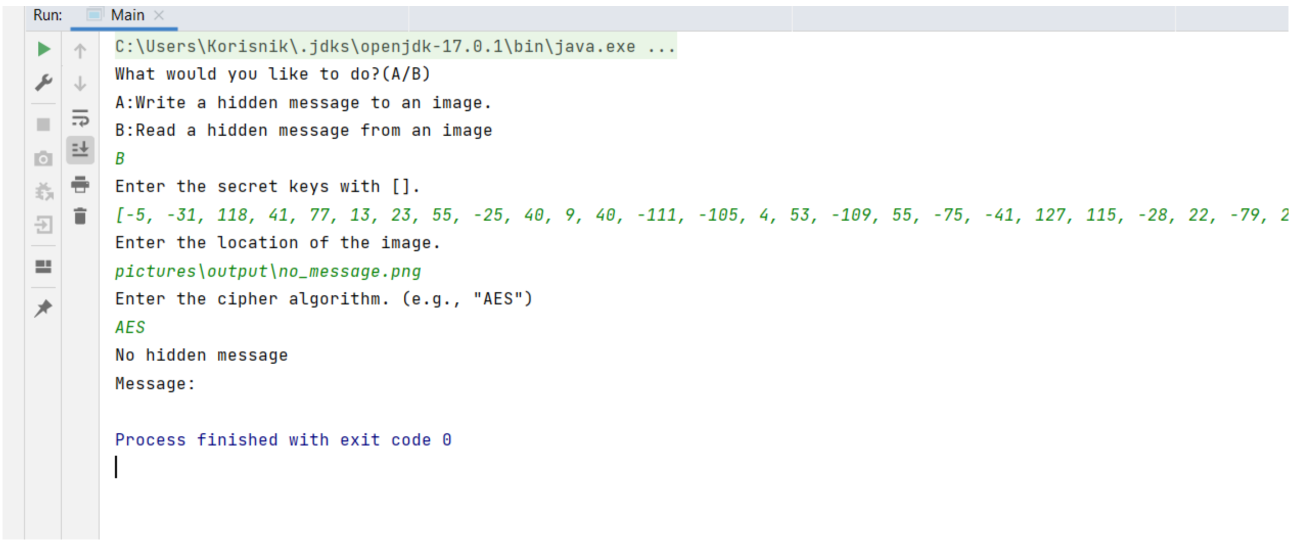
cute_dog.png
cute_dog_compressed.png
cute_dog_compressed.rar

```
        if(action.equals("A")){
            System.out.println("What message would you like to hide?");
            String message = myObj.nextLine();
```

Run: Main

```
C:\Users\Korisnik\.jdks\openjdk-17.0.1\bin\java.exe ...
What would you like to do?(A/B)
A:Write a hidden message to an image.
B:Read a hidden message from an image
B
Enter the secret keys with [].
[-5, -31, 118, 41, 77, 13, 23, 55, -25, 40, 9, 40, -111, -105, 4, 53, -109, 55, -75, -41, 127, 115, -28, 22, -79, 24, -11
Enter the location of the image.
pictures\output\cute_dog.png
Enter the cipher algorithm. (e.g., "AES")
AES
Message: Dear Bob, I love you.

Process finished with exit code 0
```

All files are up-to-date (moments ago)



Run: Main

```
C:\Users\Korisnik\.jdks\openjdk-17.0.1\bin\java.exe ...
What would you like to do?(A/B)
A:Write a hidden message to an image.
B:Read a hidden message from an image
B
Enter the secret keys with [].
[-5, -31, 118, 41, 77, 13, 23, 55, -25, 40, 9, 40, -111, -105, 4, 53, -109, 55, -75, -41, 127, 115, -28, 22, -79, 2
Enter the location of the image.
pictures\output\no_message.png
Enter the cipher algorithm. (e.g., "AES")
AES
No hidden message
Message:

Process finished with exit code 0
```