Vietnam National University Ho Chi Minh City

University Of Science

Advanced Program in Computer Science

Computer Security

Final Project - Pepetected

Student name:	*Student ID:*
Vinh-Khang Phan	20125096
Van-Hoang-Phi Le	20125106
Quang-Dung Dao	20125127

Contents

1 Introduction

Are you tired of hiding your top-secret messages in plain sight, only to have them discovered by prying eyes? Say goodbye to the age-old methods of discreetly passing notes or encrypting your texts with boring codes. It's time to embrace the future of covert communication with the Pepetected app!

Imagine a world where your confidential messages are hidden within the hilarious world of Pepes. Welcome to Pepetected, the app that turns your everyday conversations into a comedy show and a cryptic puzzle all in one!

With Pepetected, you can take your secret messages and wrap them up in a meme so funny that even if someone stumbles upon your chat, they'll be too busy laughing to decode your hidden gems. It's like having your own personal inside joke that's also a covert operation.

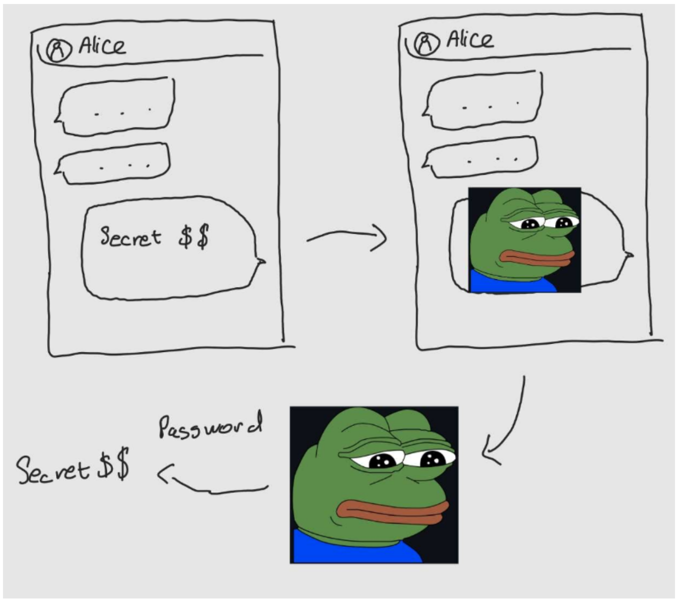
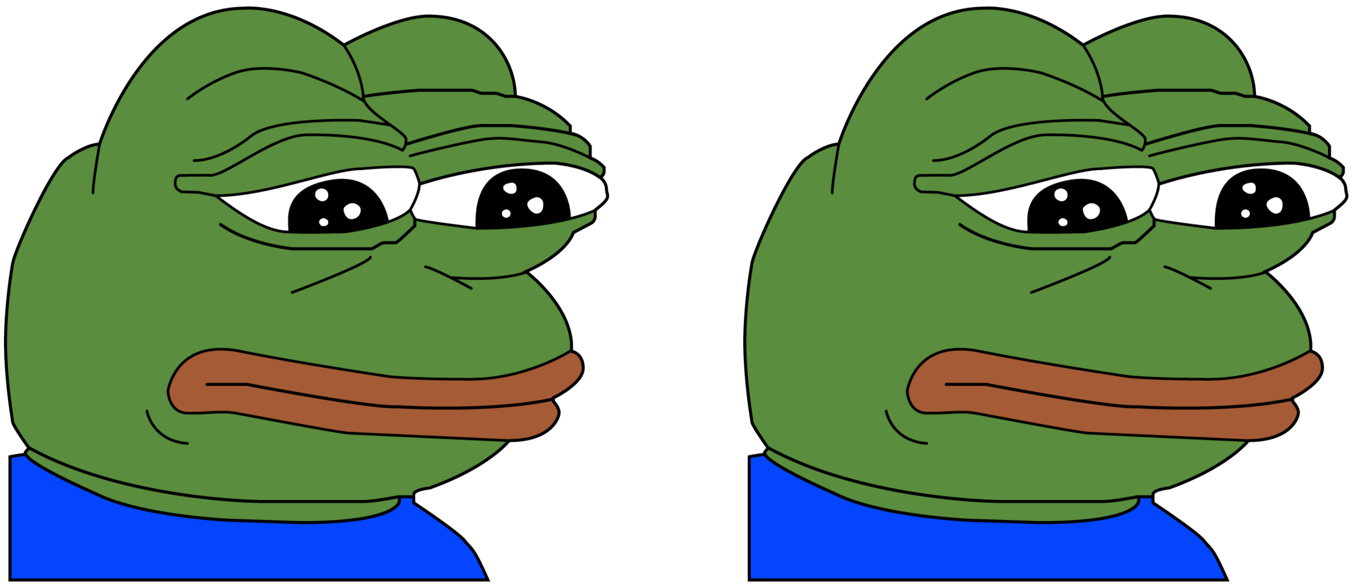


Figure 1: Overview

2 Features

Sometimes, there are some special messages that you want to keep private. Almost all popular messaging applications only allow you to delete the messages, and you need to store them manually, with many uncertainties about their security. Pepetected provides a solution, where you can replace your messages with Pepe (and any image, and text also), preventing your conversation log from interruption.

Do you think these 2 images are the same?

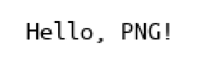


(a) Pepe (b) Encoded Pepe

Figure 2: Looks similar but not actually similar Pepes

S In fact, the right one is embedding ”Welcome to pepetected!” but it is only revealed if you enter the correct password!

We also provide a simpler version, in case you only want to hide by text only. In fact, this is just an image which fakes as a normal message.



Hello, PNG!

Figure 3: Text form

3 Method

Our application is designed to enhance the privacy and security of private messages by utilizing steganography and strong encryption techniques. Here’s a breakdown of the key components of our method:

1. Steganography with LSB Approach:

- We employ steganography to hide private messages within images using the Least Significant Bit (LSB) approach. This technique involves embedding message data within the least significant bits of the image's pixel values. This allows us to conceal the message within the image without significantly altering its appearance.

2. User-Entered Password and Bcrypt:

- To initiate the encryption process, the user enters a password of their choice. This password is crucial for both encryption and decryption.
- We validate and secure the user's password using the bcrypt hashing algorithm. Bcrypt is chosen for its robustness against brute-force attacks and its resistance to rainbow table attacks. It ensures that even if the password database is compromised, attackers face significant hurdles in cracking the passwords.

3. Encryption with PBKDF2 and AES256:

- After validating the password, we derive a strong encryption key from it using the PBKDF2 (Password-Based Key Derivation Function 2) algorithm. PBKDF2 adds an extra layer of security by making it computationally intensive to derive the encryption key from the password.
- With the encryption key in place, we use the AES256 (Advanced Encryption Standard with a 256-bit key) algorithm for message encryption. AES256 is renowned for its high level of security and is widely used in various applications. It provides strong protection against eavesdropping and unauthorized access.

4. Python Libraries - Bcrypt and Crypto:

- We implement the bcrypt hashing algorithm and the AES encryption scheme using Python libraries. Bcrypt ensures that password hashing is carried out securely, and the Crypto library provides the necessary tools for implementing AES encryption. Using established and well-maintained libraries helps ensure the reliability and correctness of our encryption methods.

Reasons for Choosing These Algorithms:

1. **Steganography (LSB Approach)**: The LSB approach is chosen for steganography because it provides a good balance between concealing the message and maintaining the image's visual integrity.
2. **Bcrypt**: Bcrypt is selected for password hashing due to its resistance to various attacks and its slow hashing process, which makes it difficult for attackers to crack passwords.
3. **PBKDF2**: PBKDF2 is utilized to derive an encryption key from the user's password because it is a widely accepted and secure key derivation function, suitable for protecting sensitive data.
4. **AES256**: AES256 is used for message encryption because it offers a high level of security and is considered a standard for symmetric key encryption. It ensures that the message is securely protected against unauthorized access.

By combining steganography with strong encryption techniques and secure password handling, our application prioritizes the privacy and security of user messages, making it a robust choice for secure communication.

4 Demo

The full source code can be found here

Encryption phase

```
    encoder = Encoder()

    password = "password1"
    password_stored_file = "sample/user/password.txt"
    cover_image = "sample/pepe.png"
    message = "Welcome to pepetected!"
    output_image = "sample/pepe_encoded.png"

    PasswordService.save_password(password, password_stored_file)

    encrypt = EncryptService()
    encrypted_message = encrypt.encrypt(message, password)

    # Encode image
    encoder.encode_image(cover_image, encrypted_message, output_image)
```

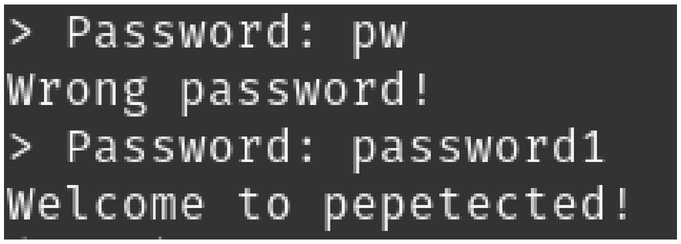
Output image

Figure 4: Output image

Decryption phase

```
    password_stored_file = "sample/user/password.txt"
    encoded_image = "sample/pepe_encoded.png"

    while True:
        print("> Password: ", end = "")
        password = input()

        if not PasswordService.validate(password, password_stored_file):
            print("Wrong password!")
            continue

        decoder = Decoder()
        encrypted_message = decoder.decode(encoded_image)

        decrypt = DecryptService()
        decrypted_message = decrypt.decrypt(encrypted_message, password)

        print(decrypted_message)

        break
```

Command line screen



```
> Password: pw
Wrong password!
> Password: password1
Welcome to pepetected!
```

Figure 5: Command line scren decryption