
Image Steganography Tool

Simple C++ **Encryption** and **Steganography** tool that uses Password-Protected-Encryption to secure a file's contents, and then proceeds to embed it inside an image's pixel-data using Least-Significant-Bit encoding. For Linux, MacOS, and Windows systems.

Encoding

```
1 $ ./steganography encode -i data/orig.png -e data/jekyll_and_hyde.zip -
  o output.png
2 Password: 1234
3 * Image size: 640x426 pixels
4 * Encoding level: Low (Default)
5 * Max embed size: 132.38 KiB
6 * Embed size: 61.77 KiB
7 * Encrypted embed size: 61.78 KiB
8 * Generated CRC32 checksum
9 * Generated encryption key with PBKDF2-HMAC-SHA-256 (20000 rounds)
10 * Encrypted embed with AES-256-CBC
11 * Embedded jekyll_and_hyde.zip into image
12 * Sucessfully wrote to output.png
```

Original image:



Image with embedded ZIP containing the entire contents of the book “Dr Jekyll and Mr Hyde”:



Decoding

```
1 $ ./steganography decode -i output.png -o "out - jekyll_and_hyde.zip"
2 Password: 1234
3 * Image size: 640x426 pixels
4 * Generated decryption key with PBKDF2-HMAC-SHA-256 (20000 rounds)
5 * Successfully decrypted header
6 * File signatures match
7 * Detected embed jekyll_and_hyde.zip
8 * Encoding level: Low (Default)
9 * Encrypted embed size: 61.78 KiB
10 * Successfully decrypted the embed
11 * Decrypted embed size: 61.77 KiB
12 * CRC32 checksum matches
13 * Successfully wrote to out - jekyll_and_hyde.zip
```

Building

```
1 $ mkdir build
2 $ cd build
3 $ cmake -DCMAKE_BUILD_TYPE=Release ..
```

```
4 $ make -j 4
```

Usage

```
1 Usage: steganography [-h] {decode,encode}
2
3 Optional arguments:
4   -h, --help      shows help message and exits
5   -v, --version    prints version information and exits
6
7 Subcommands:
8   decode          Decodes and extracts an embed-file from an image
9   encode          Encodes an embed-file into an image
```

Encoding

```
1 Usage: encode [-h] --input VAR --output VAR --embed VAR [--passwd VAR]
2
3 Encodes an embed-file into an image
4
5 Optional arguments:
6   -h, --help      shows help message and exits
7   -v, --version    prints version information and exits
8   -i, --input      specify the input image. [required]
9   -o, --output      specify the output image. [required]
10  -e, --embed       specify the file to embed. [required]
11  -p, --passwd      specify the encryption password.
```

Decoding

```
1 Usage: decode [-h] --input VAR [--output VAR] [--passwd VAR]
2
3 Decodes and extracts an embed-file from an image
4
5 Optional arguments:
6   -h, --help      shows help message and exits
7   -v, --version    prints version information and exits
8   -i, --input      specify the input image. [required]
9   -o, --output      specify the output file. [default: ""]
10  -p, --passwd      specify the encryption password.
```

Theory Of Operation

Encoding

The program operates by first randomly generating a *128-bit Password Salt* and a *128-bit AES Initialization Vector* by reading binary data from `/dev/urandom`. It then uses that *Password Salt* as a parameter in generating an encryption key, by using **PBKDF2-HMAC-SHA-256** on a user inputted string. A **CRC32** hash of the file to embed is then calculated, and stored in the header to act as a checksum for the validity of the data. It then pads the binary data of the file to embed using the **PKCS #7** algorithm, followed by actually encrypting both the header and the padded data, with **AES-256** in **CBC Mode**, using the previously generated *Initialization Vector*. Now the data is actually encoded inside the image by first picking a random offset, and then going through each bit of data and storing it inside the actual image pixel data, which it accomplishes by setting the *Least-Significant-Bit* of each channel byte of each pixel.

Decoding

The decoding process works exactly the same as the encoding process previously described above, just in reverse. The only difference is that for decoding, after the program attempts to extract and decrypt the data, it compares some of the information in the header section in an attempt to validate the extraction process. The header fields which are compared are: The 4 byte file signature custom to this program, and the **CRC32** hash of the decrypted data. If any of these fields do not match to their correct values, the decryption process will fail. This should only happen if the file which you were attempting to decrypt does not actually contain an embed, if the password you entered is wrong, or if the image file was somehow corrupted.

Detection

While the detection of data being embedded in an image is a trivial task, theoretically there is no way of knowing that it was this program that did it, and theoretically there should be no known way to decrypt the data without knowing the password, that is without spending millions of years in the process of doing so.

Disclaimer

Do not use this program to encrypt and hide important data which you wish to keep away from prying eyes. This is just a simple proof-of-concept program that I made for fun. I'm no cryptographer. I'm just a hobbyist, use at your own risk.

Copyright

This software is licensed under MIT. Copyright © 2022 Zach Collins