HIDING DATA IN IMAGE USING STEGANOGRAPHY WITH TWO-LEVEL SECURITY AND USING WEBASSEMBLY

JAMES STEPHEN MEKA Principal, WISTM, Andhra University, (jamesstephenm@gmail.com)
KASI VENKAT KIRAN, Associate Professor,WISTM (kasi.kvk@gmail.com)

Abstract

Users of digital communication can send digital data from one location to another across the internet. Intruders may be able to retrieve this data. As a result, some form of data security is required before the data leaves the sender's location. Data is protected using techniques such as cryptography and steganography. Cryptography encrypts the data, but it also discloses its presence Steganography conceals the existence of information so that anyone other than the sender and recipient is unable to recognise it. However, only standalone software that is platform-dependent is available to do steganography. Web apps have the disadvantage of requiring a browser to run and a server to accomplish heavy computational operations (like image processing or video processing, which is very slow or not possible if accomplished alone on a browser). And, because of intruders, transporting data from the user's end to the server for processing is not completely secure. To address this issue, we propose leveraging web assembly to perform cryptography, steganography, and other functions on the client-side without transmitting any data to the server.

Keywords: digital communication, Steganography**,** Cryptography, WebAssembly.

1. Introduction

In the present era, the Internet has become a convenient means of data transfer, therefore information security is one of the most pressing concerns. To keep the communication secret, a variety of methods for encrypting and decrypting the secret data have been created. Steganography is the technique of hiding data within digital media, and can also be used for watermarking. The goal is to put concealed data into the form of media for identification and copyright protection. The key constraints of this

The procedure is the quantity of message data, the requirement for invariability of embedded data under distortions such as lossy compression, and the removal or modification of third parties. The three types of data hiding techniques are cryptography, steganography, and watermarking. Cryptography makes data meaningless, whereas watermarking and, in particular, steganography conceals the presence of secret data. The digital representation of media improves data accuracy, efficiency, and mobility while also making it easier to access. Copyright violations and content modification or manipulation, on the other hand, have negative consequences. Intellectual property protection, content manipulation indication, and annotation are the key motivations for employing these tactics. However, to do steganography (using the LSB technique), there are only standalone programmes that are platform-dependent, and implementing them for many systems is expensive and time-consuming when compared to Web applications. The challenge with web apps is that they require a browser to function and a server to perform complex calculations (like image processing or video processing, which is very slow or not possible if accomplished alone on a browser). And transmitting data from the user's end to the server for further processing implies there's a danger it would be captured by attackers, which is a major deterrent to users considering online applications in the first place, and indeed steganography. To resolve this concern, we propose leveraging web assembly to accomplish cryptography, steganography, and other functions on the client-side without transmitting any data to the server. By using WebAssembly, we may attain the processing speed of a native application in a web-based application. The program is created in compiled languages such as Go, C, C++, Rust, Kotlin, C #, F # Swift, and is compiled down to a web assembly, which we can utilize straight in our web application using web assembly JS Linker code. In this paper, we used the Go language to develop code for image steganography with two-level security and compiled it to WASM code using the command 'GOARCH=wasm GOOS=js go build -o lib.wasm main.go'. Finally, we are using the wasm exec.js linker file provided by the Go language to link wasm code to js, and now the global function written in the Go programme is available for use in Js Script.

2. Literature Review

WebAssembly is a low-level assembly-like language with a compact binary structure that operates at a near-native speed and serves as a compilation target for languages like C/C++, C#, and Rust to run on the web. It's also meant to work alongside JavaScript, so the two may interact. To use web assembly, we first use functional programming to implement the algorithm and logic (in our scenario, LSB for steganography, AES for encryption, and LZW for data compression) in the programming languages that support web assembly (Functional programming is a style of programming in which pure functions are used to solve the problem).

WebAssembly was intended to complement and run alongside JavaScript; developers can import WebAssembly functionalities into a JavaScript application and exchange functionality between the two using the WebAssembly JavaScript APIs. This helps to integrate the speed and reliability of WebAssembly with both the expressiveness and versatility of JavaScript in the same project, although you do not know how to implement WebAssembly code. Importing WebAssembly modules into a web (or Node.js) application exposes WebAssembly functionalities for use through JavaScript. WebAssembly can be used by JavaScript frameworks to continue providing significantly better performance and functionalities while preserving functionality approachable to web developers.

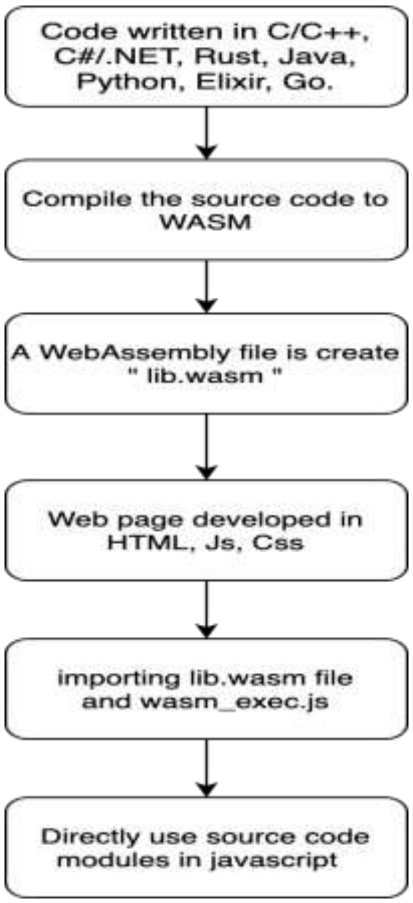

Code written in C/C++, C#/.NET, Rust, Java, Python, Elixir, Go.
Compile the source code to WASM
A WebAssembly file is create " lib.wasm "
Web page developed in HTML, Js, Css
importing lib.wasm file and wasm_exec.js
Directly use source code modules in javascript

Fig 2 Workflow of WebAssembly

Example:

C code:

```
int square(int num1) {
  return num1* num1;
}
```

WebAssembly code :

```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "square" (func $square))
 (func $square (; 0 ;) (param $0 i32) (result i32)
  (i32.mul
   (get_local $0)
   (get_local $0)
  )
 )
)
```

Js code :

```
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new WebAssembly.Instance(wasmModule, wasmImports);
log(wasmInstance.exports.square(42));
```

Output:

```
1764
```

3. Existing System

In the existing system, we be using standalone programmes to accomplish steganography with cryptography, which is written in programming languages such as (java, c++, python, and many others), and several of them are platform-dependent; to make them cross-platform, we could perhaps write code in platform-specific languages and develop mobile apps. And, updating or adding modifications regularly, should be done independently for each platform, which is an expensive and time-consuming operation.

And using a web application for this data-sensitive operation is not a smart option because the data should not leave the browser for processing. After all, the network could've been monitored by an attacker or intruder, or the service provider could record all the data without the proper authorization.

4. Proposed System

We propose implementing web assembly for client-side data processing to solve this problem. To ensure that the user trusts us, we use WebAssembly to perform client-side image processing, encryption, and data compression without exposing data to the server. We may conclude that the data is secure and private to the user solely because it is not routed from the user to the server for any data processing and all operations are performed at a noticeable speed on the user system. Since various devices have different computational resources, the developed web application can run on multiple platforms, allowing users to utilize it on as many devices as they require. However, performance may fluctuate from device to device.

Furthermore, the application may easily receive updates and fixes from the developer without having to worry about the overhead of updating numerous platforms or applications developed in multiple frameworks.

5. Methodology

In this proposed system there are two main methods for encoding secret data which is encrypted using the password in the image and decoding the secret data from the image using a password which is used in the encoding step. These two methods were written in Go, a compiled language, and then compiled to Web assembly.

5.1 Encoding

First of all the application should be provided with an image and message with the password. The message is first encrypted using the AES (Advanced Encryption Standard) algorithm and then the encrypted text is compressed using the LZW (Lempel-Ziv-Welch) algorithm because we can’t store a message of more than 12% size of the image if we are using an 8-bit image. So, we are using LZW which is a simple, and efficient lossless data compression algorithm which is used widely because of its simplicity and robustness. Then the data is encoded into an image using LSB (Least significant bit) algorithm which is used widely and effectively for 8-bit images. This algorithm uses the last bit of every byte (8 bits which are one pixel of an image) to store the data. After that, a new image is created which is identical to the image which was given before. The image is provided to the user for downloading and it contains the message. To get the message back the user should decode the image with the password.

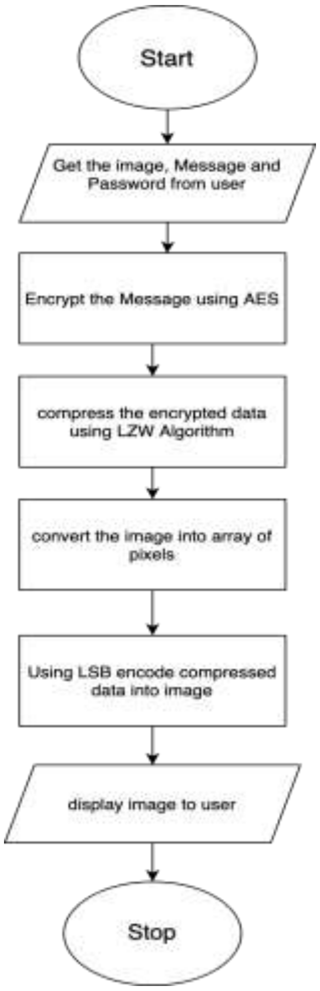

Start
Get the image, Message and Password from user
Encrypt the Message using AES
compress the encrypted data using LZW Algorithm
convert the image into array of pixels
Using LSB encode compressed data into image
display image to user
Stop

Fig 5.1(a) encoding workflow

5.2 Decoding

It is the inverse of the encoding process. The image provided is first decoded using the LSB algorithm, and thereafter compressed information is obtained, which is then decompressed using the LZW algorithm. Finally, an encrypted text is obtained, which would be decrypted using the key provided by the user, and the data is delivered to the user.

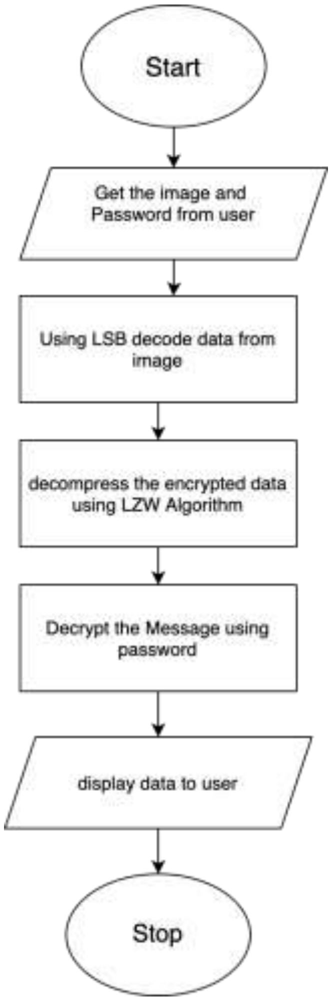
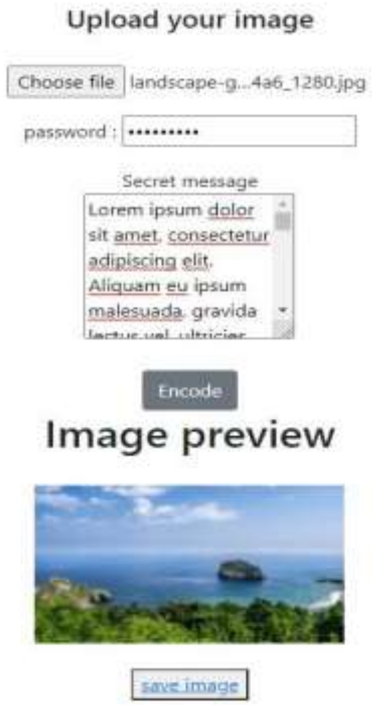

Start
Get the image and Password from user
Using LSB decode data from image
decompress the encrypted data using LZW Algorithm
Decrypt the Message using password
display data to user
Stop

Fig 5.2(a) decoding workflow

6. EXPERIMENTAL RESULTS

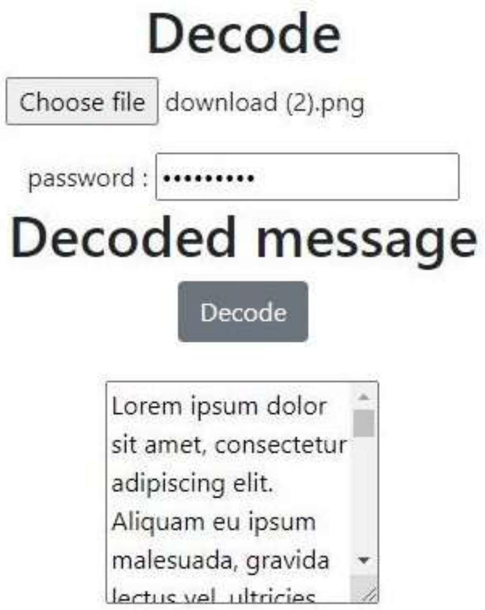
The algorithm is written in the Go programming language, compiled to WASM, and run on a Windows 10 platform. Several color photos are used to demonstrate the process. We used 577 words, 3930 bytes of Lorem Ipsum English text, and a 1280X853 "landscape" image in our tests.

Figure 6.3 shows a 1280X853 original image, and we can hide 31440 bits in the original image (see Figure 6.3). The proposed techniques are used to insert the stego image shown in Figure 6.4. Because the pixels in a "landscape" image are roughly uniformly dispersed, it is extremely difficult to discern hidden data with the human eye.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam eu ipsum malesuada, gravida lectus vel, ultricies

Fig 6.1 encoding image



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam eu ipsum malesuada, gravida lectus vel, ultricies

Fig 6.2 decoding image

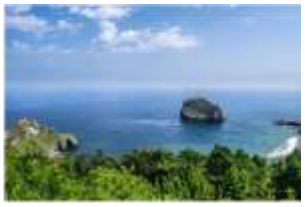


Fig 6.3 original image Fig 6.3 encoded image

Other comparison graphs of the original and stego image histograms can be seen here.

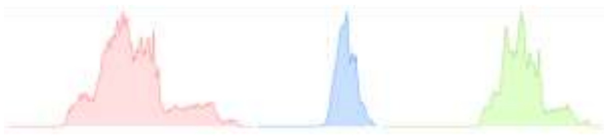


Fig 6.4 histograms of the original image

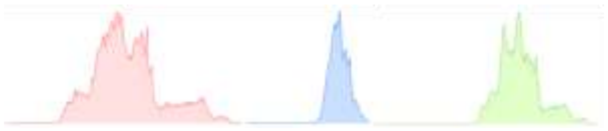


Fig 6.4 histograms of the original image

7. Conclusion

We use image steganography in this proposed work, which is a technique for transmitting secret information by concealing it under a covering material. Before concealing secret data beneath the cover image, the data is encrypted using AES for additional security and then compressed with LZW. Lossless compression is achieved using the LZW approach. Data compression is the process of transforming a stream of symbols into codes. Lossless compression is implemented in proposed approaches to reduce the size of the secret message while maintaining quality. When information is not lost even after decompression and 100 per cent of the information from compressed data is recoverable. The intruder would have to put in a lot of effort to reconstruct the original message, which would be unachievable as long as the exact encryption key is known. The AES algorithm will be used for data encryption of compressed data. This technology adds a second layer of security to information. Both the sender and the receiver must be aware of the secret keys. The Least Significant Bits (LSB) algorithm is used for steganography, which involves replacing the image's least significant bits with information bits. While the encryption process increases the procedure's complexity, it also increases its security. This is a straightforward method. This approach replaces the least significant bits of some or all of the bytes inside an image with bits of the secret message. And to achieve this, we use web assembly to process data on the client-side without sending it to the server. The World Wide Web Consortium (W3C) developed WebAssembly, or WASM for short, and released it in 2018. WebAssembly is a "compilation target," which means that developers can provide their code—typically Rust, C++, or AssemblyScript—and WebAssembly compiles it to bytecode for high-speed execution in the web browser.

8. References

[1] Min Wu; Tang, E.; Lin, B., "Data hiding in digital binary image," Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on, vol.1, no., pp.393-396 vol.1, 2000.

[2] M. Al-Shatnawi, “A new method in image steganography with improved image quality” Appl. Math. Sci., Vol. 6, 2012, no. 77-80, 3907-3915.

[3] Abboud, G.; Marean, J.; Yampolskiy, R.V."Steganography and Visual Cryptography in Computer Forensics," Systematic Approaches to Digital Forensic Engineering (SADFE), 2010 Fifth IEEE International Workshop on, vol., no., pp.25-32, 20-20 May 2010.

[4] Khalil challita, Hikmat Farhat, “combining steganography and cryptography: new directions,” IJNCAA, Vol 1, 2011, pp 199-208.

[5] An Introduction to Image Steganography Techniques. 2012 International Conference on Advanced Computer Science Applications and Technologies

[6] Mohammed Abdul Majeed and Rossilawati Sulaiman, “An improved LSB image steganography technique using bit-inverse in 24-bit color image”, Journal of Theoretical and Applied Information Technology, pp 342-348, 2015.

[7] Vipul Sharma and Madhusudan, “Two new approaches for image steganography using cryptography”, Third international conference on Image Information Processing (ICIIP), 2015.

[8] Principal Joseph and S. Vishnukumar, “A study on steganography techniques”, 2015 Global Conference on Communication Technologies (GCCT), 2015.

[9] Rupali Jain and Jayshree Boaddh, “Advances in digital image steganography”, International Conference on Innovation and Challenges in Cyber Security, 2016.

[10] Nidhi Menon and Vaithiyanathan, “A survey on image steganography”, International Conference on Technological Advancements in Power and Energy, Dec 2017.

[11] D. Herrera, H. Chen, E. Lavoie and L. Hendren, WebAssembly and JavaScript Challenge: Numerical program performance using modern browser technologies and devices, Montreal: QC: University of McGill, 2018.

[12] S. Padmanabhan and P. Jha, "WebAssembly at eBay: A Real-World Use Case", 2020, [online] https://tech.ebayinc.com/engineering/webassembly-at-ebay-a-real-world-use-case/.

[13] "WebAssembly vs. the world. Should you use WebAssembly?", 2018, [online]Available: https://blog.sqreen.com/webassembly-performance/.

[14] Lempel Ziv Welch compression and decompression, [online] Available: https://iq.opengenus.org/lempel-ziv-welch-compression-and-decompression/

[15] WebAssembly Studio, [online] Available :https://webassembly-studio.kamenokosoft.com/

[16] "WebAssembly Memory", 2020, [online] Available:https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Globalobjects/WebAssembly/Memory.

[17] "Webassembly Use Cases", 2020, [online] Available:https://webassembly.org/docs/use-cases/.

[18] Steganography: Hiding an image inside another, [online] Available: https://towardsdatascience.com/steganography-hiding-an-image-inside-another-77ca66b2acb1